

Суперфункции

Евгений Блинов для Moscow Python

Обо мне

Я - Евгений Блинов.

Пишу на Python, иногда в Open Source.

Основатель **The Mutating Company**.



Контакты

Телега: [@blinof](https://www.t.me/blinof)

Гитхаб: github.com/pomponchik

Личный сайт: pomponchik.org

The Mutating Company: mutating.tech



Суперфункции?

transfunctions

- Весь мой доклад про главную фичу библиотеки **transfunctions**: суперфункции
- github.com/pomponchik/transfunctions



Пример со `sleep()`

```
from asyncio import run
from supertime import supersleep

supersleep(5) # sleeps 5 sec.
run(supersleep(5)) # sleeps 5 sec., but ASYNCHRONOUSLY.
```



[github.com/](https://github.com/pomponchik/supertime)

[pomponchik/supertime](https://github.com/pomponchik/supertime)

Что под капотом?

```
from transfunctions import superfunction, sync_context, async_context, await_it

@superfunction(tilde_syntax=False)
def supersleep(number: int | float) -> NoReturn:
    with sync_context:
        sync_sleep(number)
    with async_context:
        await_it(async_sleep(number))
```

Как работает @superfunction?

- Под капотом:
 - Кодогенерация через AST
 - “Препарирование” сгенерированных функций
 - Роутинг на основе счетчика ссылок



[pomponchik.org/talks/
python-doesnt-eat-itself-anymore-how-to-defeat-fragmentati
on-on-sync-and-async/](https://pomponchik.org/talks/python-doesnt-eat-itself-anymore-how-to-defeat-fragmentation-on-on-sync-and-async/)

Шаг 1: генерация кода через AST

- Берем исходную функцию-шаблон
- Инспектируем исходный код
- Парсим его и получаем AST
- Преобразуем AST как нам нужно
- “Компилируем” AST в другую функцию
- Результат будет сломан в куче мест:
 - Отвалятся глобальные переменные и замыкания
 - Будет потерян контекст модуля

Шаг 2: препарируем новую функцию

- Наша задача – заставить ее работать как оригинал
- Нужно подсунуть исходные глобалы и нонлокалы
- Подробнее – на докладе с [PyCon 2025](#)

Шаг 3: прикручиваем роутинг

- Функция должна сама понимать, как она была использована
 - Обычная функция
 - Асинхронная
 - Генераторная (до кучи, тоже поддерживается)
- 2 типа роутинга:
 - На базе счетчика ссылок (уже видели в примере с **supertime**)
 - На базе тильда-синтаксиса

Роутинг на базе счетчика

- Функция возвращает специальный объект (трейсер)
- Объект умеет “работать” корутиной или генератором и запоминать это
- Когда за объектом приходит счетчик ссылок, он “вспоминает” свою историю и может запустить синхронный вариант функции
- Плюс:
 - **Нативный синтаксис**
- Минусы:
 - **Нельзя возвращать значения**
 - **Нельзя поднимать исключения**

Роутинг на базе тильда ~ синтаксиса

- Объект-трейсер все еще есть
- Не используется счетчик ссылок
- Работает через перегрузку оператора
- Плюсы:
 - Можно возвращать значения
 - Можно поднимать исключения

Как выбрать роутинг?

```
@superfunction(tilde_syntax=False)
```

```
my_superfunction() # Роутинг через счетчик  
result = ~my_superfunction() # Тильда-роутинг
```

Зачем и кому все это нужно?

- Разработчикам библиотек с дублирующимся API

Что выбрать: трансфункции или суперфункции?

- Суперфункция – это обертка над трансфункцией
- Для разработки универсальных декораторов – трансфункции
- Для обеспечения единообразного API библиотеки - суперфункции

Третий API



Все!

Контакты

Телега: @blinof

Гитхаб: github.com/pomponchik

Личный сайт: pomponchik.org

The Mutating Company: mutating.tech



За красивый шаблон для презентации благодарность компании Evrone ❤️