Too small for a PEP: minor new typing features in Python 3.11

Jelle Zijlstra

Quora

What's new in Python 3.11?

- PEP 646 (TypeVarTuple + Unpack)
- PEP 655 (Required + NotRequired)
- PEP 673 (Self)
- PEP 675 (LiteralString)
- PEP 681 (@dataclass_transform) (hopefully!)
- … but I'm not going to talk about those

What's new in Python 3.11?

- reveal_type()
- Never
- assert_never()
- assert_type()
- get_overloads()
- .__final__ = True
- Any as a base class

typing.reveal_type()

```
1    from typing import reveal_type
2    reveal_type(1)  # Revealed type is 'Literal[1]'
```

- reveal_type() was widely implemented but unstandardized and hard to discover
- Some use cases for having it available at runtime:
  - Running test suite while debugging types
  - Runnable examples in education

typing.Never

```
1    from typing import Never
2    def f() -> Never:
3        assert False
```

- Equivalent to NoReturn
- Represents the bottom type
- Recommendation: Treat as alias to NoReturn, output “Never” in 3.11+

typing.assert_never()

```
1    from typing import assert_never
2    def get_value(x: bool) -> int:
3        match x:
4            case True: return 1
5            case False: return 0
6            case _: assert_never(x)
```

- Assert that code is unreachable
- Commonly implemented helper function
- No special treatment in type checkers

typing.assert_type()

```python
from typing import assert_type
def f(x: int) -> None:
    assert_type(x, int)  # ok
    assert_type(x, bool)  # E: Type is int (expected bool)
```

- Check that type inference works as expected
- Useful for testing stubs and libraries

typing.get_overloads()

```
1   from typing import get_overloads, overload
2   @overload
3   def f(x: int) -> str: ...
4   def f(x: int) -> object: ...
5   print(get_overloads(f))  # [<function f at ...>]
```

- Enable runtime introspection of overloads
- Use cases: runtime type checkers, help()
- Downside: Memory usage, overhead when defining overloads

@typing.final sets .__final__ = True

```
1    from typing import final
2    @final
3    class X: pass
4    assert X.__final__ is True
```

- Another runtime introspection helper
- All typing decorators are now introspectable at runtime

Any as a base class

```
1    from typing import Any
2    class Mock(Any): ...
3    assert issubclass(Mock, Any)
```

- Type checkers already allowed this
- Previously threw an error at runtime
- Useful for mock objects

Takeaways

- Still room for improvements
- We can get new useful things into typing.py
- Let’s make 3.12 even better

Bonus: Generic NamedTuples?

```
4   class NT(NamedTuple, Generic[T]):
5       foo: int
6       bar: T
```

- PR opened today by Serhyi
- I'd like to merge it, unless the typing community has concerns
- Bonus to the bonus: What about generalized multiple inheritance support?