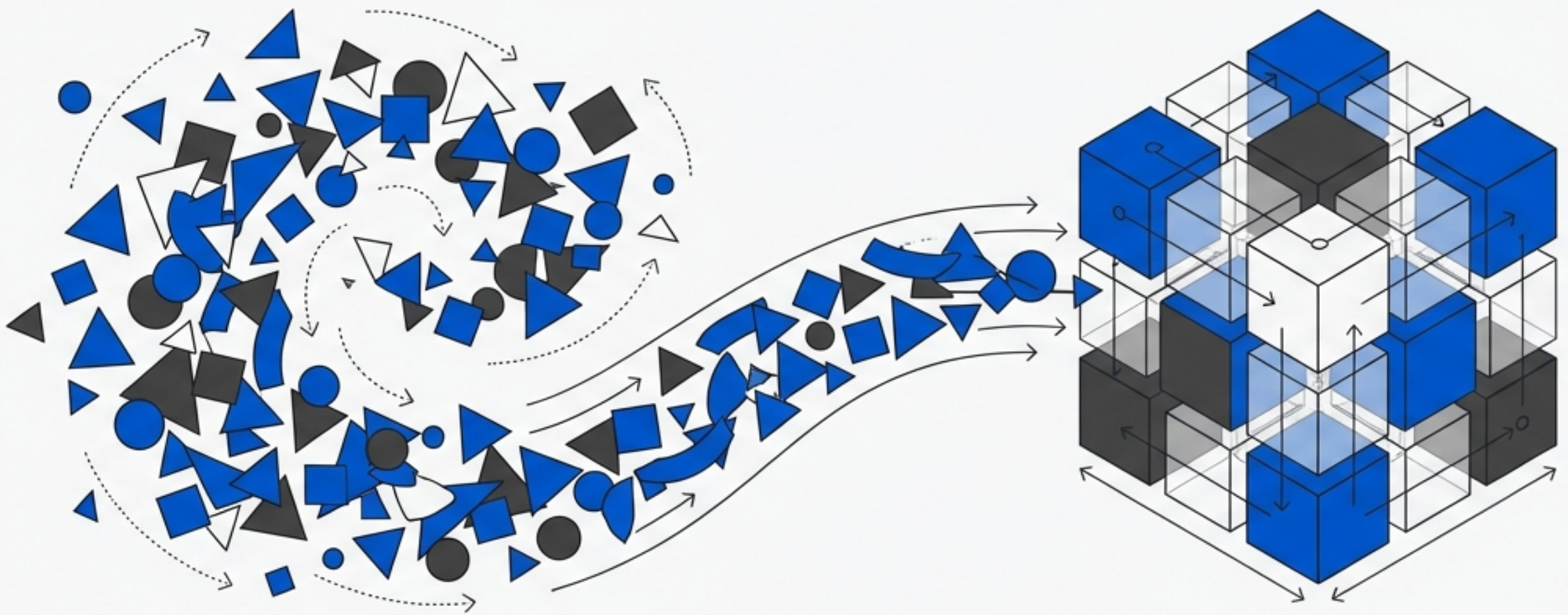
Úvod do objektově orientovaného programování v Pythonu



Od chaosu k elegantnímu kódu

```
class Object(ABC): ...
```

Problém: Chaos samostatných proměnných

Code Grey

```
xA = 3.5
yA = 2.0
xB = 2
yB = 4

# Funkce je oddělená od dat
def vzdalenost(x, y):
    return math.sqrt(x**2 + y**2)
```

Critique

$$d = \sqrt{x^2 + y^2}$$

- Data (souřadnice) a funkce (výpočet) jsou oddělené.

- Ztrácíme logickou vazbu: Co když máme 100 bodů?

- Nutnost tvořit x1...x100, y1...y100.

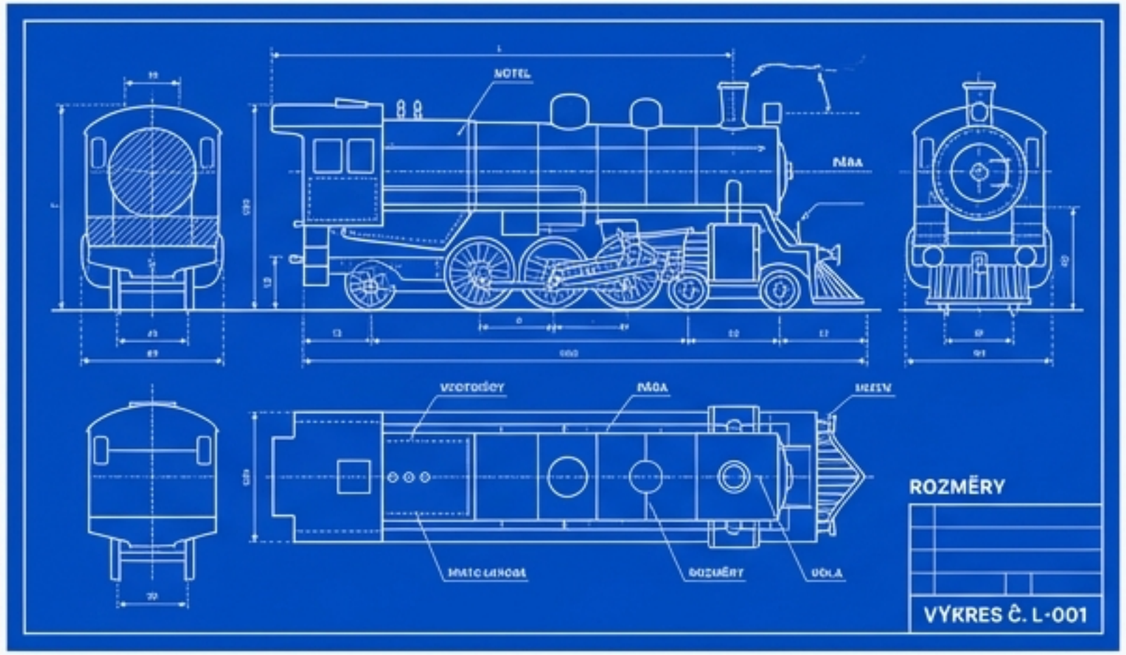
- Kód je nepřehledný a těžko se spravuje.

Řešení: Objektově orientovaný přístup

"OOP je způsob myšlení, kde program modeluje reálný svět. Místo volných proměnných máme jeden objekt, který si svá datat nese s sebou."

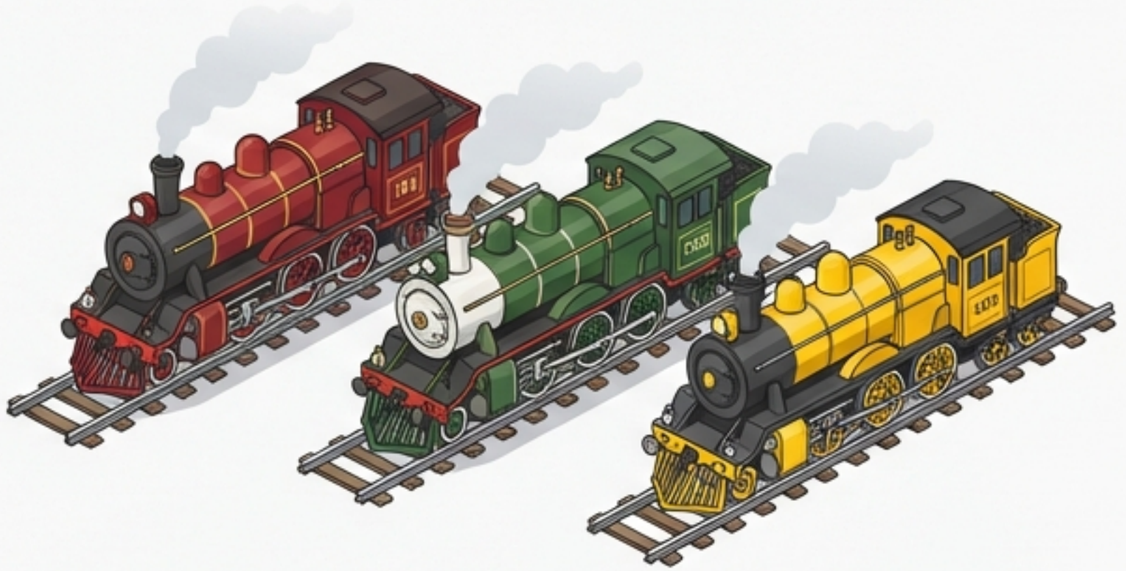
Analogie: Výkres vs. Lokomotiva

Třída (Class)



Technický výkres. Šablona.
Definuje, co je to lokomotiva.
Výkres sám o sobě nejezdí.

Objekt (Instance)



Konkrétní stroje vyrobené podle výkresu. Každý má své unikátní číslo.
Můžeme jich vyrobit tisíce.

Píšeme 'Výkres' v Pythonu

Definice nové třídy (CamelCase)

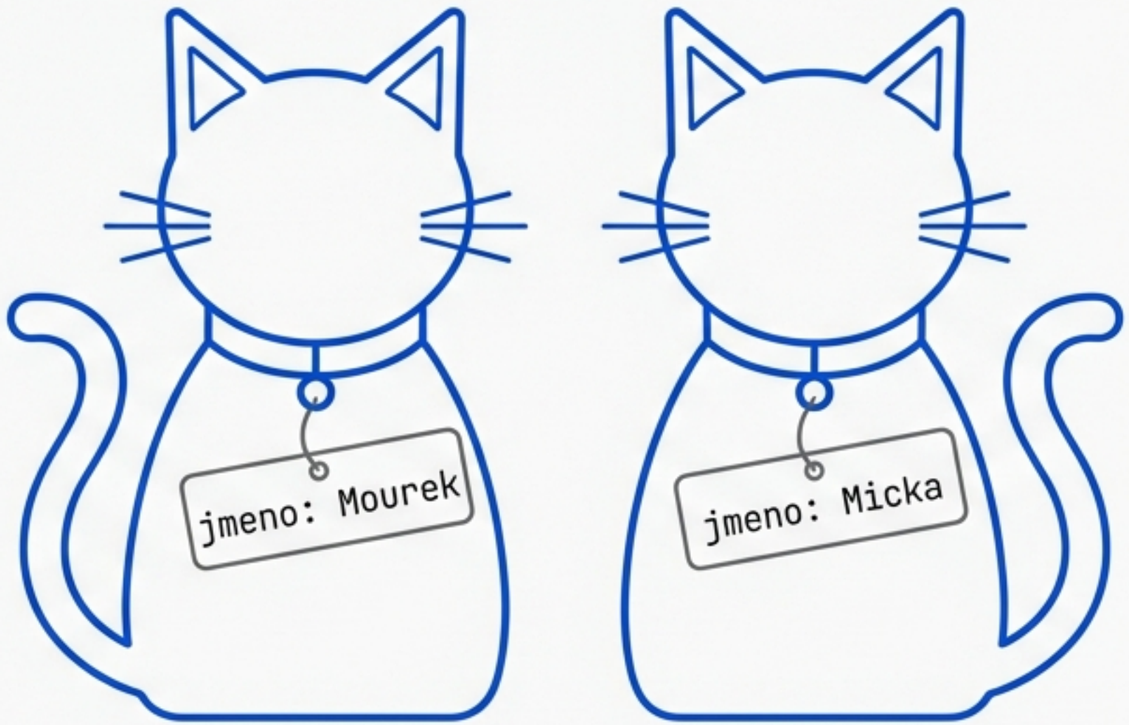
```
class Bod:
    def vzdalenost(self):
        return math.sqrt(self.x**2 + self.y**2)
```

Definice metody (funkce uvnitř třídy)

Třída Bod je náš vzor. Zatím jsme nevytvořili žádný konkrétní bod, jen jsme popsali, jak se bude chovat.

Vytvoření objektu a Atributy

```
class Kotatko:
    pass

# Vytvoření instance (zrození)
mourek = Kotatko()
micka = Kotatko()

# Nastavení atributů (data)
mourek.jmeno = 'Mourek'
micka.jmeno = 'Micka'
```



Tečková notace (.jmeno) nám umožňuje přistupovat k datům uvnitř konkrétního objektu.

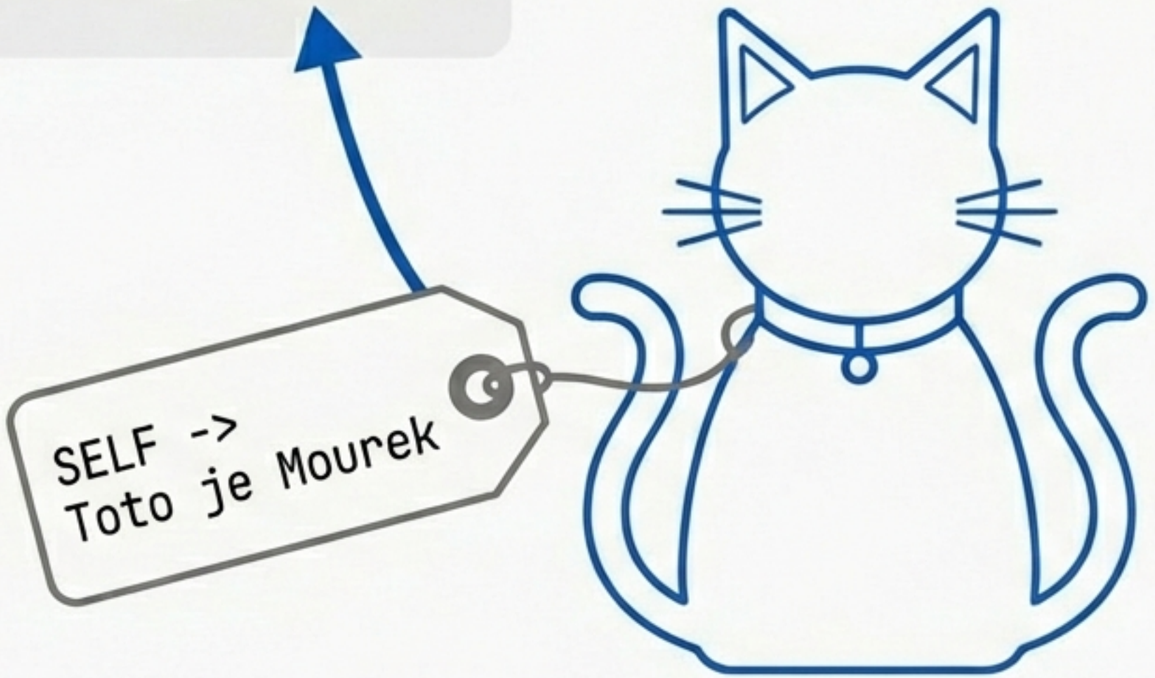
Metody: Chování objektu

```
class Kotatko:
    def zamnoukej(self):
        print('Mňau!')

mourek = Kotatko()
mourek.zamnoukej()
```

Volání: `objekt.metoda()`. Příkaz 'Mourku, zamňoukej!'.

Tajemství parametru self

```
def zamnoukej(self):
```

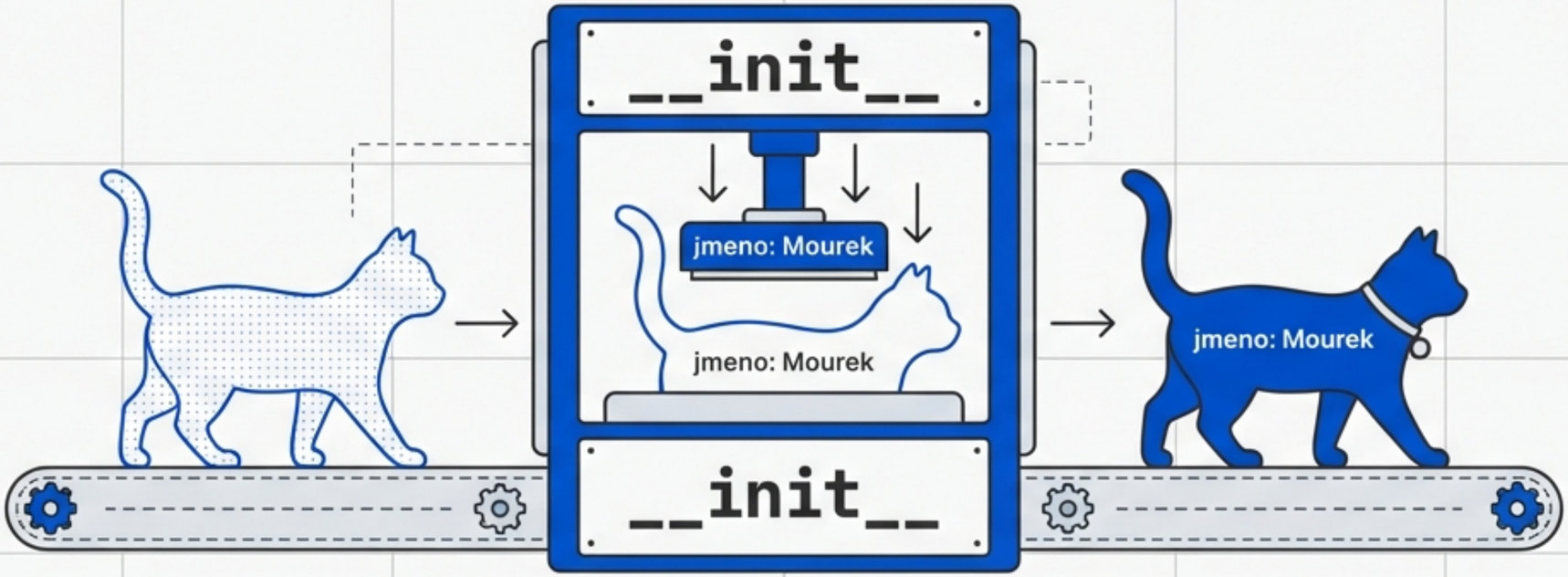


Otázka: Jak metoda ví, *které* koťátko má mňoukat?

Odpověď: **`self` je odkaz na konkrétní objekt.**

`mourek.zamnoukej()` (Python překládá na) → `Kotátko.zamnoukej(mourek)`

Konstruktor __init__: Tovární nastavení



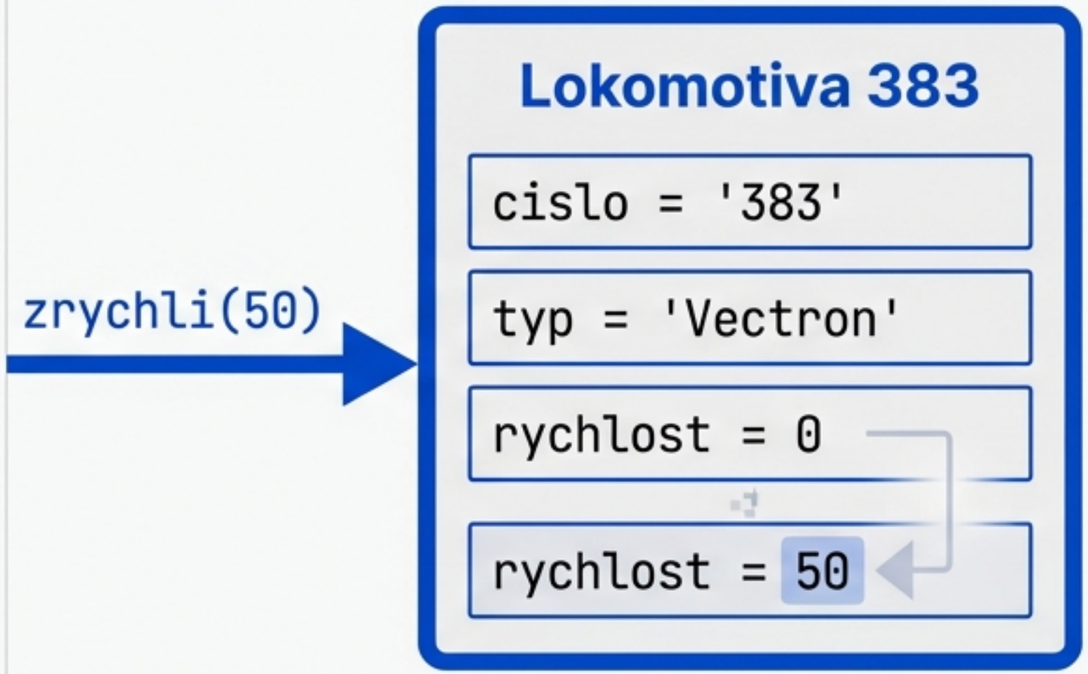

```
class Kotatko:
    def __init__(self, jmeno):
        self.jmeno = jmeno  # Uložení do paměti
```

Metoda, která se spustí automaticky. Zaručuje, že objekt je připraven k použití ihned po vytvoření.

Vše pohromadě: Třída Lokomotiva

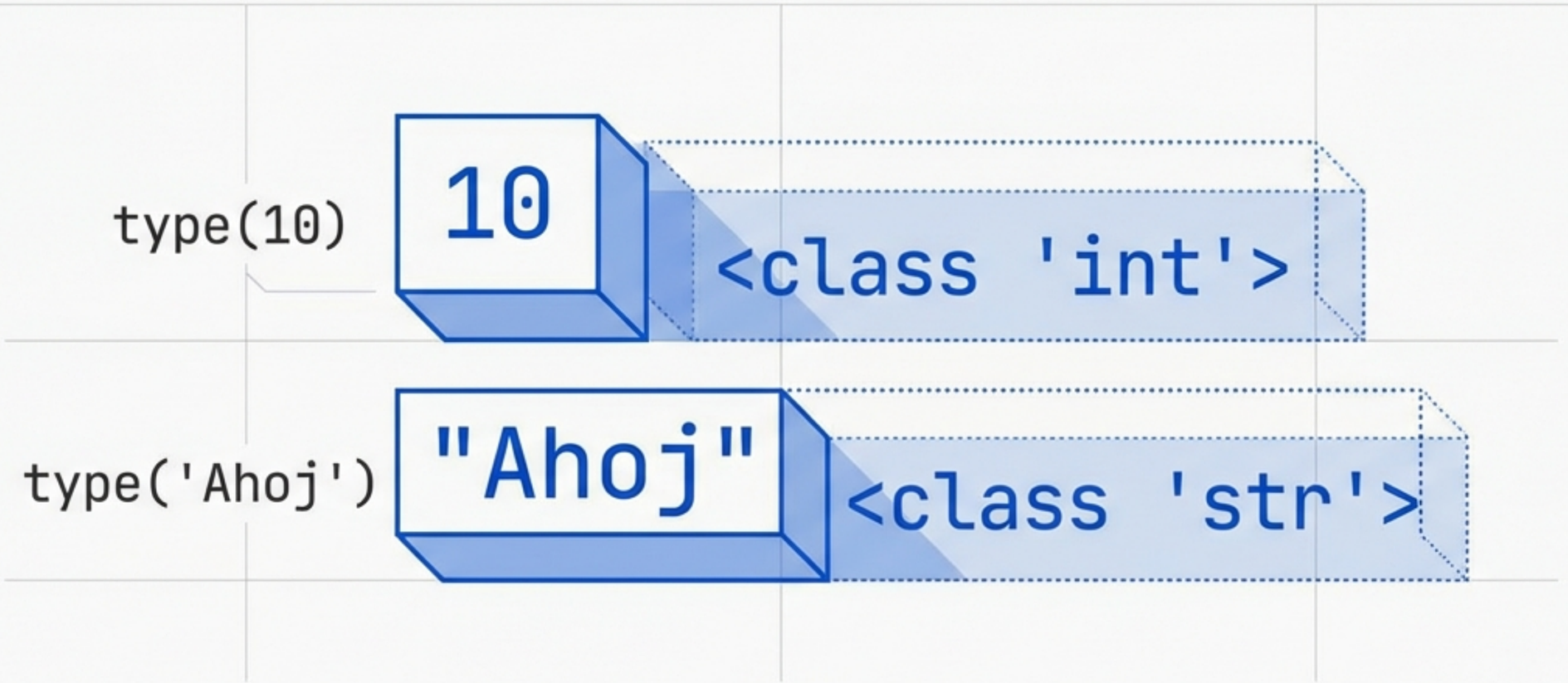
```
class Lokomotiva:
    def __init__(self, cislo, typ):
        self.cislo = cislo
        self.typ = typ
        self.rychlost = 0

    def zrychli(self, o_kolik):
        self.rychlost += o_kolik
        print(f'Mašina {self.cislo} jede
            {self.rychlost} km/h.')

stroj1 = Lokomotiva('383', 'Vectron')
stroj1.zrychli(50)
```

Zapouzdření (Encapsulation)



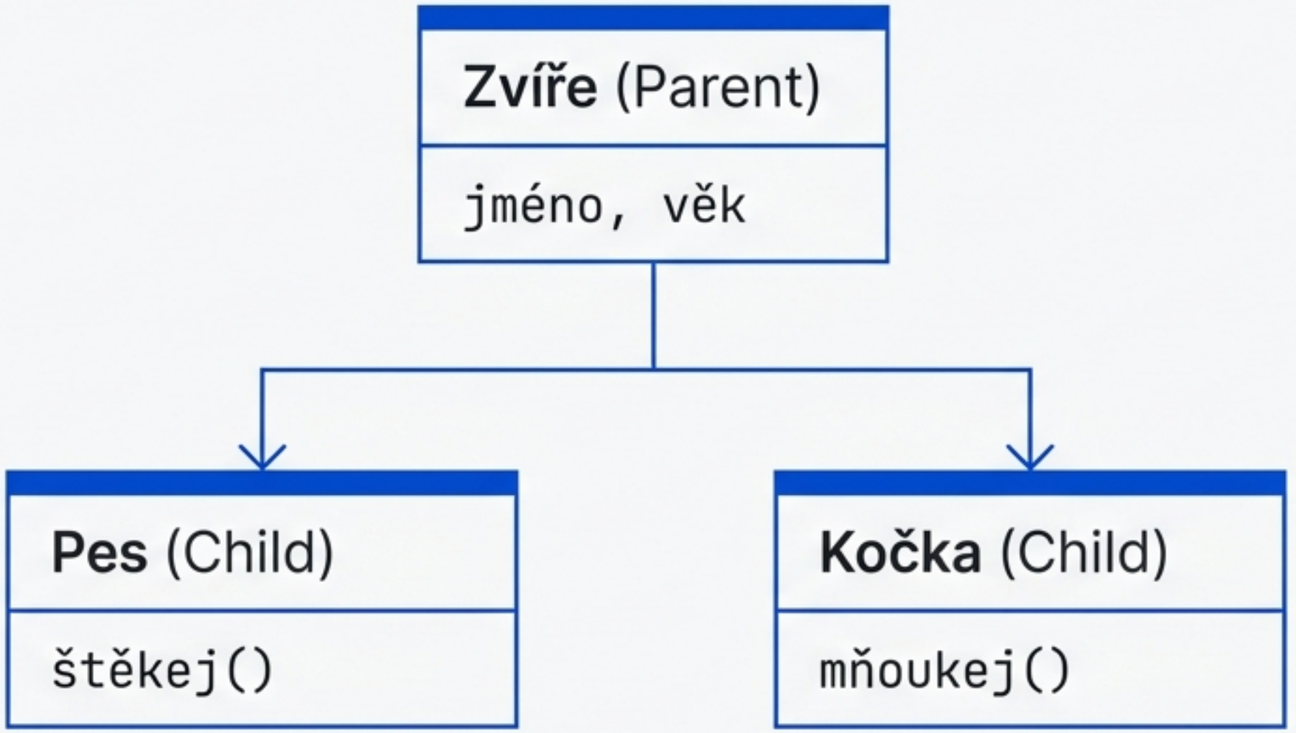
Nikde nepíšeme `rychlost = rychlost + 50`. Říkáme stroji: `Zrychli!` a on si svá data upraví sám.

Všechno v Pythonu je objekt



OOP není exotické. Používáte ho od začátku. Funkce jako `str()` nebo `int()` jsou ve skutečnosti konstruktory tříd.

Dědičnost: Princip specializace

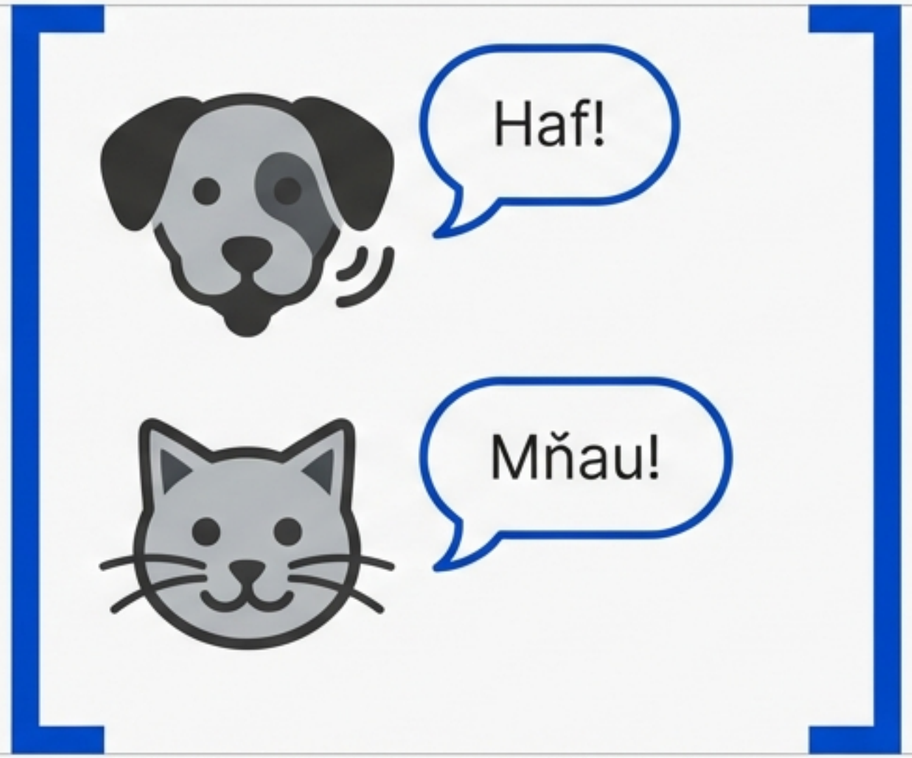



```
class Pes(Zvíře):
    def stekej(self):
        print('Haf!')
```

Pes má automaticky jméno i věk od Zvířete. Píšeme jen to, co je unikátní.

Polymorfismus:
Stejný příkaz, jiná akce

ozvi_se()




```
zvirata = [pes, kocka]
for zvire in zvirata:
    zvire.ozvi_se()
```

Nemusíme zjišťovat typ objektu. Zavoláme metodu a každý objekt ví, jak reagovat.

Atributy třídy vs. Atributy instance

Instance

self.jmeno
(Unikátní pro každého)

Class

Clovek.populace
(Sdílené všemi)

```
class Clovek:
    populace = 0  # Třídní atribut

    def __init__(self, jmeno):
        self.jmeno = jmeno
        Clovek.populace += 1
```

OOP Tahák (Cheat Sheet)

Třída (Class)

Šablona / Výkres

```
class Pes:
```

Objekt (Instance)

Konkrétní existence

```
alik = Pes()
```

Atribut

Data objektu

```
alik.jmeno
```

Metoda

Chování objektu

```
alik.stekej()
```

__init__

Konstruktor (Startovní nastavení)

```
def __init__(self):
```

self

Odkaz na konkrétní instanci

```
self.data
```

Dědičnost: `class Pes(Zvire):`