Аннотации типов в Python

Д.В. Иртегов

НГУ 2024

Статическая типизация в Python?

- Многие видели (и некоторые использовали) примеры кода

```
#! env python3.5
from typing import List
Vector = List[float]
def scale(scalar: float, vector: Vector) -> Vector:
    return [scalar * num for num in vector]
# typechecks; a list of floats qualifies as a Vector.
new_vector = scale(2.0, [1.0, -4.2, 5.4])
```

Не совсем типизация

- Аннотация типов (type hints)
- Используется статическими анализаторами (mypy, PyCharm)
- Не влияет на исполнение
  - Это немного странно, т.к. типизация – мощный хинт оптимизатору
  - На самом деле, влияет
  - (во всяком случае, вы можете добыть эту информацию в рантайме)

```
>>> a: Optional[int] = None
>>> type(a)
<class 'NoneType'>
>>> locals()['__annotations__’]
{'a': typing.Union[int, NoneType]}
```

Что включает поддержка типизации

- Сами аннотации для переменных, параметров и возвращаемых значений
  - В том числе, объявление переменной без присваивания значения
- По умолчанию типизованные переменные non-nullable

```
a: int = None # Error
```

- Пользовательские типы (аналог typedef в C/C++)

```
from typing import NewType
UserId = NewType('UserId', int)
some_id = UserId(524313)
```

- Типизованные списки и таплы
- Интерфейсы и протоколы
- Шаблоны (generic)

Шаблоны (generics)

- Функции

```
from typing import List, TypeVar
T = TypeVar("T")
def first(container: List[T]) -> T:
    return container[0]
```

- Типы

```
from typing import Dict, Generic, TypeVar
T = TypeVar("T")
class Registry(Generic[T]):
    def __init__(self) -> None:
        self._store: Dict[str, T] = {}
```

Интерфейсы, протоколы и абстрактные классы

- Протокол: поддержка duck typing
  - Если объект имеет нужные методы/атрибуты, он считается экземпляром протокола
  - collections.abc.Iterable, Callable, Sizable
  - numbers.Number
  - По умолчанию, работает только при статической проверке
  - При определении протокола, можно включить проверку во время исполнения
- Интерфейсы и абстрактные классы
  - Классы с абстрактными методами (нельзя инстациировать)
  - Нужно наследоваться от них
  - Разработчик абстрактного класса также может реализовать duck typing во время исполнения (__subclasshook__(cls, subclass))

Что еще включает поддержка типизации

- Тип Any
- Объединение типов (немного похоже на C union)
- Опциональный (nullable) тип: эквивалент Union[X, None]

```
def foo(arg: Optional[int] = None) -> None:
```

- Literal

```
MODE = Literal['r', 'rb', 'w', 'wb']
def open_helper(file: str, mode: MODE) -> str:
  ...
open_helper('/some/path', 'r') # Passes type check
open_helper('/other/path', 'typo') # Error in type checker
```