Programování 2: Typové anotace

Martin Mareš
mj@ucw.cz

Katedra Aplikované Matematiky
MFF UK Praha

2021

Proč a nač?

Jak se programovací jazyky chovají k typům dat:

- **statické** typování – typ mají proměnné a funkce
- **dynamické** typování – typ mají hodnoty

Obojí má své výhody i nevýhody.

K čemu třeba jsou užitečné statické typy:

- Lepší dokumentace
  (např.: funguje **math.exp** na komplexní čísla?)
- Kontrola konzistence typů před spuštěním programu
- Napovídání v editorech

Dnešní Python umí kombinovat statické a dynamické typy . . .

Typové anotace

```
def f(x: int, y: str) -> int:
    return x + len(y)
```

Čteme: **f** je funkce, parametr **x** je typu **int**, **y** typu **str**,
vrací výsledek typu **int**.

```
a: int = 42
```

Proměnné **a** jsme deklarovali typ **int**.

Python sám typy nekontroluje (to dělají samostatné type checkery),
ale pamatuje si je:

```
>>> f.__annotations__
{'x': int, 'y': str, 'return': int}
```

```
>>> help(f)
Help on function f in module __main__:
f(x: int, y: str) -> int
```

Složitější typy

```
from typing import List
def g(seznam: List[str]) -> None:
```

Parametrizovaný typ pro seznamy.

```
from typing import Optional
def center(s: str, w: Optional[int]=None) -> str:
```

Buď **int** nebo **None** (též chybí-li parametr při volání).

```
class Třída:
    def metoda(self, něco: int):
```

self automaticky dostane typ **Třída**.

```
from typing import Sequence, TypeVar
T = TypeVar("T")
def choose(s: Sequence[T]) -> T:
```

Funguje pro jakoukoliv homogenní posloupnost (seznam/tuple/...).