facebook

Why you should care about types

How Python typing helped my team scale

Luka Šterbić

Software Engineer @ Facebook London

luka@fb.com

github.com/sterbic

Agenda

1. Why is typing important
2. Typing 101
3. Types in the real world
4. Tips & tricks

Why should I even care about typing?

What jobs?

```
def submit(jobs):
    for job in jobs:
        try:
            submit_job(job, priority=job.meta.priority)
            log(job.name, success=True)
        except queue.Full:
            log(job.name, success=False)
```

What jobs?

```
def submit(jobs):
    for job in jobs:
        try:
            submit_job(job, priority=job.meta.priority)
            log(job.name, success=True)
        except queue.Full:
            log(job.name, success=False)
```

What jobs?

```
def submit(jobs):
    for job in jobs:
        try:
            submit_job(job, priority=job.meta.priority)
            log(job.name, success=True)
        except queue.Full:
            log(job.name, success=False)
```

What jobs?

```
def submit(jobs):
    for job in jobs:
        try:
            submit_job(job, priority=job.meta.priority)
            log(job.name, success=True)
        except queue.Full:
            log(job.name, success=False)
```

- Grep for it?

```
[luka:~/fbcode:0 (e7090bd|remote/master)]$ grep 'class Job' | wc -l
1376
```

What jobs?

```
def submit(jobs):
    for job in jobs:
        try:
            submit_job(job, priority=job.meta.priority)
            log(job.name, success=True)
        except queue.Full:
            log(job.name, success=False)
```

```
import scheduler
scheduler.submit()
```

```
from scheduler import submit
submit()
```

What jobs?

```
def submit(jobs):
    for job in jobs:
        try:
            submit_job(job, priority=job.meta.priority)
            log(job.name, success=True)
        except queue.Full:
            log(job.name, success=False)
```

```
from scheduler import submit
submit(create_jobs_10_modules_away())
```

What jobs?

```
def submit(jobs):
    for job in jobs:
        print(type(job).__name__)
        print([
            name for name in dir(job)
            if not name.startswith("__")
        ])
        try:
            submit_job(job, priority=job.meta.priority)
            log(job.name, success=True)
        except queue.Full:
            log(job.name, success=False)
```

What jobs?

```
from typing import Iterable
from fb.scheduler.jobs import Job

def submit(jobs: Iterable[Job]) -> None:
    for job in jobs:
        try:
            submit_job(job, priority=job.meta.priority)
            log(job.name, success=True)
        except queue.Full:
            log(job.name, success=False)
```

Are Python types Pythonic?

Are Python types Pythonic?

Yes!

```
> import this
...
Explicit is better than implicit.
...
Readability counts.
...
In the face of ambiguity, refuse the temptation to guess.
```

- If nothing else, because Guido said so

Typing 101

What needs to be annotated?

```python
def my_print(message):
    print(f"< {message} >")


class Square:
    def __init__(self, a):
        self.__a = a
```

```python
def my_print(message: str) -> None:
    print(f"< {message} >")


class Square:
    def __init__(self, a: int) -> None:
        self.__a = a
```

What needs to be annotated?

```
def my_print(message):
    print(f"< {message} >")


class Square:
    def __init__(self, a):
        self.__a = a
```

```
def my_print(message: str) -> None:
    print(f"< {message} >")


class Square:
    def __init__(self, a: int) -> None:
        self.__a = a
```

What needs to be annotated?

- In some cases, variables

```
dictionary = {}
# complex code
# nested if-else blocks
# function calls
func(dictionary)  # expects str keys
```

What needs to be annotated?

- In some cases, variables

```
dictionary: Dict[str, int] = {}
# complex code
# nested if-else blocks
# function calls
func(dictionary)  # expects str keys
```

- For Python < 3.6

```
dictionary = {}  # type: Dict[str, int]
```

Collections

```python
from typing import List, Set, Dict, Tuple

def filter_countries(
    countries: Dict[int, str],              # country code to name
    locations: List[Tuple[float, float]],  # lat / long of users
    blacklist: Set[str],                    # shouldn't be available
) -> List[str]:
    pass
```

Collections

```python
from typing import List, Set, Dict, Tuple

def filter_countries(
    countries: Dict[int, str],              # country code to name
    locations: List[Tuple[float, float]],  # lat / long of users
    blacklist: Set[str],                    # shouldn't be available
) -> List[str]:
    pass
```

Collections

```python
from typing import List, Set, Dict, Tuple

def filter_countries(
    countries: Dict[int, str],              # country code to name
    locations: List[Tuple[float, float]],  # lat / long of users
    blacklist: Set[str],                    # shouldn't be available
) -> List[str]:
    pass
```

Collections

```python
from typing import List, Set, Dict, Tuple

def filter_countries(
    countries: Dict[int, str],              # country code to name
    locations: List[Tuple[float, float]],  # lat / long of users
    blacklist: Set[str],                    # shouldn't be available
) -> List[str]:
    pass
```

Unions

```python
from typing import Union

def get_by_id(id: int) -> Union[User, Page]:
    pass


luka: User = get_by_id(1451268501)
pycon_au: Page = get_by_id(221942894648201)
probably_none = get_by_id(-1)
```

Unions

```
from typing import Union

def get_by_id(id: int) -> Union[User, Page, None]:
    pass


luka: User = get_by_id(1451268501)
pycon_au: Page = get_by_id(221942894648201)
probably_none = get_by_id(-1)
```

Unions

```
from typing import Union, Optional

def get_by_id(id: int) -> Optional[Union[User, Page]]:
    pass
```

```
luka: User = get_by_id(1451268501)
pycon_au: Page = get_by_id(221942894648201)
probably_none = get_by_id(-1)
```

```python
from typing import Type

class BaseClass:
    pass

class DerivedClass(BaseClass):
    pass

def factory(clazz: Type[BaseClass]) -> BaseClass:
    return clazz()


base = factory(BaseClass)  # BaseClass
derived = factory(DerivedClass)  # BaseClass
```

TypeVars

```
from typing import Type, TypeVar

T = TypeVar("T", bound=BaseClass)

def factory(clazz: Type[T]) -> T:
    return clazz()

base = factory(BaseClass)  # BaseClass
derived = factory(DerivedClass)  # DerivedClass
```

The Type Checker

```python
# test.py
import math
ten = math.sqrt(100)
ten = math.sqrt("100")
```

```
$ mypy test.py
```

```
test.py:4: error:
Argument 1 to "sqrt" has incompatible type "str"; expected "float"
```

Advanced Topics

- Forward references
- TYPE_CHECKING
- @overload
- Pyre --> https://github.com/facebook/pyre-check

Typing in the real world

The Easy Bit

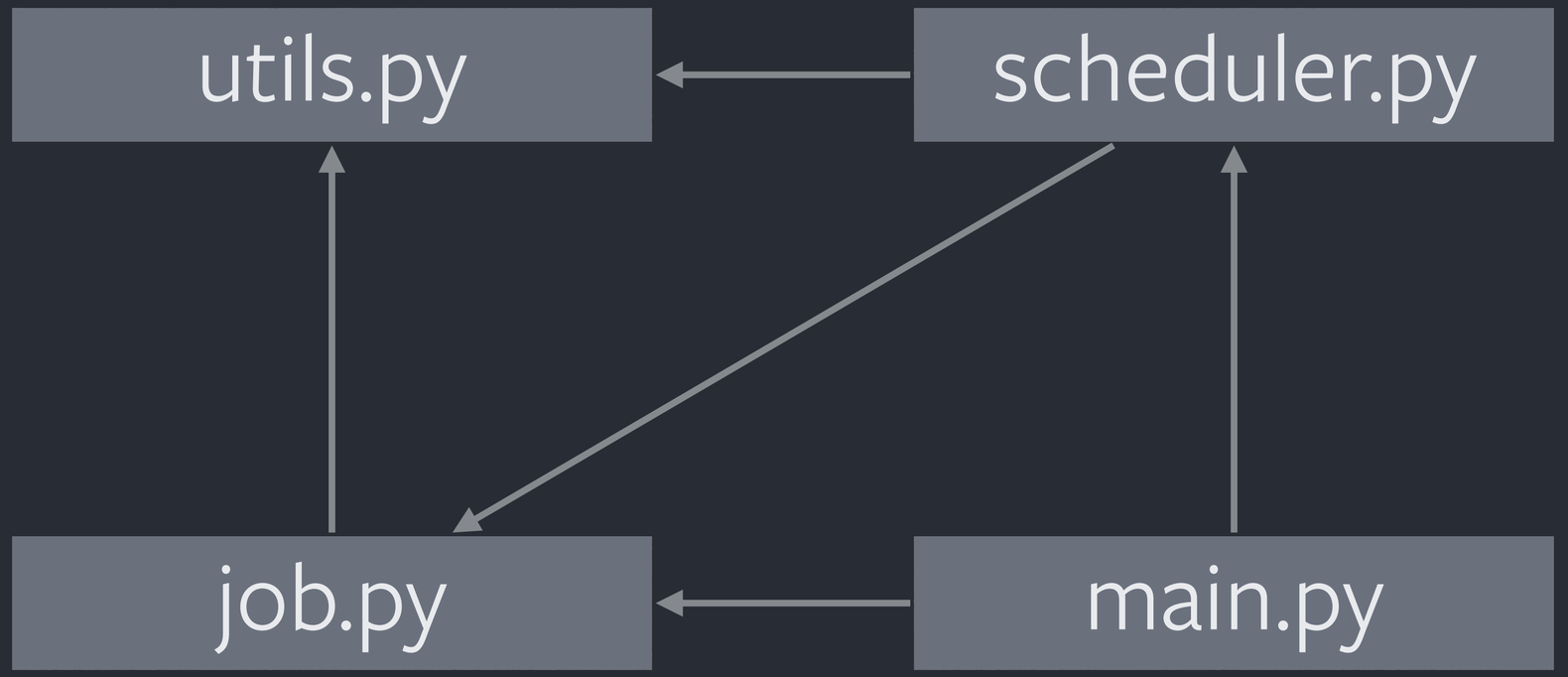
New projects

1. 100% typed coverage from Day 1
2. Type checker in CI
3. Profit!

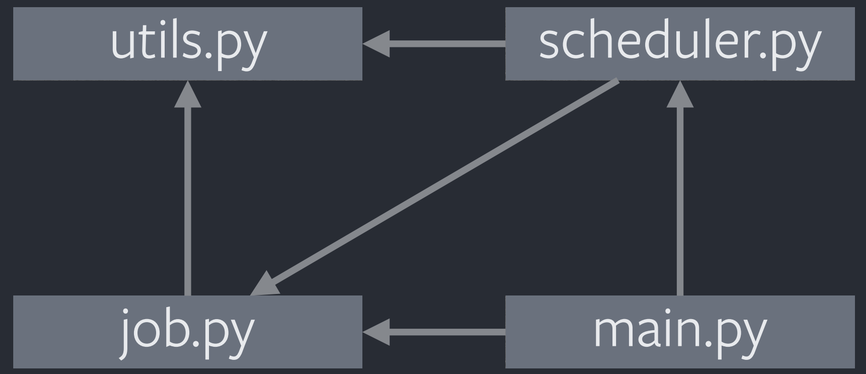
The Hard Part

Existing (Large) Projects

1. ... first understand gradual typing

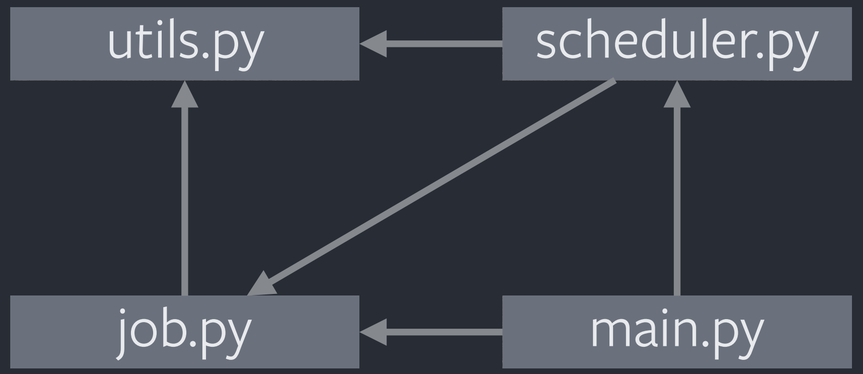

utils.py
scheduler.py
job.py
main.py

```python
# utils.py
import json
def pretty_print(job: Job) -> None:
    print(json.dumps(job, indent=4))


# job.py
from utils import pretty_print
def create_job():
    job = Job()
    pretty_print("job created!")
    return job
```



```python
# utils.py
import json
def pretty_print(job: Job) -> None:
    print(json.dumps(job, indent=4))
```

```python
# job.py
from utils import pretty_print
def create_job() -> Job:
    job = Job()
    pretty_print("job created!")
    return job
```



error: Argument 1 to "pretty_print" has incompatible type "str"; expected "Job"

The Hard Part Bit

Existing (Large) Projects

1. Type common libraries --> utils.py
2. Try running the type checker
3. Type checker in CI
4. Ask people to add types to new code
5. Use MonkeyType

MonkeyType

https://github.com/Instagram/MonkeyType

1. Collects types at runtime and logs them
2. Generates type stubs
3. Can even apply back type information to your code

Stubs

```
# Stubs for math
# See: http://docs.python.org/2/library/math.html

from typing import Tuple, Iterable, SupportsFloat, SupportsInt

import sys

e = ...  # type: float
pi = ...  # type: float
if sys.version_info >= (3, 5):
    inf = ...  # type: float
    nan = ...  # type: float
if sys.version_info >= (3, 6):
    tau = ...  # type: float

def acos(x: SupportsFloat) -> float: ...
def acosh(x: SupportsFloat) -> float: ...
def asin(x: SupportsFloat) -> float: ...
def asinh(x: SupportsFloat) -> float: ...
def atan(x: SupportsFloat) -> float: ...
def atan2(y: SupportsFloat, x: SupportsFloat) -> float: ...
def atanh(x: SupportsFloat) -> float: ...
```

MonkeyType

https://github.com/Instagram/MonkeyType

```
$ pip install monkeytype

$ monkeytype run main.py

$ monkeytype stub utils
def pretty_print(job: str) -> None: ...

$ monkeytype apply utils
```

Tips & Tricks

Where did the ducks end up?

Duck typing in the typed world

```python
def duck_things(duck: Duck) -> None:
    duck.quack()
    duck.swim()
```

```python
class PharaohDuck:
    def quack(self) -> str:
        return "quaaack"
    def swim(self) -> str:
        return "🦆"
```

```python
class GeckoDuck:
    def quack(self) -> str:
        return "quack??"
    def swim(self) -> str:
        return "Gecko doesn't swim"
```

Protocols

```
duck_things(PharaohDuck())
```

```
error: Argument 1 to "duck_things" has incompatible type
"PharaohDuck"; expected "Duck"
```

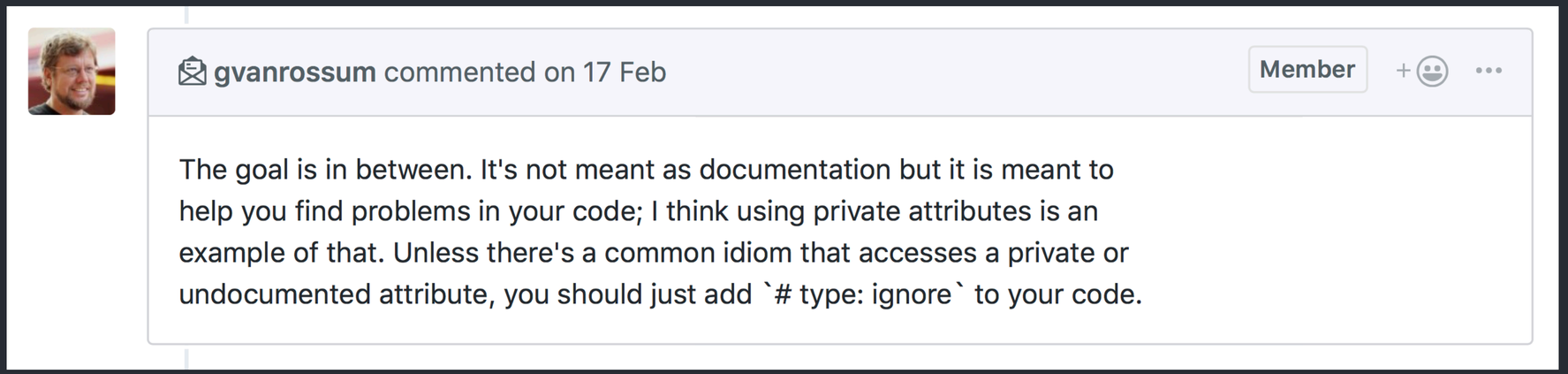
```
from typing_extensions import Protocol

class Duck(Protocol):
    def quack(self) -> str: ...
    def fly(self) -> str: ...
```

The Typeshed

- Typing for the standard library is not part of it ?!
- Separate repo: https://github.com/python/typeshed
- Bundled with your type checker
- Still far from perfect --> send a PR!

type: ignore

- Silences typing error on one statement
- Not necessarily evil
- https://github.com/python/typeshed/pull/1885



Summary

Summary

- Type annotations are Pythonic
- Start using them for the readability win
- Get the extra safety of the type checker for free
- The easy bit, new projects should be 100% typed
- The hard part, existing large codebases
  - Start small with gradual typing
  - Use MonkeyType

https://github.com/Sterbic/PyCon-AU-2018

Luka Šterbić

Software Engineer @ Facebook London

luka@fb.com

github.com/sterbic

facebook