

# Typing Meetup Notes

## September 2023

Eric Traut, Rich Chiodo, Stricter TypeGuard ([PEP 724](#))

[Slides](#)

[Recording](#)

[Chat](#)

Jelle Zijlstra, [Proposal](#) to create a Typing Council

## August 2023

Nikita Sobolev, Inline Syntax for TypedDict

[Slides](#)

[Recording](#)

[Chat](#)

Notes:

- What are some real-world use cases where we would want an inline TypedDict instead of a named TypedDict?
- To what extent would people want to reuse TypedDict fields (“inheritance”)?
- Nesting: Should a nested TypedDict be written as a raw dictionary literal or be wrapped with `TypedDict[{...}]`?
- How feasible are generic inline TypedDicts at runtime?

## PyCon Typing Summit: April 20, 2023

Jelle Zijlstra, The State of Typing 2023

[Slides](#)

[Recording](#) (this one unfortunately starts from the middle of the talk)

## Alex Waygood, Common Typing Pain Points from User Questions

[Slides](#)

[Recording](#)

Someone: What about class decorators? And could better error messages help here?

Alex: Definitely!

Shantanu: Better error messages can help

Rebecca: Anecdote: People don't understand TypeVar, but think they do.

Carl: Same thing. Type checkers don't always error when a typevar is used in a weird way.

Jelle: Maybe new syntax will help?

Steven: Experienced users from other languages will be helped by PEP 695.

Shantanu: I think PEP 695 will help with some problems, but maybe will also lead to people reaching for it unnecessarily.

Pradeep: People are like, why didn't the type checker catch this? People don't know about PEP 484 mandating unannotated functions aren't checked.

?: Often people don't annotate `__init__` and then everything is untyped.

?: Type system is rich enough that there are undecidable types. Some article about Python typing being Turing-complete.

Shantanu: Nobody in mypy actually does this Turing-complete stuff. Doesn't actually cause issues.

Jelle: Maybe PEP 484 was wrong in not type checking unannotated function.

Guido: Might be different between IDEs and type checkers.

Guido: Every type checker is different. Not just common cases, but also corner cases. That can confuse users.

Alex: Maybe because of the sample I looked at.

Shantanu: Default values for generics. Pyright handles this better than mypy.

?: Common issue: hard to become fully typed in a reasonable amount of time. Interaction between untyped and fully typed code is hard.

stump: Old SO questions: Did you find any incorrect or outdated answers? Or are popular answers good?

Alex: Some are outdated, like pre-PEP 673 answers about Self.

Guido: Confusion between runtime and type checker behavior? "But this `isinstance()` works!"

Alex: In some SO questions, some of the big misconceptions are around expecting types to be enforced at runtime.

"int is not a Number": mismatch between numbers and collections.abc ABCs.

Works because of typeshed lies.

Guido: It's done with protocol.

Alex: No Sequence is not.

Samuel: I think `isinstance()` should be really dumb.

Alex: Yes `isinstance()` checks on ABCs are really slow. Maybe we shouldn't have done ABCs with `isinstance()`.

Carl Meyer, PEP 649 and runtime `__annotations__`

[Slides](#)

[Recording](#)

Guido: Are you a coauthor?

Carl: No, would have written it differently. Larry can be the author.

Jukka: Will this solve all problems?

Carl: It's scary. It does a good job. Technique of regenerating might not work with flow control. `TypeVarTuple` caused problems. New features might not work well.

Some discussion about PEP 692. Jelle: No new syntax.

Jelle: Docs syntax might need another option.

Carl: Yes, I might have done that differently.

Guido: 1 an an arg to `__annotate__`?

Carl: VALUE/FORWARDREF/SOURCE options get passed down to `__annotate__`. That's primarily to support wrappers. Things like `functools.partial`, which want to mutate annotations before returning them.

1 is the value of `inspect.VALUE`

## Samuel Colvin, Using Type Hints at Runtime

[Slides](#)

[Recording](#)

Jelle: Several things you mention could be done in typing.

Samuel: Recursive generic models are hell.

Shantanu: PEP 681? `dataclass_transform`

Samuel: Want a way to disable `dataclass_transform` stuff in child classes.

Guido: PEP 695 TypeVar syntax. Is this going to cause issues?

Samuel: Apologies for not looking at PEPs.

Jelle: 3.6/3.7 dropping. `typing_extensions`?

Samuel: `typing_extensions` is great. Had to reimplement `get_typed_hints`. 3.6/3.7 allowed us to drop some code.

Pradeep: `TaggedUnion`? What is the issue?

Samuel: We have a runtime equivalent. Mypy has issues.

Jukka: Maybe can fix it in mypy.

Samuel: Union of lots of `TypedDict`s with lots of literal keys to discriminate.

Jukka: Adding anything to typing is generally difficult.

Jelle: How does this work? It's just a hint to the type checker to how to check against this type.

Samuel: Also helps with better type checker errors.

Pradeep: Maybe type checkers can do this automatically.

Samuel: With inference, you can't know you're right. You still have to show all the types.

?: Simplest case is when function returns a union and you return a literal dict, you have to go through the whole dict.

## Kevin Millikin, Specification of the Type System

Slides (TODO)

[Recording](#)

Jelle: Not sure the problem is as big as you make. Want a lightweight process.

Pradeep: Conformance test suite should be useful

Shantanu: Agree with test suite. Also, end user docs could be useful, and compete for the same resources.

Jukka: Overall spec is too much work. Test suite is useful. Focus on areas of disagreement between tools.

e.g., subtyping is central to the type system, but mostly settled. Focus on specific areas of benefit.

Kevin: Need some framework to put things together.

Samuel: Great idea, but not volunteering to write it. Valuable to have everything together, even if it's stapled together.

Improving the docs: would this improve docs? It's more valuable as end-user documentation.

Kevin: My doc is intended for implementers, not end users. It's a base for a tutorial, but not meant for end users.

Guido: C++ users don't read the C++ standard.

Rebecca: New features are only proposed by the language team in practice. I like that PEPs come from a variety of sources, not just implementers but also end users.

Kevin: I don't think this needs to change. We do hear of cases where people have a good idea, but some people disappear before writing a PEP.

Some recent PEPs explicitly leave out some features because it's too complicated.

Steven: Channeling Jia. He strongly agrees that the complexity of the language leads to consistency features.

Shantanu: Curious how many pages PEP 612 will take.

Pradeep: What would the test suite look like.

Kevin: Don't know how to write a conformance test without a spec.

Jelle: typeshed has tools for writing such a conformance suite.

Shantanu: You said you wrote a typechecker. Why?

Kevin: It runs in a language server inside an IDE. Different design tradeoffs from pytype.

## Carl Meyer, Adding Type Annotations using MonkeyType

[Slides](#)

[Recording](#)

?: Considered trying to generate data to decide if a supertype is better?

Carl: We don't. Falls in the category of making monkeytype not very smart.

?: How do we run MonkeyType on prod data?

Carl: It's in the docs. There's a programmatic API to turn on MT in a process.

Alex: Does MonkeyType have any behavior for cases where runtime values disagree with what it sees.

Carl: It's configurable. By default, we don't override existing annotations. Also a mode to happily overwrite anything. No mode to decide if they're compatible.

Jukka: Did MonkeyType affect the way engineers wrote types? Did they prefer more abstract types? At Dropbox, people used a lot of NewTypes.

Carl: It did. We had too many lists and not enough Sequences. Also had a codemod that replaced List with Sequence. If the type checker was OK it would commit it.

Rebecca: Did you ever get human pushback on that automated process?

Carl: No. We didn't do return types though.

?: What was the situation when you had 0% types? Did people want it but not want to do the work, or did people come on board over time?

Carl: Everybody wanted more types. With a very very large codebase, the benefits of types become very obvious. Also, lots of devs coming from other typed languages, especially Hack.

Guido: Was this before or after Python 3?

Carl: Python 3 first, types second.

Samuel: Does IG use Django? Did that cause problems?

Carl: "use Django" was very limited. Had our own ORM. Codebase so large that internal network effect was more important than third-party libraries. Biggest interface was with data layer API. Getting that typed was a priority.

Carl: Design was more suitable for types. In untyped Python it's tempting to build low-code, highly flexible solutions. In typed Python you get more codegen.

## Steven Troxler, Type Coverage Tooling

[Slides](#)

[Recording](#)

Jukka: Any estimates how much you can get out of these tools?

Steven: pyre infer only a few %. Haven't had much success with mypy and pytype because of type checker mismatches. Maybe 10% from 50-70% is the best case. ML models may do better. Hard to give upper bound. ML models can use variable names.

Pradeep: You can use these tools even without MonkeyType, e.g. without a production server. We run a codemod weekly that adds more types based on existing.

Jelle: autotyping has a tool for adding annotations based on variable names, based on work by Zac HD.

Guido: tool by someone called Leo.

Samuel: Could this tool be used for the stdlib.

Jelle: The hard question is whether we want to add types to the stdlib, not how to do this.

Steven: Stdlib has a lot of eyes on it.

## Steering Council Panel, moderated by Guido van Rossum

[Recording](#) (Starts about half an hour in)

Guido: Introductions?

Thomas: 4 years on SC, 3.12 RM

Emily: Newest member, organized pycon US for years, run a consulting business

Pablo: SC for 3 years. 7 years as core dev. RM for 3.10 and 3.11. Bloomberg for linkers.

Thomas: 2 others are Greg and Brett. Greg arriving tonight. Brett is next door talking about webassembly.

Brett is coming in 5 min.

Guido: Last year SC rejected PEP 677 (callable syntax). This year you accepted PEP 695 (TypeVar syntax). What was different?

Pablo: Lot of conversation for both. Major concern with arrow syntax was hard to read. Cost of readability and runtime effect was too big for benefit to typing. We think that generics is not as intrusive syntactically speaking. Soft keyword is delicate but that's OK. Balance of cost on the language vs. typing benefit.

Emily: Weird place where this benefits typing but we're considering it. We recognize typing is very important part of the ecosystem.

Thomas: Contrast. Both invasive syntactic changes. One was more common and one more esoteric. Callable only benefited one specific place. PEP 695 clearly improves the situation around defining TypeVars in all respects, making something that's currently awkward.

Brett: I also posted on Discuss about this.

?: What is the purpose of typing in Python? Prevent errors from developers? Runtime validation?

Guido: Can also be used for docs.

Brett: I view typing as a way to help tooling. Great for linters, great for pydantic. I work for vscode, so find it great for autocomplete. Great for docs. A way for the language to assist tools in doing \*something\*.

Thomas: I think of it as a way to describe the intent of the semantics of the code. It shouldn't change the meanings of the code, but elaborate on the intent. But that doesn't work well with pytype, which uses a different model. Maybe it doesn't matter, we just have to live with what people are using it for. Like Python itself.

Pablo: All of these cases are valid. Initially typing was thought of as static only, but now it's also used heavily at runtime. Not as vocal initially, but a huge use case. My own view is similar to Brett's, but other use cases are valuable too.

Emily: It's there to be a tool. An important supporting tool should also be used in many places.

Guido: Followup. Could there ever be some category of use of Python that may be in conflict with other uses, where we say some use case is out of scope?



Brett: Yes, because you set this precedent. Like annotations, where Larry wanted to use them for something else, but Guido decided they were for type hints. So yes, maybe we will make a similar decision in the future.

Pablo: It's not just static vs dynamic typing, but also untyped Python vs other stuff. There could be a place of too much cost for a feature.

Emily: The goal would be to preserve existing behavior that people depend on. Deprecations are hard. Every time we accept/reject a PEP, it's a microdecision that something is or is not going to be supported. We may have to sometimes upset a lot of people.

Lukasz: Can you accept PEP 649 right now?

Thomas: Pretty much already did.

Brett: Tell Larry not to make last-minute changes.

Thomas: PEP 649 is still in flux. We can still say we want to make some changes. But we won't because we can't change Larry's mind. Unlikely to make it into 3.12. It will definitely not be in 3.12 without the future import because I'm the RM and I say so. Without the future import, maybe in 3.13.

Guido: When will the other future be deprecated?

Thomas: We don't have a process for removing future import. Hard to decide how to do that. Need to wait for a while for things to settle down.

Lukasz: Can only remove the future when the oldest supported Python version supports 649. I really want it in 3.12 but I don't want a future import.

Thomas: No.

Shantanu: Skipping larger questions about future of typing you skip. What are these larger questions?

Thomas: We didn't skip those. Preferably typing changes need to be beneficial for non-typing use cases too. Obviously there's things that are good only for static typing, and we have discussions about whether we can do that. Every time the discussion comes up: Should the SC be the ones making this decision for the whole community. Or should the typing community have their own council? If we can't accept a clearly beneficial change like PEP 695, maybe we should separate the languages more. Could allow more improvements, but also come with costs.

Guido: Fascinating to hear how you think about this stuff. Sometimes it's very frustrating to hear so little about how you all think about this stuff when we haven't physically forced you to do it. Is there a form of communication from SC to typing that we could add that allows the SC to tell us how seriously they are taking everything.

Carl: They should write opinions like the Supreme Court.

Brett: It's hard. Easiest thing to do is record all meetings and broadcast them. Can't do that because of sensitive context. We do post notes but it's not enough. It doesn't happen more because it's hard to know what people want.

Pablo: Even if you know what everyone thinks, it's not enough. What happens is that the initial position of members often changes a lot. Typing PEPs especially get a lot of discussion. This is a great thing about the SC because we share ideas. But it makes it hard to share info. Ideas: One hour of something like this, more often. Early communication can help. Ask SC questions early? Still an oracle, but it can help.

Emily: There's a lot of process behind closed doors. We often change our minds. Monthly updates say "we discussed this PEP", but we don't get anything about the +/- . Improve the cadence of these updates? We're more aware of the reasons why we reject the PEP. We don't say much about accepting. We can improve the cadence and content of updates.

Brett: I like Emily's answer better.

Thomas: Monthly updates have always been problematic, because not all details matter for everyone, and it takes a lot of time to synthesize them. Easy to forget some details about a discussion. Not sure if monthly updates would be a good place to expand. We should write more elaborate acceptance and rejection notices. But that takes a lot of time. And goes through layers of review. The feedback can postpone acceptance by like 3 weeks. Maybe we could first put the acceptance, and then post the accept/reject later. Could even do it for old PEPs. Maybe we need office hours, like we already meet with Lukasz. Could be done in public. More interaction and less written in stone.

Emily: Double-edged sword. Consensus on an email draft can take a long time because we often change our minds. Takes a lot of time to synthesize.

Pablo: Elaborate responses can trigger more discussion. It's complicated. Limited resources. Burnout. Where do you get time for all this?

Emily: No great process for bringing things up to us early. Discussion should not come up as a surprise.

Thomas: Two more things. Often play devil's advocate. Suggest sending us emails about why a particular PEP was accepted/rejected.

Guido: Didn't mean to imply that we needed ahead-of-time information about what everyone is thinking. Do think more communication while you're making up your mind can be good. Also about communication after you have decided. Even for me, it's hard to write up an email to the SC. For normal users, even harder.

Thomas: <cut off>

Irit: Concrete suggestion. I had a PEP rejected, and the issue that caused the rejection never came up in the discussion. Suggestion: If you find something that never came up in discussion, bring it up to the author first.

Thomas: In this case, it was not the whole reason. There were other reasons. We liked other solutions better. If those don't work out, we can resurrect the PEP. In other cases, we have gone to the author with that. In the past, individual SC members also sometimes give their opinion. In Irit's case, we didn't think the PEP would come before the SC.

Guido: Third approach: Instead of accepting and rejecting a PEP, send it back for discussion. We have also done that in the past.

Thomas: We have done that in the past. SC no isn't end of discussion.

Brett: Sometimes we're also still thinking about it before it's submitted. Sometimes we think we'll accept it and then change our mind. A new thing might come up. If we do step in, people take it as "the SC has already made up its mind" and we want to avoid that.

Guido: Maybe the solution to the conundrum is SC members should participate in discussions more, so people know about their opinions early.

Thomas: I wish I had the time to contribute to more discussions. I wish we could have one-on-one discussions with PEP authors. But it's not workable with the resources we have. And the PSF can't just hire 5 people full time to do that.

Pablo: My policy is if I'm uncomfortable with something, I put my own view forward. And then people try to convince me. But I got emails where people said they wished I hadn't spoken up somewhere. If you think we're unfair, tell me.

Emily: A rejection of a PEP isn't necessarily a rejection of the whole idea.

Thomas: It's scary to email the SC. I don't want it to be scary to email the SC. We're not that smart. We're just making it up. If you have ideas on how to make it easier to contact the SC, contact the SC.

Amethyst: Getting back to type hints. Why not add type hints to the stdlib?

Thomas: it was a decision from Guido's era. I don't know why.

Pablo: It's hard. Difficult to do.

Brett: When this started, some core devs who just didn't like typing. Some core devs were like "if we do that, I'll quit".

Guido: If a brand-new module gets added, I'd be fine with it having types. But the existing stdlib is full of weird hacks and shortcuts for the sake of performance. It's hard to type check the body of the function. If type checkers aren't only for external consumption, it's very hard.

Brett: The other thing. We have to run a typechecker. Which one?

Lukasz: There are some annotations in the stdlib. Yes, there are lots of hard places. But also lots of easy places. With future import annotations, my original plan was to get things into the stdlib. Also, often type annotations require importing typing. That's much less the case now.

Guido: We need a mechanism to match up typeshed type hints to the stdlib.

Jelle: For type checking users, no need to for type annotating stdlib. It has to be for the benefit of the stdlib itself.

Pablo: It's been very helpful for the PEG generator.

## November 15, 2022

Recording (Google Drive link - TBD, [temporary Zoom recording](#))

[Chat transcript](#)

### Typing support for Deprecated and Typing Errors, Jelle Ziljstra

[Slides](#)

Notes:

- `@deprecated`: Jelle proposed a decorator that allows functions (and overloads?) to be considered deprecated. The type checker should report the error message mentioned in the decorator argument.
  - Example:

```
@deprecated("Use inspect.signature instead")
def getargspec() → None: ...
```
  - The overall feedback was positive.
- `@typing_error`: Jelle also proposed making certain combinations of parameter types to be deprecated
  - Example:


```
@typing_error("Using pow() with a mod of 0 will throw a
runtime error")
def pow(base: int, exp: int, mod: Literal[0]) → Never: ...
```
  - Feedback from the audience: The above overload doesn't help prevent `pow(x, y, 1 - 1)` since that won't match the above overload. Other examples are needed to motivate this feature.
- PyCon Typing Summit 2023: We decided to propose another summit at PyCon, given that the last one was well-received and the 2023 proposal deadline is coming up. Jelle and Pradeep agreed to co-organize the event. We will probably reach out to speakers early next year.

## September 13, 2022

Recording ([long-term Google Drive link](#))

[Chat transcript](#)

### Next Steps in `attrs` Static Analysis, Tin Tvrtković

Slides -  Next steps for attrs static analysis.pdf

Notes:

- Tin talked about the problem of querying and selecting fields from an ORM/ODM class in a type-safe manner.
- Querying/Projection: The `find` function needs to return a tuple of types based on the tuple of arguments passed to it. This is something that can be addressed by the `Map` operator (proposed as a followup to PEP 646 - Variadic generics).
- Selection: If the user specifies a filter such as `id == 1`, we would like to type check that the `id` field is of type `int` (see slide 21). Doing this could be hard for an arbitrary number of fields with arbitrary filtering operations. There was no good consensus on this during the meetup.
- Slightly related: Can `ClassVars` wrap a `TypeVar T`? Mypy disallows it, whereas Pyre allows it. Afaik this is not addressed in the specification, so we'd probably want to standardize on it.

August 9, 2022

Recording ([temporary Zoom recording](#), [long-term Google Drive link](#))  
[Chat transcript](#)

@override - Joshua Xu, Steven Troxler

[Slides](#)

- Joshua and Steven presented an `@override` decorator that can be used to indicate that a child method overrides a base class method. This is useful in cases where someone changes the base method name but not the child method name, which would not cause a type error.
- This bug is infrequent, but it can be costly and hard to detect in proprietary code bases where there are frequent refactors.
- Should we have a class decorator that forces subclasses to use `@override`? This doesn't seem particularly useful even for typeshed maintainers or library authors, so we probably don't want it in the PEP.
- David Hagen: Another weird and hard-to-debug case is when we add a new method to the base class but there exists a child method with the same name. Checking for overrides in strict mode would flag this as an unintentional override.
- Jelle: This is also useful as documentation (Explicit is Better than Implicit).

Exhaustiveness checking: Unions vs Enums, David Hagen, John Hagen

[Slides](#)

- David and John presented the motivating problem of three possible types of messages: `Quit`, `Write`, and `Move`. When using `match` or an if-elif ladder on a message, we want to check that the user has exhaustively checked all three possible variants.

- If we use a Union to represent the three variants, then we can enforce exhaustiveness-checking. However, if we want a common base class with a classmethod (such as `from_data`), we have to repeat it as a base class for each variant, and import it separately when we want to use `Message.from_data`. The base class and the union type are two different things to keep track of.
- Proposal: Decorate `Message` with `@enum` and nest the variants within the class. This allows classmethods and requires the user to track a single `Message` class.
- There were a *lot* of arguments for and against the proposal, and questions about the implementation. Overall, there was no clear consensus.

## June 28, 2022

Recording ([temporary Zoom recording](#), [long-term Google Drive link](#))  
[Chat transcript](#)

### TypeVar syntax PEP, Eric Traut

[Slides](#)

#### Meeting notes

Some notes from the meetup:

+ Eric mentioned that he's explicitly not including syntax for generic Callables in this PEP. That is, we are not looking to have syntax for `Callable[T][[T], T]`, etc.

+ Sebastian: I think when Jukka first suggested the `type` keyword, he had some more examples where such a keyword would be useful.

+ The PEP suggests `T: int` as a replacement for `TypeVar("T", bound=int)`. One concern about the upper-bound syntax is that it might be confusing for users who see `x: int` as a variable annotation elsewhere. That syntax usually means that any value of type compatible with `int` can be assigned to `x`, but that `x` will be seen as having type `int`. That is not the case for `T` with upper bound `int`, since it will be specialized to the specific type that was passed by the user. So, it might be worth having syntax such as `T <: int` or `T extends int`.

+ Regarding extensibility of the TypeVar "mini-language", Eric mentioned that, while it is harder to extend syntax, we might get away by using type annotation.

+ One idea suggested by Kevin was that we allow keyword arguments when specializing types. For example, if we have `GenericParent[T, R]`, we might have `class Base(GenericParent[R, T=int]): ...`. Note that PEP 637 (Support for indexing with keyword arguments) [2] was rejected

[3]. Though, the SC did say, "The strongest argument for the new syntax comes from the typing side of Python", so it might be worth trying again in the future (not in this PEP).

+ Jelle asked, "What if someone uses `from foo import *` and that imports `T`. (I suspect Eric already addressed this, from his mail above, but I'm noting it down for posterity.)

## June 01, 2022

Recording ([temporary Zoom recording](#), [long-term Google Drive link](#))  
[Chat transcript](#)

### Inline Syntax for TypeVars, Sebastian Rittau

[Slides](#)

#### Meeting notes

+ Context: In the Typing Summit panel [2] during PyCon 2022, we had asked Steering Council members (Thomas Wouters and Pablo Galindo Salgado) about future syntax proposals for typing. They replied that "The more constrained the syntax is, the less resistance there will be from the community." and "We need to consider that you can put type annotation syntax into a cast as an expression and it would work, so syntax changes are not just affecting types".

So, we decided to dedicate the meetup to a discussion about TypeVar syntax, since it is a constrained syntax that will (mostly) affect the syntax of statements rather than expressions.

+ Sebastian mentioned three possible syntaxes for declaring a TypeVar:

- (a) angular brackets `def foo<T>(x: T) -> T:`
- (b) square brackets `def foo[T](x: T) -> T:`
- (c) new scope syntax: such as, `let T = TypeVar("T") in def foo[T](x: T) -> T:` (Jelle had proposed this earlier)

Other proposals from the audience:

- (d) decorator syntax
- (e) new delimiter for TypeVar declarations: something different from `<>` and `[]`. Maybe `{}`.

For more details, see the slides linked in the doc.

+ Sebastian also described the "mini-language" to specify variance, upper bound, etc. The main approaches here were:

- (a) punctuation only (``def foo[T <= int](x: T) -> T``)
- (b) keyword (``def foo[T bound int](x: T) -> T``)
- (c) function-style syntax (``def foo[T (bound=int)](x: T) -> T``)

The most extensible option is (c). We could add new flags to the TypeVar declaration without having to change the language syntax.

+ Objections:

- (a) Angular brackets could lead to ambiguity with existing ``<`` and ``>``.
- (b) Square brackets could be hard to read given how much they are already used.
- (c) New scope syntax: This could hurt readability if generic classes and methods need to be indented. If we don't indent the statement after ``let T = TypeVar("T") in ``, it's unclear if T is scoped only to the next statement or to the rest of the file. Most importantly, it is pretty verbose in the current proposal - we have to write what we currently write (``T = TypeVar("T")``) plus ``let ... in``. It doesn't seem to be clearly better than the existing syntax.

+ Usability improvements:

- Guido suggested that we don't have to cover all use cases with the syntax. That might over-complicate the syntax and leave us without a feasible solution. Instead, we might want to have a concise syntax for the most common use cases, and defer to the older syntax for the remaining cases.
- Eric Traut suggested that "If a TypeVar is associated with only one scope, its variance could be inferred from its usage". That might eliminate the need for special syntax to specify variance.

+ Poll: I polled participants at the end of the meeting, to get a rough sense of where people stand. It looks like we were split on the high-level syntax and in favor of a function-style mini-language:

- high-level declaration:
  - + Angle brackets - 5/12 respondents (Steven, Jelle, Shannon, Erik, Konstantin)
  - + Introduce scope - 5/12 respondents (Carl, Batuhan, Sebastian, Jia, Alex)
  - + Square brackets - 2/12 respondents (James, Jukka)
- mini-language:
  - + Function-style syntax - 8/12 respondents (Carl, Steven, Batuhan T, Jelle Zijlstra, Sebastian Rittau, Jia Chen, Konstantin Schukraft, Jukka Lehtosalo)
  - + Punctuation - 3/12 respondents (Shannon, James, Alex)
  - + Keyword - 1/12 respondents (Erik)



If you'd like to continue this discussion, please reply to the **\*dedicated\*** TypeVar syntax thread: [3], so that this thread is purely for Typing Meetup logistics.

[1]: Meeting recording and notes:

<https://docs.google.com/document/d/17iqV7WWvB0lwA43EPIIqIUS6Xuvk08X3sEudAA-gQlo/edit?usp=sharing>

[2]: Typing Summit recording and notes:

[https://docs.google.com/document/d/1\\_CBM9SeBrFTFULtwAzGmxIBRhqzplArQ65cWa2WwfTI/edit#heading=h.hl5q80pc2xbz](https://docs.google.com/document/d/1_CBM9SeBrFTFULtwAzGmxIBRhqzplArQ65cWa2WwfTI/edit#heading=h.hl5q80pc2xbz)

[3]: TypeVar syntax thread:

<https://mail.python.org/archives/list/typing-sig@python.org/thread/GPQO2F4NHFVWZC2KEJBGDYIRFSR4AYHF/#2NQ4IEVWKUORDPWRB4QJ3YMOCLYOQP6W>

## PyCon Typing Summit, April 30, 2022

See notes [here](#).

### April 06, 2022

[Recording](#)

[Chat transcript](#)

### Inference Carve-outs: What to do when variables are unannotated, Shannon Zhu and Eric Traut

- Type checkers differ in their behavior for many cases of unannotated variables, function parameters, or attributes.
- Shannon and Eric pointed out some of these cases and argued that we should standardize on the behavior for some, if not all, of them.
- I polled the participants and found 16 in favor of standardization, none against, and 2 saying “I don’t know”.

[Slides](#)

### Extra fields in TypedDicts: Jonathan Scholbach and Steven Troxler

[Slides](#)

February 16, 2022

[Recording](#)

[Chat transcript](#)

Pyright Overview and Architecture, Eric Traut

[Slides](#)

January 13, 2022

[Recording](#)

[Chat transcript](#)

dataclasses-json, Charles Li

[Slides](#)

- Charles: I used `.from_dict` in my REST API example, but it should be `Person.schema.load` (n.b. `load` not `loads`)
- Charles: I didn't go into this tangent during the talk, but the library also features `.to_json()` and `.from_json()` (and `.to_dict` / `.from_dict`) that does not do validation. This was the original API before `.schema()`, and changing it to validate like `.schema()` will break backwards-compatibility (plan to add this to the README)
- Can we upstream this to the official dataclass library? Jelle: CPython will be hesitant to add this
- Type checking support: can you add an attribute to a dataclass?
  - Need to look into `dataclass-transform` (or `attribute-transform`) to make type checkers recognize `Person.from_dict`, etc.
  - [Example](#) type error
- Potential feature: Roll up `@dataclass` and `@dataclass-json` into a single decorator so that users don't have to use two decorators. But might be confusing or hard to use.

Phantom-types, Anton Agestam

[Slides](#)

- Phantom-types allows us to narrow types using arbitrary predicates.
- For example, Anton showed how to express `SequenceButNotStr` using his library
- See their [docs](#) for more details.

December 01, 2021

[Recording](#)

[Chat transcript](#)

pyanalyze: a semi-static type checker - Jelle Zijlstra

Slides:

<https://docs.google.com/presentation/d/10kLUbzo9R6AUVZldvHplAlgqRfJYbaknp55t6XKqg8M/edit?usp=sharing>

- Jelle discussed how pyanalyze can use runtime information. For example, it can handle dataclasses without any special handling for constructors.
- He also mentioned that plugins can also leverage runtime information.
- Question: How to use the `ExternalType` feature? [Docs](#)

Dataclass-transform - Eric Traut

Slides: **TODO**

- Eric walked through his design for dataclass-transform, a typing framework that provides precise attribute and method information for various dataclass-like libraries.
- Feedback
  - Pradeep (Pyre): We have ad hoc support for dataclass, attrs, and sqlalchemy. A common framework would remove the need to maintain such custom code.
  - Rebecca (Pytype): +1 to what Pradeep said. pytype has at least two overlays that do nothing but allow other libraries to behave like dataclasses.
- Question:
  - Nikita Sobolev: What about `\_\_hash\_\_` generation? The rules are different in dataclasses and attrs.
  - Nikita Sobolev: One more question: we also need to transform types quite often. For example, we transform `CharField()` to `str` in django's mypy plugin. How can we do that with this proposal?
  - Carl Meyer: One question I have is whether this proposal would allow just saying that Django's base Field class is a field descriptor, or would require explicitly listing every leaf Field type.

Typing Summit at PyCon

Pradeep brought up PyCon 2022 and solicited co-organizers for the Typing Summit.

November 03, 2021

[Recording](#)

[Chat transcript](#)

## Callable syntax - Steven Troxler

We will discuss the remaining question about the Callable syntax proposal.

There was some heated debate about this (from 17:00 to 33:00 in the recording).

- TL;dr: Shorthand syntax remains unanimously supported. Extended syntax is controversial - some want it now, others don't want it at all, others want the proposal of having a function name as a type.
- People from Pyre and Pyright propose having just the shorthand proposal in the PEP.
- Guido: The shorthand proposal seems weaker than the extended shorthand proposal if the rationale is that it is more work to implement the latter. Also seems weird to propose the complex syntax in a later PEP. I recommend adding the extended syntax and offering to slim it back to the shorthand syntax if the Steering Council wants.
- Shannon: Prefer the shorthand syntax, not because it is less work to implement, but because the extended shorthand syntax leads to complex edge cases.
- Pradeep: The stats represent untyped code as well, which is unconstrained by the limits of the Callable type. Even in that kind of untyped code, callbacks using keyword arguments, etc. were rare.
- A straw poll came out to ~6 in favor of either shorthand or shorthand + Lukasz proposal and 5 in favor of extended shorthand. (Exact counts are hard because there were partial votes, but the gist is that it was pretty much a tie.)

**Takeaway:** We've been discussing this topic for months now and there are *\*no\** prospects of a firm consensus on extended syntax. Given that the 3.11 deadline is early May and we will probably need to iterate on the PEP based on SC feedback, the current plan is for us to go ahead and draft the PEP with shorthand syntax. Hopefully, we can implement it, submit it to the Steering Council, and get it approved (can take ~2 months) before the deadline for 3.11.

Slides:

<https://drive.google.com/file/d/1NGnJi3m6KVyTJgRAofuUFyMZeYfozwhu/view?usp=sharing>

## Precise Types for **\*\*kwargs** - Franek Magiera

This is a highly-requested feature on both typing-sig and GitHub

People agreed that this is a valuable feature. The main questions were about the special form: `Expand[Movie]` vs `**Movie` vs `Unpack[Movie]`.

Slides:

<https://drive.google.com/file/d/1LPnUhuE1h1nRihxm15iBOdguhr5KNpMv/view?usp=sharing>

## Better Discussion forum than typing-sig

- Better code markup, likes, and easy-to-follow threads
- Try GitHub Discussions for a month and see if there are major drawbacks compared to typing-sig.
- Downside of the mailing list
  - Spam moderation
  - New contributors have to wait before their mail is approved.
- Downside of GitHub issue tracker
  - Guido mentioned that historically people used to post in the wrong issue tracker.
  - Issue tracker has an old pile of dead issues.
- Sebastian Rittau volunteered to clean up the issue tracker so that it's up-to-date and we can try out typing discussions on [github.com/python/typing/discussions](https://github.com/python/typing/discussions)
  - If this is not doable or it's too inconvenient, we could have a separate repo.
- Getting overwhelmed by GitHub notifications
  - We could set up a standard filter for typing-related discussions vs help questions (Shannon Zhu?)

## 20 September 2021 - Callable Syntax Discussion

Slides:

Pradeep - Callable syntax stats -

[https://drive.google.com/file/d/1k\\_TqrNKcbWihRZdhMGf6K\\_GcLmn9ny3m/view?usp=s...](https://drive.google.com/file/d/1k_TqrNKcbWihRZdhMGf6K_GcLmn9ny3m/view?usp=s...)

Steven Troxler - Callable examples and questions -

[https://drive.google.com/file/d/119gt9-nCa\\_mGUjAVanWT2mscC3KIhfN4/view?usp=s...](https://drive.google.com/file/d/119gt9-nCa_mGUjAVanWT2mscC3KIhfN4/view?usp=s...)

Eric Traut -

<https://www.dropbox.com/s/sshgtr4p30cs0vc/Python%20Callable%20Syntax%20Proposals.pdf?dl=0>

Doc containing proposal details:

<https://docs.google.com/document/d/11VkNk9N8whxS1SrBv2BNN1Csl5yeTlgG7WyLrCB5...>

### 2. Meeting notes:

- Nobody was in favor of the def-style syntax by itself, since it was verbose for the most common cases. The need for forward-slash to indicate positional-only parameters was another confusing pitfall.

- Jake Bailey pointed out that, if lambdas get special syntax, Callable types would need to be distinguishable from them. Steven suggested that having different arrow styles could help: something like `(int) -> str` for callable types and `(x, y) => x + y` for lambdas.
- Jelle Zijlstra asked if the rejection of PEP 637 (Support for indexing with keyword arguments) by the Steering Council could bode poorly for callable syntax. Guido mentioned that the rejection there was more about keyword arguments in indexing. Typing syntax was only a peripheral concern in that PEP.

4. Rationale for hybrid over shorthand: I'm interested in hearing about the use cases that people care about here. If you favor hybrid over shorthand, could you share your reasons why? We can try to find a solution that addresses those use cases while being well-specified.

## 13 May 2021 - Typing Summit, PyCon 2021

Recording: <https://www.youtube.com/watch?v=ld9rwCvGdhc&t=9786s>

Talks:

- Type Syntax Simplifications (Maggie Moss) [0:06](#)
- Validating JSON with TypedDict, trycast, and TypeForm (David Foster) [29:45](#)
- Type Variables for All (Pradeep Kumar Srinivasan) [54:20](#)
- Static Python: Types in Bytecode Compilation & Runtime (Carl Meyer) [1:25:16](#)
- Incremental Check in Pyre (Jia Chen) [1:50:55](#)
- Scaling Typed to 1000 Packages (Jukka Lehtosalo) [2:17:01](#)
- Catching Tensor Shape Errors Using the Type Checker (Pradeep Kumar Srinivasan, Matthew Rahtz) [2:43:06](#)
- Type Arithmetic (Alfonso Castaño) [3:11:10](#)

## Topic for Later Meetups

- Literal string types
- Non-required Protocol fields (Sebastian Rittau?)
- Type support for sealed classes (Adrian Freund?)
- Safe, typed JSON validation