14. Типови анотации и linting

15 април 2026

Първо, въпрос

- С кои руни се прави Enigma runeword?
- Jah, Ith, Ber
- Простено ви е, ако не знаете, просто исках да ви припомня какво правихме предния път
- И по тази тема, малък (поискан) reminder за виртуалните среди

Типови анотации

- Добавят мета-информация за аргументи, връщани стойности и променливи
- Не влияят на изпълнението на кода (игнорират се от интерпретатора)
- Подобряват:
  - Поддръжка на големи проекти
  - Интеграция с инструменти (IDE, линтери, статични анализатори)
  - Качество на auto-complete и документация
- Пример:

```python
def add(x: int, y: int) -> int:
    return x + y
```

А ако имаме повече от един тип?

```python
def parse_int(s: str) -> int | None:
    ...


def add(x: int | float, y: int | float) -> float:
    ...
```

- `T | None` или `Optional[T]` *(преди 3.11)* → стойността може да е `T` или `None`
- `A | B` или `Union[A, B]` → стойността може да бъде или `A`, или `B`
- Може да са много повече, но тогава има други решения, вместо да пишете всичките експлицитно във всяка една анотация

Колекции и типовете на данните им

```python
from typing import Dict, List

grades: Dict[str, List[int]] = {
    "Ivan": [6, 5, 6],
    "Maria": [4, 5],
}
```

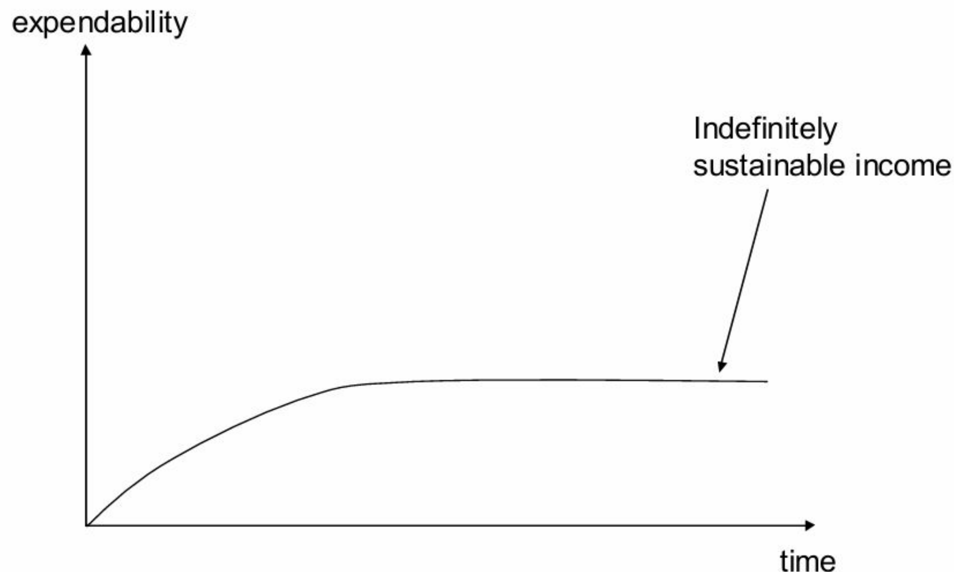
- Позволяват да опишем какви елементи съдържа една структура
- Наример речник от низове към списъци с цели числа

Хубаво де, но защо?

- Статично (**MyPy**, **Pyright**) - проверява типовете по време на разработка
- Runtime (**Pydantic**, **beartype**) - валидира стойности в движение
- Arguably най-важното - IDE подсказки!
- Безценно при рефакториране
- А и помагат за документацията
- Нека да си сложим малко анотации и ще видим разликата в IDE-то

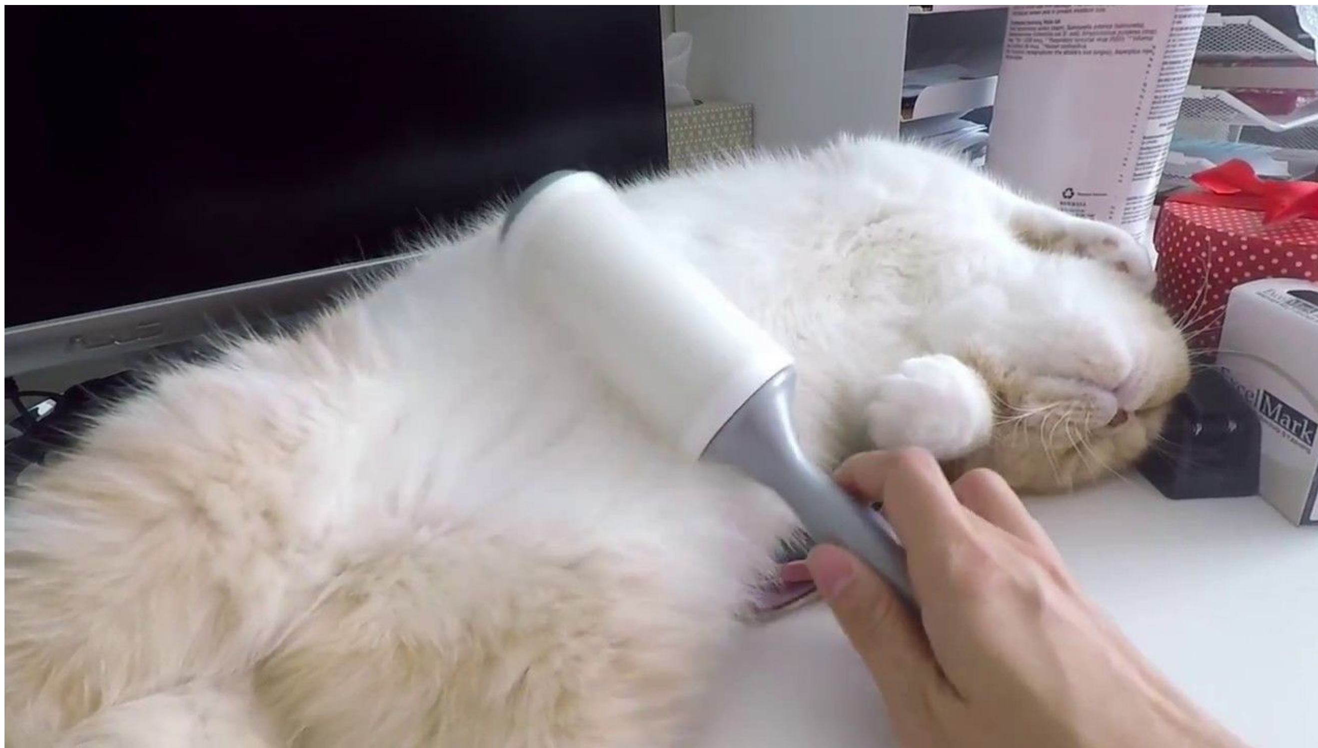
По темата с рефакторирането

- Refucktoring is the process of taking a well designed piece of code and, through a series of small, reversible changes, making it completely unmaintainable by anybody except yourself.
- *“This thing is fucked, thanks to Richo's refucktoring.”*
- В същото време може да е успешен бизнес модел.



Lint

Linting



В контекста на програмирането?

- Статичен анализ на кода
- Помага да открием:
  - Стилистични проблеми
  - Проблеми с типизацията
  - Потенциални проблеми / бъгове
- В Python:
  - PEP 8 – Style Guide for Python Code
  - Много различни линтъри:
    - pylint
    - flake8
    - black
    - ruff
    - И още хиляда неща с много различни правила и проверки
- Има и нещо различно - formatters - работят на базата на линтъри и форматират кода съобразно

pylint

- pylint е най-популярният по исторически причини linter
- Повечето други линтъри преизползват голяма част от неговите правила, защото е the OG и имплементира повечето PEP8 проверки
- Можете да си го инсталирате с?
- Правилно, с pip:

```
python -m pip install pylint
```

- Оттам насетне е лесно:

```
pylint some_file.py
```

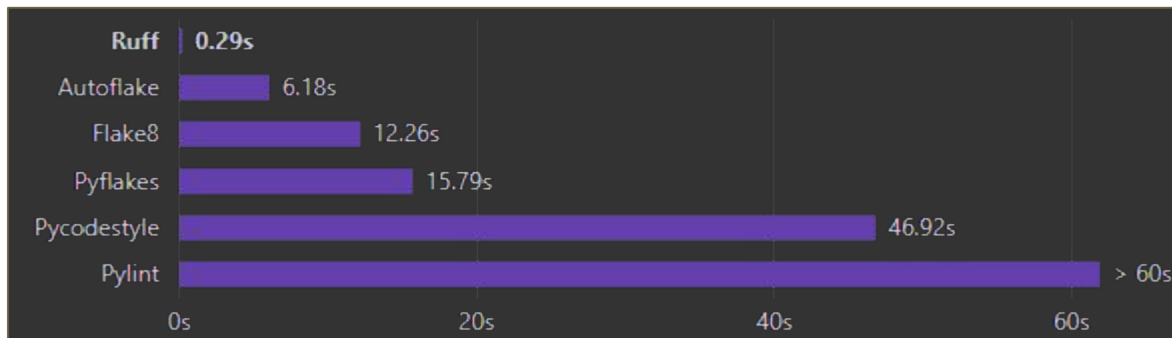
- Може и да се конфигурира… грозно…

Ако предпочитате по-грубо

- Препоръчваме rough!
- Пардон, ruff
  - Страшно бърз
  - Включва правила от различни линтъри и форматъри - flake8, black, etc.
  - Включва много правила и извън тях
  - Лесна интеграция с VS Code

```
python -m pip install ruff
```

- `ruff check`
- `ruff format`



Конфигуриране

- Също доста лесно - трябва да променим `ruff.toml`
- Например:

```
[lint]
select = ["ALL"]
```

- Или пък:

```
[lint]
select = ["E", "F"]
ignore = ["F401"]
```

- И още дузина неща, които можете да настроите - exclude, line-length, target-version, documentation style, formatting rules и т.н.

За любознателните

- git pre-commit
- Hook, който ни позволява да изпълним някакви действия преди commit
- Като например да си форматираме кода, да му пуснем статична проверка и много други
- По тази тема - толкоз

Въпроси?