TYPE ANNOTATIONS IN PYTHON

Kai Willadsen

DISCLAIMERS

DISCLAIMERS

- I will completely ignore Python 2

DISCLAIMERS

- I will completely ignore Python 2
- I am not a type theory expert

DISCLAIMERS

- I will completely ignore Python 2
- I am not a type theory expert
- I am ambivalent about typing

WHAT ARE TYPE ANNOTATIONS?

- Everything in Python is already typed!

WHAT ARE TYPE ANNOTATIONS?

- Everything in Python is already typed!
- ...it's just not statically checked.

WHAT ARE TYPE ANNOTATIONS?

- Everything in Python is already typed!
- ...it's just not statically checked.
- Type annotations give tools enough information to check that types make sense

WHAT ARE TYPE ANNOTATIONS?

- Everything in Python is already typed!
- ...it's just not statically checked.
- Type annotations give tools enough information to check that types make sense
- Python typing is gradual typing

SIMPLE TYPE CHECKING

```python
things = None
for thing in things:
    ...
```

SIMPLE TYPE CHECKING

```python
things = None
for thing in things:
    ...
```

```
$ mypy why.py
why.py:3: error: "None" has no attribute "__iter__" (not
iterable)
```

SIMPLE TYPE CHECKING

```python
from random import random

things = None
if random() > 0.5:
    things = range(5)

for thing in things:
    ...
```

SIMPLE TYPE CHECKING

```python
from random import random

things = None
if random() > 0.5:
    things = range(5)

for thing in things:
    ...
```

```
$ mypy why.py
why.py:6: error: Item "None" of "Optional[range]" has no
attribute "__iter__" (not iterable)
```

SIMPLE TYPE CHECKING

```python
def do_things(things):
    for thing in things:
        ...

things = None
do_things(things)
```

SIMPLE TYPE CHECKING

```python
def do_things(things):
    for thing in things:
        ...

things = None
do_things(things)
```

```
$ mypy why.py
$
```

SIMPLE TYPE CHECKING

```python
def do_things(things):
    for thing in things:
        ...

things = None
do_things(things)
```

```
$ mypy why.py
$
```

```
$ mypy --strict why.py
why.py:1: error: Function is missing a type annotation
why.py:6: error: Call to untyped function "do_things" in
typed context
```

TYPE ANNOTATION HISTORY

- 3.0: Function annotation syntax added
- 3.5: Type annotation syntax, `typing` module added
- 3.6: Variable annotation syntax
- 3.7: Data classes, delayed annotations, faster `typing` module
- 3.8: Protocol typing, final qualifier, `typing` is now official

PYTHON 3

FUNCTION ANNOTATIONS (PEP 3107)

- Unused when introduced
- Stored in `__annotations__` on the function, class or module
- Even now, using these for type annotations is mostly optional

```
>>> def foo(a: 'hey look numbers', b: 1 + 1) -> range(3):
...     ...
...
>>> foo.__annotations__
{'a': 'hey look numbers', 'b': 2, 'return': range(0, 3)}
```

PYTHON 3.5

TYPE ANNOTATION SYNTAX (PEP 484)

Function annotations define argument and return types

```
>>> def foo(a: str, b: int) -> str:
...     return a + str(b)
...
>>> foo.__annotations__
{'a': <class 'str'>, 'b': <class 'int'>, 'return': <class
'str'>}
```

ANNOTATED TYPE CHECKING

```python
def foo(a: str, b: int) -> str:
    return a + str(b)

bar = 1 + foo('hey', 'nope')
```

```
$ mypy foo.py
foo.py:4: error: Unsupported operand types for + ("int" and
"str")
foo.py:4: error: Argument 2 to "foo" has incompatible type
"str"; expected "int"
```

`typing` MODULE

- Module is technically "provisional" until 3.8
- `typing` provides additional syntax for:
  - typing-specific notions e.g., `Union`, `Any`, `Callable`
  - generic types e.g., `List[float]`, `Dict[str, int]`
  - distinct types e.g., `ObjectID = NewType('ObjectID', int)`

typing MODULE

```python
def dumb_sum(
    numbers: List[Union[int, float]],
) -> Optional[int]:

    # Check out this formatting disaster ^^

    if numbers:
        return int(sum(numbers))
```

PYTHON 3.6

VARIABLE ANNOTATIONS (PEP 526)

Variable annotations define or clarify variables in scope

```python
class Colour:
    name: str = 'unnamed'
    red: int
    green: int
    blue: int

palette: Dict[str, Colour] = {}
text_colour: Colour  # No initial value
```

PYTHON 3.7

DATA CLASSES (PEP 557)

```
from dataclasses import dataclass

@dataclass
class Colour:
    red: int
    green: int
    blue: int
    name: str = 'unnamed'
```

```
>>> c1 = Colour(red=127, green=127, blue=255)
>>> c1
Colour(red=127, green=127, blue=255, name='unnamed')
```

FORWARD ANNOTATIONS (PEP 563)

Actually just delayed annotation evaluation

```python
def do_a_thing(thing: A):
    ...

class A:
    ...
```

FORWARD ANNOTATIONS (PEP 563)

Actually just delayed annotation evaluation

```
def do_a_thing(thing: A):
    ...

class A:
    ...
```

Until Python 4, requires

```
from __future__ import annotations
```

FORWARD ANNOTATIONS (PEP 563)

```
>>> import typing
>>> class Now:
...     a: typing.List[int]
...
>>> Now.__annotations__
{'a': typing.List[int]}
>>> from __future__ import annotations
>>> class Later:
...     b: typing.List[int]
...
>>> Later.__annotations__
{'b': 'typing.List[int]'}
```

SUMMARY

- Python's typing has evolved a lot over recent releases

SUMMARY

- Python's typing has evolved a lot over recent releases
- Type annotations can help your CI, your editor and you

SUMMARY

- Python's typing has evolved a lot over recent releases
- Type annotations can help your CI, your editor and you
- mypy is your best starting point for type checking

MORE INFORMATION

- `typing` module https://docs.python.org/3/library/typing.html
- `mypy` cheat sheet https://mypy.readthedocs.io/en/latest/cheat_sheet_py3.html