New issue

# A Comparable type? #59

Closed

gvanrossum opened on Mar 19, 2015

I (Guido) wondered how I would compare two arguments whose type is a type variable:

```
def cmp(a: T, b: T) -> int:
    if a < b: return -1
    if a > b: return 1
    return 0
```

Jukka wrote:
"""
This is a long-standing issue, and we've discussed this before in some detail (e.g., https://github.com/JukkaL/typing/issues/9). However, it wasn't discussed in typehinting, as far as I can tell. Currently we need to use a cast to use < instead of ==, which is pretty ugly.

We could define a Comparable ABC and support that as an "upper bound" for a type variable (bounded polymorphism). About the simplest useful implementation would be like this:

```
class Comparable(metaclass=ABCMeta):
    @abstractmethod
    def __gt__(self, other: Any) -> bool: pass
    ... # __lt__ etc. as well

...

CT = TypeVar('CT', bound=Comparable) # This is different from TypeVar('CT',
Comparable)!

def min(x: CT, y: CT) -> CT:
    if x < y:
        return x
    else:
        return y
```

```
f(1, 2) # ok, return type int
f('x', 'y') # ok, return type str
f('x', 1) # probably ok, return type Comparable, which is not optimal
f(int, int)  # error, types aren't comparable
```

However, this doesn't verify whether the types can be compared to each other, only that they can be compared to something (because of the Any argument type). This feature would be easy to add to mypy.

The way Java etc. do this is to support a fancier language feature called "F-bounded polymorphism" or "F-bounded quantification". With it we can say that int can be compared with only int (Comparable[int]), etc. For example:

```
class Comparable(Generic[T]):
    @abstractmethod
    def __gt__(self, other: T) -> bool: pass
    ... # __lt__ etc. as well

...


CT = TypeVar('CT', bound=Comparable['CT'])   # Any type that is comparable with
itself

def min(x: CT, y: CT) -> CT:
    if x < y:
        return x
    else:
        return y

f(1, 2) # ok, return type int
f('x', 'y') # ok, return type str
f('x', 1) # error, since these are not comparable to each other
f(int, int)  # error, types aren't comparable at all
```

This would be more involved to add to mypy. It would probably be one or two days of work for a minimal implementation. I'm not even quite sure that the above would be sufficiently general (Comparable should be contravariant, I think, so that Comparable[object] is a subtype of Comparable[int] :-/ ).

Comparable is probably the only common use case for this *complex* feature.
"""

gvanrossum on Mar 24, 2015 Member Author •••

> I think that for now the first solution is fine -- Python's rules for what can be compared to what are complex (every type gets to decide for itself) so even the F-bounded nonsense won't capture reality. It looks like we

could still add a more complex version in the future since it would involve a new keyword arg to `TypeVar()`.

**vlasovskikh** changed the title ~~A Comparable metaclass?~~ A Comparable type? on Apr 13, 2015

vlasovskikh on Apr 14, 2015 — Member

Jukka, Mark, and myself believe that it's a good idea to have the generic type `Comparable` and not to introduce any bounded polymorphism for rare cases such as `min(x, y)`:

```
def min(x: Comparable, y: Comparable) -> Any:
    ...
```

JukkaL on Apr 14, 2015 — Contributor

I was actually arguing for supporting bounds, but not using 'F-bounded polymorphism'. I.e., `min` could be written like this (we just need to figure out what to call `bound`):

```
from typing import TypeVar, Comparable

T = TypeVar('T', bound=Comparable)

def min(x: T, y: T) -> T:
    ...
```

👍 1

JukkaL on Apr 14, 2015 — Contributor

And maybe define `Comparable` conceptually like this:

```
class Comparable(metaclass=ABCMeta):
    @abstractmethod
    def __lt__(self, other: Any) -> bool: pass
    @abstractmethod
    def __gt__(self, other: Any) -> bool: pass
    def __le__(self, other: Any) -> bool:
        return not self > other
    def __ge__(self, other: Any) -> bool:
        return not self < other
```

Equality would be inherited from `object`, so it doesn't need to specified explicitly.

JukkaL on May 4, 2015 Contributor •••

Are we going to include `Comparable` and type variable bounds?

gvanrossum on May 4, 2015 Member Author •••

I would like it yes.

gvanrossum on May 5, 2015 Member Author •••

**@JukkaL** please tell me what syntax this would have (I'm blanking out on what we decided).

JukkaL on May 7, 2015 Contributor •••

As far as I can remember, the only presented alternative has been this, and I'm okay with it:

```
T = TypeVar('T', bound=Comparable)
```

To make it more explicit (but also more verbose and a bit technical sounding), we could use `upper_bound`:

```
T = TypeVar('T', upper_bound=Comparable)
```

We could also have `lower_bound` for generality. It would probably be at least marginally useful. Example:

```
T = TypeVar('T', lower_bound=int)

def append_first(x: List[int], y: List[T]) -> None:
    y.append(x[0])

a = [1]
b = [...]  # type: List[Union[int, str]]

append_first(a, b)  # Okay, but only because of lower_bound
```

FWIW, Java uses this terminology (upper/lower bound): http://docs.oracle.com/javase/7/docs/api/javax/lang/model/type/TypeVariable.html

The example below would be invalid -- you can only have a bound or constraints (not sure if we should come up with a new term, since a bound is also a constraint?), but not both:

```
T = TypeVar('T', int, str, bound=Comparable)    # Error
```

 gvanrossum on May 7, 2015 Member Author •••

Hm, I would go with `bound=` and forget about the lower bound. Or if you really want the latter we can make `bound=` a shortcut for `upper_bound=`.

On Wed, May 6, 2015 at 9:11 PM, Jukka Lehtosalo notifications@github.com wrote:

> As far as I can remember, the only presented alternative has been this, and I'm okay with it:
>
> ```
> T = TypeVar('T', bound=Comparable)
> ```
>
> To make it more explicit (but also more verbose and a bit technical sounding), we could use upper_bound:
>
> ```
> T = TypeVar('T', upper_bound=Comparable)
> ```
>
> We could also have lower_bound for generality. It would probably be at least marginally useful. Example:
>
> ```
> T = TypeVar('T', lower_bound=int)
>
> def append_first(x: List[int], y: List[T]) -> None:
> y.append(x[0])
>
> a = [1]
> b = [...] # type: List[Union[int, str]]
> ```

append_first(a, b) # Okay, but only because of lower_bound

FWIW, Java uses this terminology (upper/lower bound):
http://docs.oracle.com/javase/7/docs/api/javax/lang/model/type/TypeVariable.html

The example below would be invalid -- you can only have a bound or constraints (not sure if we should come up with a new term, since a bound
is also a constraint?), but not both:

T = TypeVar('T', int, str, bound=Comparable) # Error

—
Reply to this email directly or view it on GitHub
#59 (comment).

--Guido van Rossum (python.org/~guido)

JukkaL on May 7, 2015 Contributor •••

Okay, let's go with just `bound=`. We can revisit this discussion later if we see compelling use cases for lower bounds. My above example was very contrived.

**gvanrossum** added **bug** on May 18, 2015

gvanrossum on May 18, 2015 Member Author •••

This needs text to be added to the PEP and an implementation for typing.py.

**gvanrossum** added a commit that references this issue on May 19, 2015

Add type variables with upper bound. (Fixes issue #59.) bc6276d

**gvanrossum** mentioned this on May 19, 2015

Implement TypeVar(..., bound=<boundary_type>) python/mypy#689

**1 remaining item**

Load more

**gvanrossum** closed this as completed on May 19, 2015

**smallnamespace** mentioned this on May 20, 2016

Will recursive or mutually recursive bounds on TypeVar be supported? python/mypy#1561

mitar on Dec 16, 2017 Contributor

Hm, but `Comparable` itself has never been defined, no?

👍 10

Dentosal on Dec 25, 2017

The implementation of `Comparable` shouldn't be too complicated. I've been using this version:

```
import typing
from typing import Any
from typing_extensions import Protocol
from abc import abstractmethod

C = typing.TypeVar("C", bound="Comparable")

class Comparable(Protocol):
    @abstractmethod
    def __eq__(self, other: Any) -> bool:
        pass

    @abstractmethod
    def __lt__(self: C, other: C) -> bool:
        pass

    def __gt__(self: C, other: C) -> bool:
        return (not self < other) and self != other

    def __le__(self: C, other: C) -> bool:
        return self < other or self == other

    def __ge__(self: C, other: C) -> bool:
        return (not self < other)
```

👍 18

ilevkivskyi on Dec 28, 2017 · edited by ilevkivskyi  Edits  Member  •••

Hm, since ABCs in `typing` now support structural subtyping and there appeared many things that are "automatically" comparable (attr classes and dataclasses) maybe we can add `Comparable` protocol to `typing`?

**@Dentosal**
It looks like there is a problem with your `Comparable` class that it allows `42 < "abc"`.

👍 23

**TV4Fun** mentioned this on Oct 3, 2018
Set lower bound on TypeVars #585

**carver** mentioned this on Apr 16, 2019
Add numeric clamp utility ethereum/eth-utils#150

**poscat0x04** mentioned this on Aug 28, 2019
Lower bound for TypeVars #674

**russelldavis** mentioned this on May 26, 2020
mypy gives error on PEP 484 bound= example python/mypy#8889

MartinThoma on Jun 25, 2020  •••

**@Dentosal** Thank you for sharing your implementation! 2.5 years later, would you change something (for Python 3.8+)? For example, the `@abstractmethod` might lead to people subclassing the Protocol instead of using it as a type annotation.

👍 3 👀 1

**trishankatdatadog** added 2 commits that reference this issue on Nov 30, 2020

A decent implementation of Comparable. ••• Verified f8d3437

Update timestamp (#12) ••• Verified 83a94a9

**lossyrob** mentioned this on May 12, 2021
Added support for summaries stac-utils/pystac#264

**lhchavez** mentioned this on Aug 19, 2021

Miguel 3.2 - Stack Min [Python] techqueria/data-structures-and-algorithms#83

**mlenzen** mentioned this on Jul 16, 2022

Type annotation for RangeMap keys being comparable mlenzen/collections-extended#191

**marcelm** added a commit that references this issue on Aug 20, 2022

Add "Comparable" Protocol ... fa6e1be

**Karrenbelt** mentioned this on Nov 14, 2022

[1.27.0] Tests for custom data types valory-xyz/open-aea#428

**kadarakos** mentioned this on Nov 30, 2022

Give schedules access to the key, step, and last eval score explosion/thinc#804

**bdraco** mentioned this on Jun 24, 2024

Bump voluptuous to 0.15.0 home-assistant/core#120268

**toofishes** mentioned this on Oct 21, 2024

Fix typing issues around Range/RangeValue MagicStack/asyncpg#1196



## Metadata

**Assignees**

No one assigned

**Labels**

No labels

**Projects**

No projects

**Milestone**

No milestone

**Relationships**

None yet

**Development**

No branches or pull requests

**Participants**

 +2