Роман Соляник

Руководитель разработки DBaaS, Тинькофф

solyan

ТИНЬКОФФ

Как сделать статический анализатор типов своим лучшим другом

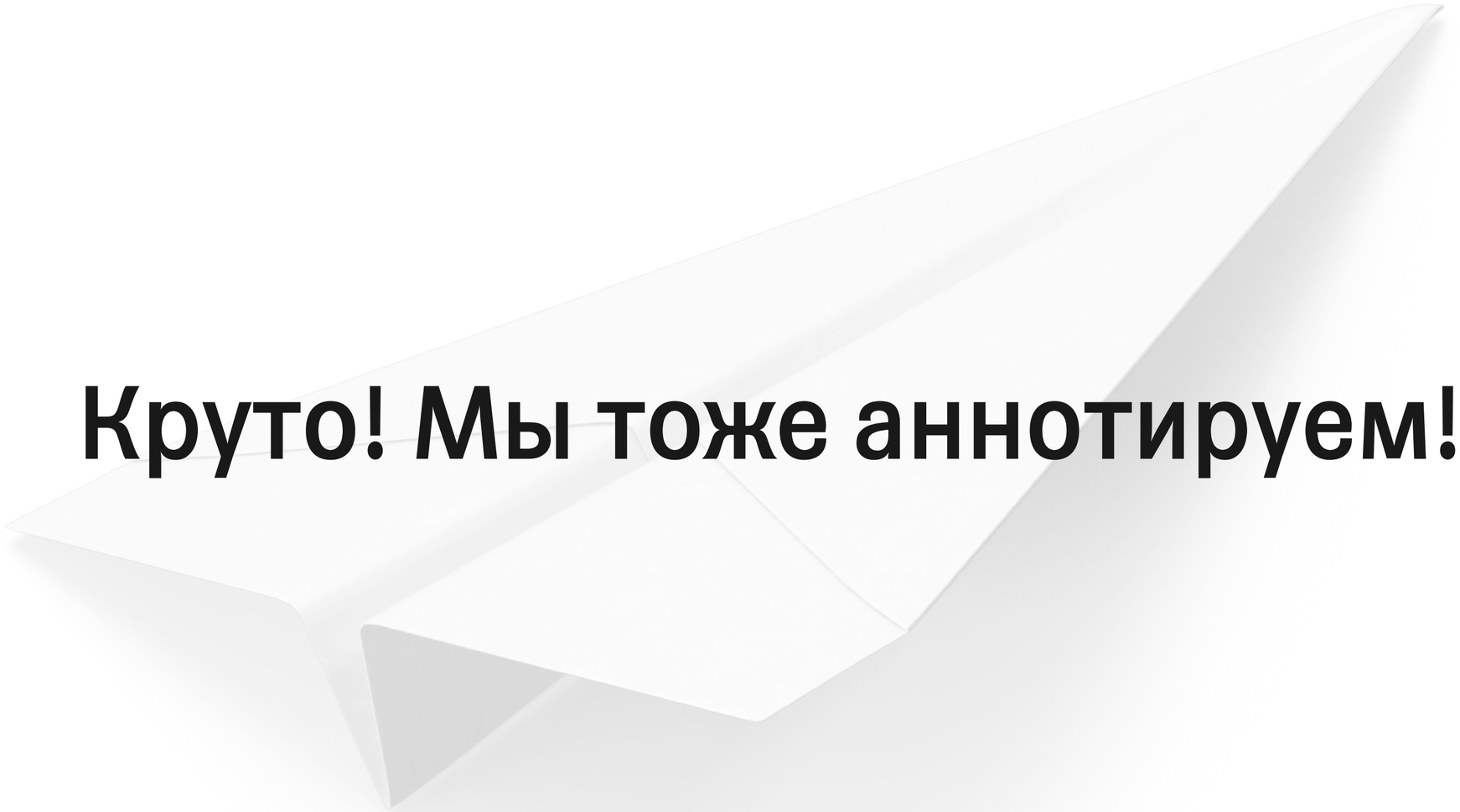
Зачем аннотировать типы в Python

Подсказки IDE, удобный API

Подсказки нейросетей

Меньше сюрпризов в рантайме

Облегчение сопровождения кода



Круто! Мы тоже аннотируем!

А потом такие

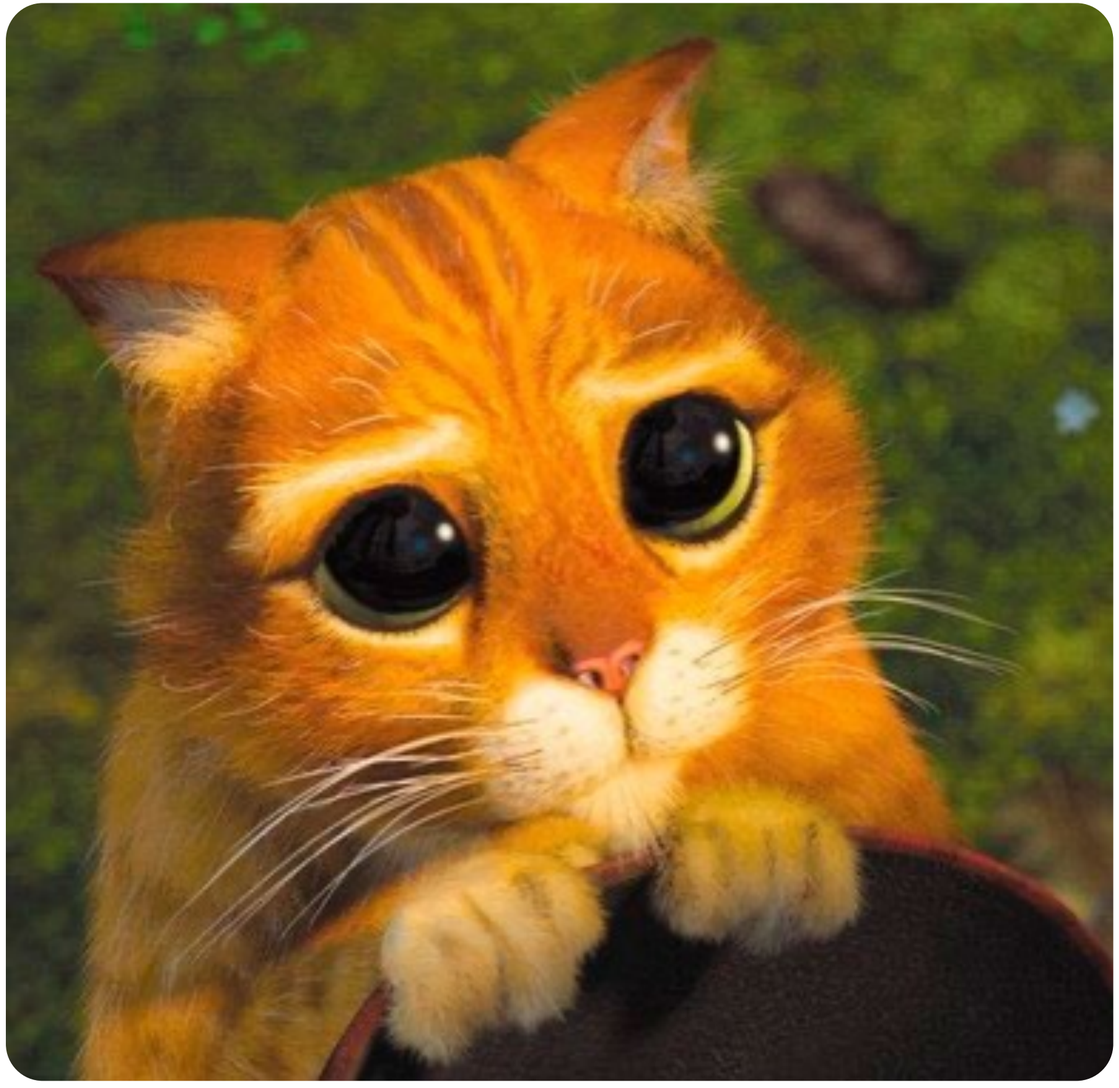
```
def find_smallest_item(items: list) -> Any: ...
```

```
def calculate_overall_cost(orders: dict[str, int | None]) -> int:
    overall_cost = 0
    for cost in orders.values():
        overall_cost += cost
    return overall_cost
```

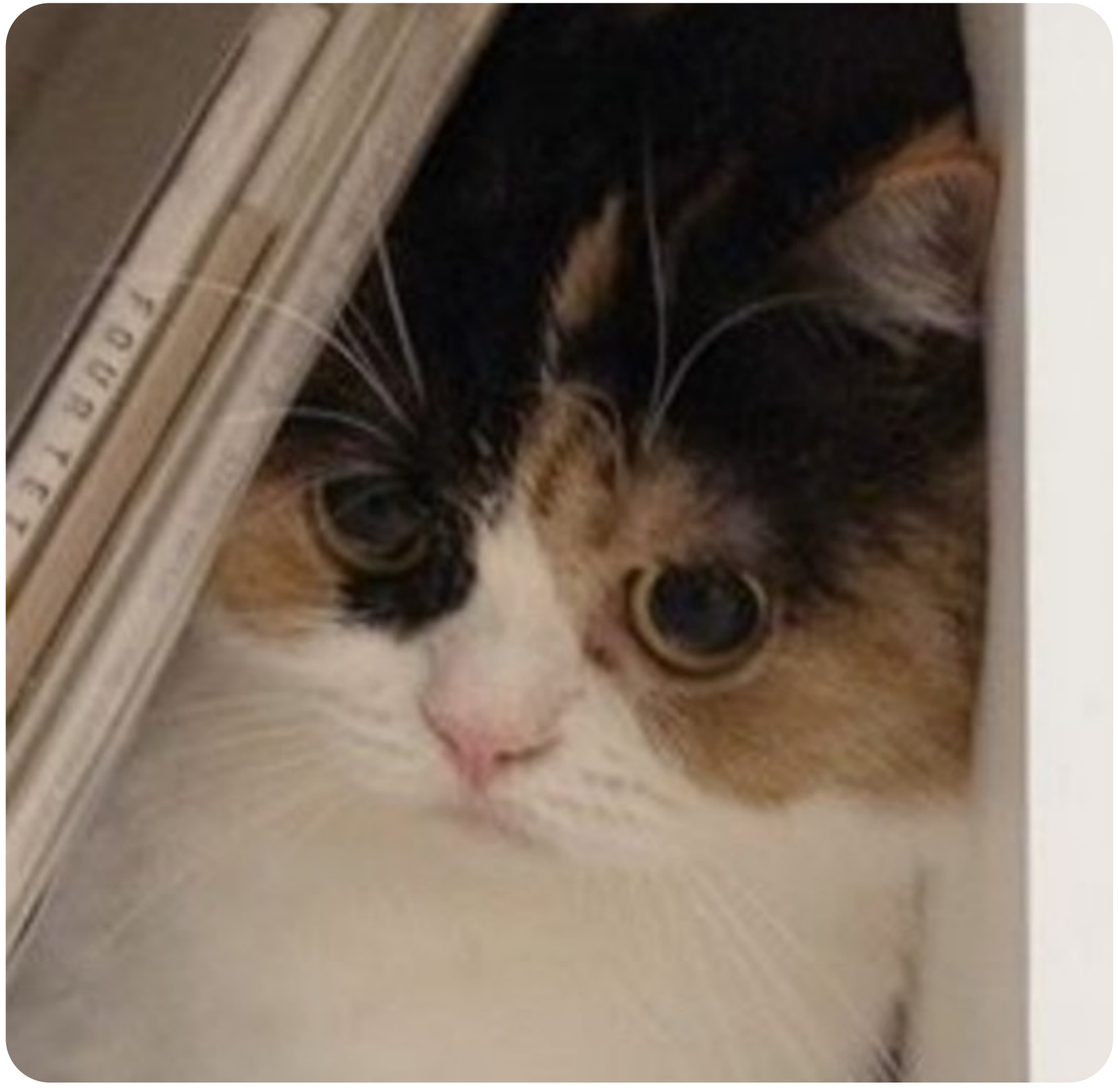
Запускаем анализатор

```
def find_smallest_item(items: list) → Any: ...
error: Missing type parameters for generic type "list"  [type-arg]

def find_smallest_item(items: list[Item]) → Item: ...
```

```
def calculate_overall_cost(orders: dict[str, int | None]) → int:
    overall_cost = 0
    for cost in orders.values():
        overall_cost += cost
    return overall_cost
error: Unsupported operand types for + ("int" and "None")  [operator]
def calculate_overall_cost(orders: dict[str, int | None]) → int: ...
    overall_cost = 0
    for cost in orders.values():
      if cost is not None:
```



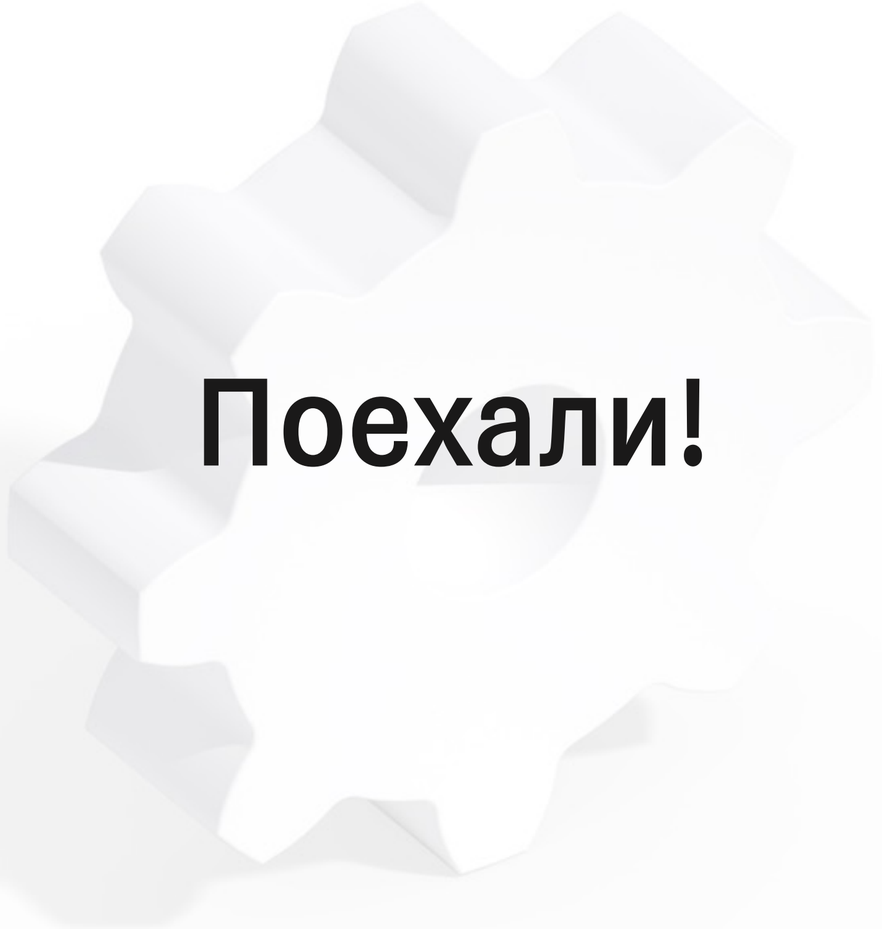
А можно нам эти плюшки?



Только с анализатором типов

Что делать?

- Внедрить type-checker в процесс разработки
- Проставлять полные аннотации
- Понимать ограничения анализаторов
- Не забывать о здравом смысле



Поехали!

Внедряем статический анализ типов

01

Выбираем type-checker

- Mypy
- Pyright
- Pyre
- Pytype

02

Настраиваем его

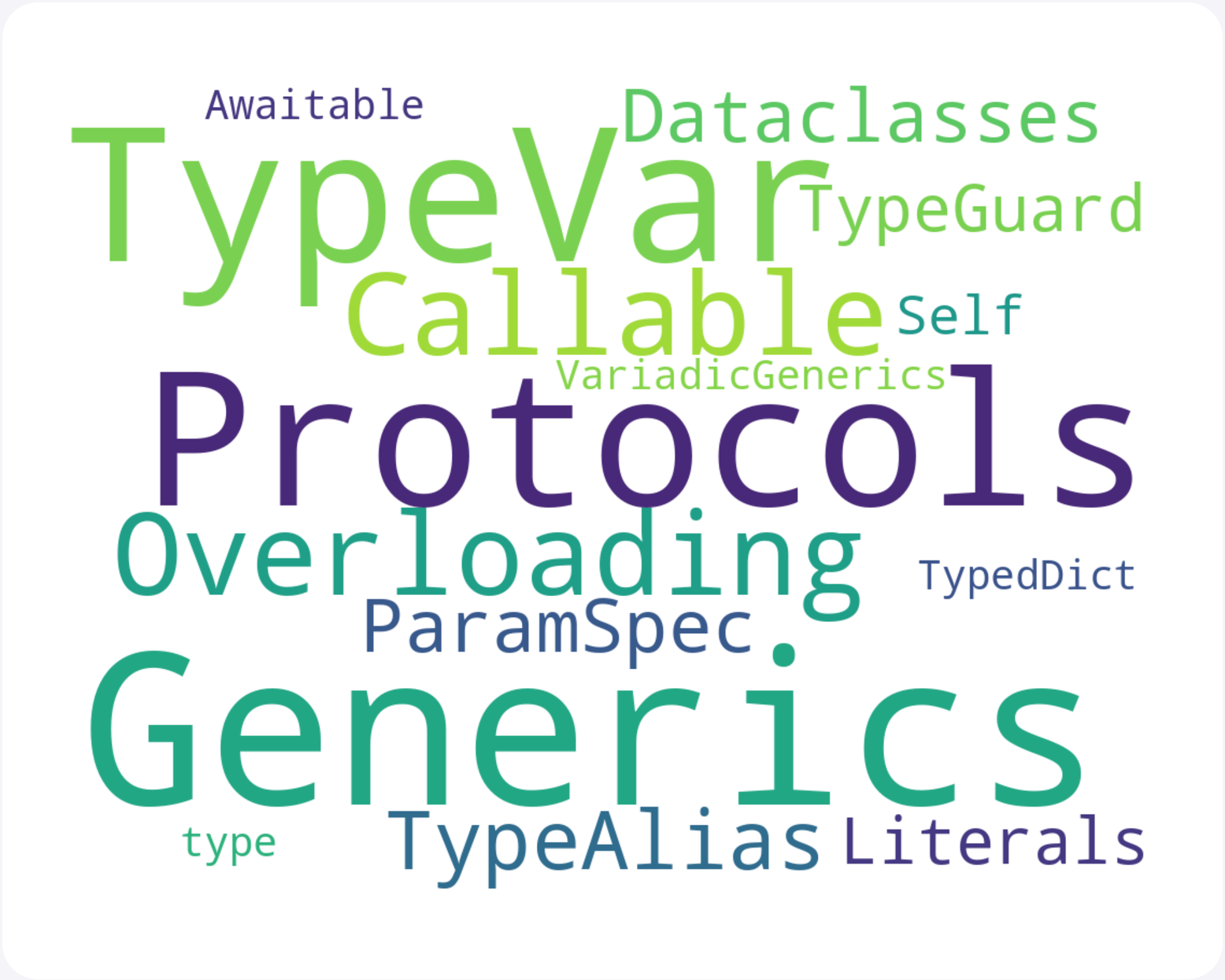
- Strict-режим
- Per-module настройки
- Конфиг – часть репы

03

Делаем частью CI

- Аннотируем весь новый код
- Pre-commit (опц.)
- git bash hooks
- CI-Pipeline

Аннотируем


Awaitable
Dataclasses
TypeVar
TypeGuard
Callable
Self
VariadicGenerics
Protocols
Overloading
TypedDict
ParamSpec
Generics
type
TypeAlias
Literals

Зависимость входных и выходных типов

```
def find_smallest_item(items: list[Item]) → Item:
    return min(items, key=lambda item: item.get_size())

T = TypeVar("T")
def find_smallest_item(items: list[T]) → T:
    return min(items, key=lambda item: item.get_size())

error: "T" has no attribute "get_size"  [attr-defined]

class SupportsGetSize(Protocol):
    def get_size(self) → float: ...


def find_smallest_item(items: list[SupportsGetSize]) → SupportsGetSizeT:
    return min(items, key=lambda item: item.get_size())

Success: no issues found
```

Generics – Python 3.12

```
def find_smallest_item[T: SupportsGetSize](items: list[T]) → T:
    return min(items, key=lambda item: item.get_size())

error: PEP 695 generics are not yet supported  [valid-type]
```

Support for PEP 695 #15238

Open · 10 tasks · erictraut opened this issue on May 14 · 1 comment

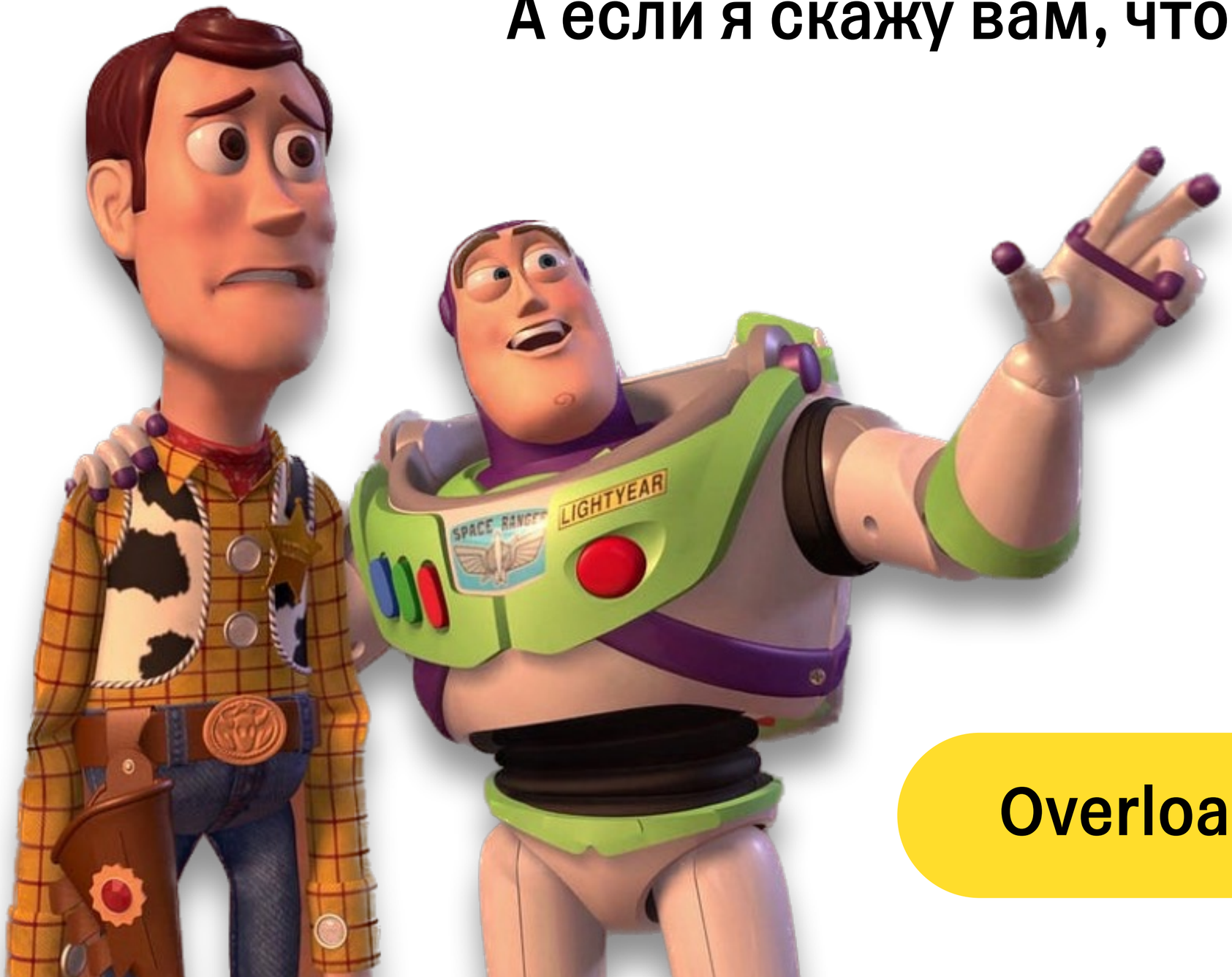
https://github.com/python/mypy/issues/15238

А можно overload-ить?

```python
class Item:
    def get_size(self) -> float: ...

@overload
def add(item_1: int, item_2: int) -> int: ...

@overload
def add(item_1: Item, item_2: Item) -> float: ...


def add(item_1: Item | int, item_2: Item | int) -> float | int:
    if isinstance(item_1, int) and isinstance(item_2, int):
        return item_1 + item_2
    if isinstance(item_1, Item) and isinstance(item_2, Item):
        return item_1.get_size() + item_2.get_size()
    raise TypeError("...")
```

Deferred PEP-3124

А если я скажу вам, что



Overload-ы повсюду

typeshed

https://github.com/python/typeshed/

```
@overload
def pow(base: int, exp: int, mod: int) → int: ...
@overload
def pow(base: int, exp: Literal[0], mod: None = None) → Literal[1]: ...
@overload
def pow(base: int, exp: _PositiveInteger, mod: None = None) → int: ...
@overload
def pow(base: int, exp: _NegativeInteger, mod: None = None) → float: ...
@overload
def pow(base: int, exp: int, mod: None = None) → Any: ...
@overload
def pow(base: _PositiveInteger, exp: float, mod: None = None) → float: ...
@overload
def pow(base: _NegativeInteger, exp: float, mod: None = None) → complex: ...
@overload
def pow(base: float, exp: int, mod: None = None) → float: ...
@overload
def pow(base: float, exp: complex | _SupportsSomeKindOfPow, mod: None = None) → Any: ...
@overload
def pow(base: complex, exp: complex | _SupportsSomeKindOfPow, mod: None = None) → complex: ...
```

Literal

```
@overload
def pow(base: int, exp: Literal[0], mod: None = None) → Literal[1]: ...
```

```
def open(  # Unbuffered binary mode: returns a FileIO
    file: FileDescriptorOrPath,
    mode: OpenBinaryMode,  # Literal[rb, ...]
    buffering: Literal[0],
    ...
) → FileIO: ...


@overload  # Buffering is on: return BufferedRandom, BufferedReader, or BufferedWriter
def open(
    file: FileDescriptorOrPath,
    mode: OpenBinaryModeUpdating,  # Literal[wb+]
    buffering: Literal[-1, 1] = -1,
    ...
) → BufferedRandom: ...
```

Видишь читаемость?

```python
class UserAccount(ApiModel): ...
class ServiceAccount(ApiModel): ...

class Resource(GenericApiModel, Generic[ResourceT]): ...
class Collection(GenericApiModel, Generic[ResourceT]): ...

@router.get("/")
def list_accounts() -> AccountCollCollection[Resource[UserAccount] | Resource[ServiceAccount]]:
    ...
```

И я не вижу

А она есть!

```python
from typing import TypeAlias

AccountResource: TypeAlias = Resource[UserAccount] | Resource[ServiceAccount]
AccountCollection: TypeAlias = Collection[AccountResource]

@router.get("/{resource_id}")
def get_account(resource_id: str) → AccountResource: ...

@router.get("/")
def list_accounts() → AccountCollection: ...
```

A B 3.12

```
type AccountResource = Resource[UserAccount] | Resource[ServiceAccount]
type AccountCollection = Collection[AccountResource]
```

А как анализатор понимает мой код?

А как анализатор понимает мой код?

```
def exactly_one_to_float(a: int | None, b: int | None) → float:
    if a is None and b is None:
        raise TypeError("ОБА НЕ определены")
    elif a is not None and b is not None:
        raise TypeError("ОБА Определены")
    # один из них определен

    elif a is not None:  # a определен
        return float(a)

    return float(b)  # b определен

error: Argument 1 to "float" has incompatible type "int | None";
       expected "SupportsFloat | SupportsIndex | str | Buffer"  [arg-type]
```

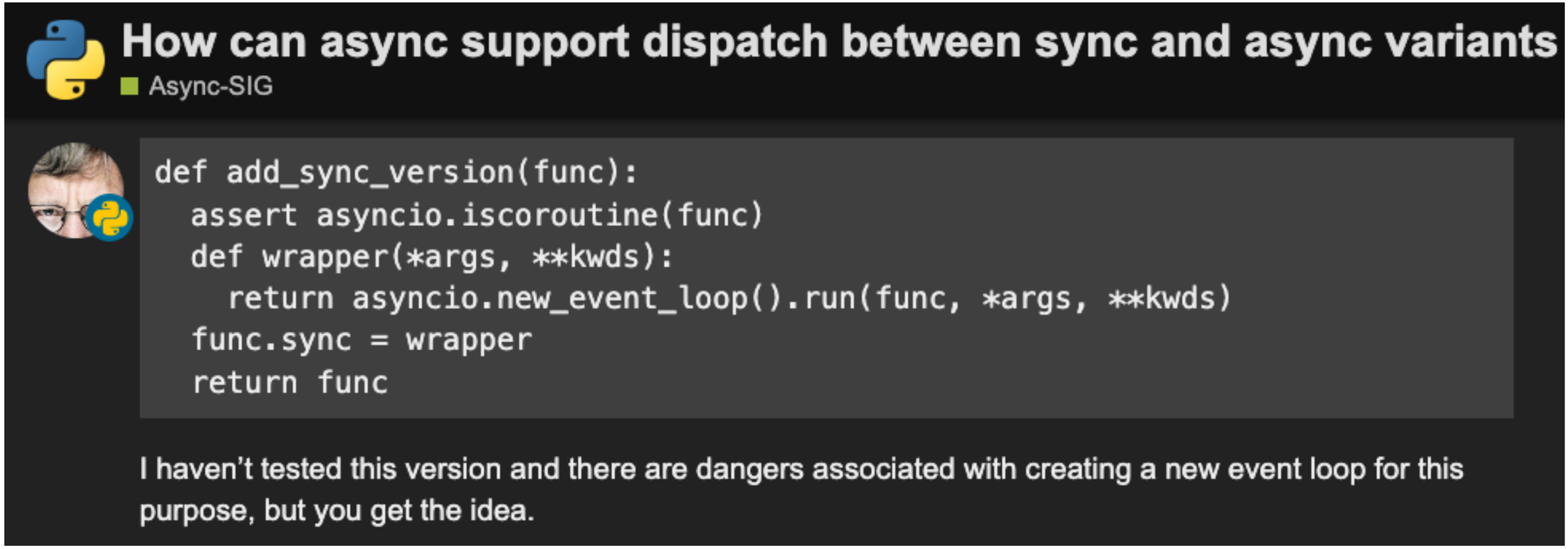
Решаем

```
def exactly_one_to_float(a: int | None, b: int | None) → float:
    if a is None and b is None:
        raise TypeError("Оба определены")

    elif a is not None and b is not None:
        raise TypeError("Ни один определн")

    # один из них определен
    elif a is not None:  # a определен
        return float(a)

    elif b is not None:  # b определен
        return float(b)
    raise TypeError

Success: no issues found in 1 source file
```

ОтDRYим код?

```
def search(query: Query) → Result:
    ...

async def search(query: Query) → Result:
    ...

asyncio.run(search(query))  # 😞

await search(query)
search(query)
```

С этим вопросом я вышел в интернт



https://discuss.python.org/t/how-can-async-support-dispatch-between-sync-and-async-variants-of-the-same-code/15014/7

Сделаем асинхронное синхронным?

```python
Param = ParamSpec("Param")
Result = TypeVar("Result")
def synchronously_callable(
  async_func: Callable[Param, Awaitable[Result]]
) -> Callable[Param, Awaitable[Result]]:
    @wraps(async_func)
    def wrapper(*args: Param.args, **kwargs: Param.kwargs) -> Result:
        return asyncio.run(async_func(*args, **kwargs))
    async_func.sync = wrapper
    return async_func
```

miro

Сделаем асинхронное синхронным?

```
@synchronously_callable
async def search(query: Query) → Result:
    ...

await search(query)
search.sync(query)

error: "Callable[[Query], Awaitable[Result]]" has no attribute "sync"
```

Это вообще лечится?

```
@synchronously_callable
class search:
    async def __call__(self, query: Query) → Result:
        ...

await search()(query)

@synchronously_callable
class search:
    async def __new__(cls, query: Query) → Result:  # 😣
        ...
```

А если у меня класс?

```
class ApiClient:
    async def get_user(self, user_id: str) → User:
        ...

    async def list_users(self) → list[User]:
        ...
```

Сгенерируем синхронные варианты

```
class AsyncToSyncMetaclass(type):
    def __new__(mcs, clsname: str, bases: Any, clsdict: Any) -> Any:
        new_clsdict = {}
        for attr_name, attr_value in clsdict.items():
            if inspect.iscoroutinefunction(attr_value):

                def sync_method(self: Any, *args: Any, **kwargs: Any) -> Any:
                    return asyncio.run(attr_value(self, *args, **kwargs))

                new_clsdict[f"{attr_name}_sync"] = sync_method

        return super().__new__(mcs, clsname, bases, new_clsdict)
```

Попробуем?

Попробуем?

```
class ApiClient(metaclass=AsyncToSyncMetaclass):
    async def get_user(self, user_id: str) → User:
        return ...

    async def list_users(self) → list[User]:
        return ...

client = ApiClient()
client.get_user_sync("id")

error: "Sync" has no attribute "go_sync"  [attr-defined]
```

Я знаю, мы можем лучше

```
class ApiClient(metaclass=AsyncToSyncMetaclass):
    async def get_user(self, user_id: str) → User:
        return ...  # имплементация

    async def list_users(self) → list[User]:
        return ...  # имплементация


    def get_user_sync(self, user_id: str) → User:
        ...  # Только сигнатура

    def list_users_sync(self) → list[User]:
        ...  # Только сигнатура
```

А иногда и такое встречается

```python
from pydantic import BaseModel

T = TypeVar("T", bound=BaseModel)
def search(request: str, response_schema: Type[T]) -> list[T]:
    response = requests.post(Endpoints.search, data={"request": request}).text
    # '[{"user_id": "dead-beef", "session_count": 10}]'
    return TypeAdapter(list[response_schema]).validate_json(response)

class UserSessionCount(BaseModel):
    user_id: str
    session_count: int

search("user_id, session_count", UserSessionCount)

error: Variable "response_schema" is not valid as a type  [valid-type]
```

https://github.com/python/mypy/issues/14458

Ограничения

Плохо работают трюки с динамичностью

Тулинг всегда в положении догоняющего, но он пытается

А статики мне точно хватит?

- pydantic
- marshmallow
- beartype
- typeguard (typechecked)

Все любят Pydantic?

Сначала на API

Потом прикручивают к БД моделям

Потом и сервисные схемы заодно

На больших RPS - может быть накладно

Выводы

- Код понятнее и удобнее
- Ошибки быстрее исправляются и сложнее добавляются
- Меньше ошибок доживает до рантайма
- Не забываем о здравом смысле
- Инструментарий неидеален, но постоянно развивается

ТИНЬКОФФ

Спасибо!