Objektovo orientované programovanie - *pokročilí*

Učiteľ:
Ing. Jozef Wagner PhD.

Učebnica:
https://oop.wagjo.com/

OPGP Pokročilí 10 (2 týždne)

1. Súkromné atribúty
2. Vymazanie atribútov a metód
3. Špeciálne metódy
4. Variabilný počet argumentov
5. Práca s formátom JSON

Súkromné atribúty

Riešené pomocou dohody a konvencie

Názov začínajúci s jedným podčiarníkom - interný atribút alebo metóda a nemá byť používaný mimo triedy.

```
class Osoba:
    def __init__(self, meno, vek):
        self._vek = vek        # interný podľa dohody
        self.meno = meno

    def vek(self):
        return self._vek

o = Osoba("Eva", 25)
print(o._vek)  # Dá sa k nemu dostať, ale nemalo by sa
```

Súkromné atribúty

názvy začínajúce s dvoma podčiarníkmi sa používajú pre súkromné atribúty a metódy.

Python vykoná špeciálne premenovanie (anglicky **name mangling**)

```
class Osoba:
    def __init__(self, meno, vek):
        self.__vek = vek    # súkromný s name manglingom
        self.meno = meno

    def zobraz(self):
        print(self.__vek)

o = Osoba("Eva", 25)
o.zobraz()          # funguje
# print(o.__vek)    # AttributeError
```

Vymazanie atribútov a metód

Pomocou príkazu del sa dajú vymazať atribúty, metódy alebo objekty.

```
print(p.vek)    # 30
del p.vek       # vymazanie atribútu vek

print(Osoba.druh)     # Clovek
del Osoba.druh        # vymazanie triedneho atribútu druh

p.privitanie()        # Ahoj, ja som Fero
del Osoba.privitanie # Vymazanie metódy privitanie
```

Vymazanie atribútov a metód

Využitie vymazania atribútov a metód

- Uvoľnenie pamäti
- Dynamická zmena správania triedy
- Pokročilé techniky zapuzdrenia
- Odstránenie testovacieho kódu a dát
- Metaprogramovanie

Vymazanie objektu

```
p = Osoba("Fero", 30)
del p;
```

Objekt sa uvoľní z pamäti ak na neho neukazuje žiadna premenná

Tesne pred vymazaním sa zavolá špeciálna metóda `__del__`

Metóda, ktorá sa volá tesne pred vymazaním sa volá **deštruktor**

```
class Demo:
    def __del__(self):
        print("Deštruktor sa zavolal!")

d = Demo()
del d  # alebo keď skončí program
```

Dunder metóda `__repr__`

Ako `__str__`, ale vráti technicky presnú reprezentáciu objektu, ideálne tak, aby sa z nej dal spätne vytvoriť rovnaký objekt.

Metóda `__repr__` sa zavolá pri volaní funkcie `repr`. Spätné vytvorenie objektu z textu je možné pomocou funkcie `eval`

```
class Osoba:
    def __repr__(self):
        return f"Osoba(meno={self.meno!r}, vek={self.vek!r})"

o = Osoba("Eva", 30)
print(repr(o)) # Osoba(meno='Eva', vek=30)

eval(repr(o))  # vytvorenie objektu z textovej reprezentácie
```

Dunder metóda `__eq__`

`__eq__` sa zavolá pri porovnaní objektov pomocou `==`

Ak ju nemáme, Python bude porovnávať podľa identity.

Ak porovnať nevieme, napr. argument je iného typu, vrátime hodnotu `NotImplemented`. Python následne skúsi porovnať iným spôsobom.

```
def __eq__(self, other):
    if not isinstance(other, Osoba):
        return NotImplemented
    return self.meno == other.meno and self.vek == other.vek
```

Aritmetické a relačné špeciálne metódy:
`__add__`, `__sub__`, `__mul__`, `__lt__`, `__gt__`, ...

Variabilný počet argumentov

Premenlivý počet pozičných argumentov - `*args` - argumenty budú v `tuple`

```
def sucet(*args):
    print(args)  # tuple všetkých argumentov
    return sum(args)

print(sucet(1, 2, 3))  # → 6
print(sucet(10, 20))   # → 30
```

Variabilný počet argumentov

Premenlivý počet kľúčových argumentov - `**kwargs` - argumenty budú v `dict`

```
def pozdrav(**kwargs):
    for meno, vek in kwargs.items():
        print(f"{meno} má {vek} rokov")


pozdrav(Jozef=25, Anna=30)
# Output:
# Jozef má 25 rokov
# Anna má 30 rokov
```

Rozbalenie pri volaní funkcie

Do argumentov funkcií môžeme rozbaliť hodnoty so zoznamu/tuple alebo aj celého slovníka (dict).

Pre rozbaľovanie použijeme rovnaký symbol ako pri deklarácií varargs v metóde (*, **)

```
def add(x, y, z):
    return x + y + z

nums = [1, 2, 3]
print(add(*nums))  # rozbali tuple/list → 6

data = {"x": 10, "y": 20, "z": 5}
print(add(**data))  # rozbali dict → 35
```

JSON (JavaScript Object Notation)

JSON je jednoduchý formát pre štruktúrované dáta, čitateľný pre ľudí a jednoducho spracovateľný počítačom.

Populárny dátový formát hojne používaný vo webových službách a aplikáciách.

Podporuje **čísla**, **reťazce, boolean** a 2 štruktúry: **pole** (ako list) a **objekt** (dict)

```
{
  "meno": "Jozef",
  "vek": 25,
  "student": true,
  "znamky": [1, 2, 1, 3]
}
```

JSON (JavaScript Object Notation)

Metóda `json.loads` načíta JSON z reťazca, `json.load` so súboru

Pre zápis dát do JSON sa používajú metódy `json.dump` a `json.dumps`

```
import json
data = '{"meno": "Jozef", "vek": 25, "student": true}'
# Konverzia textu na Python objekt (dict, list, atď.)
obj = json.loads(data)

print(obj)
print(obj["meno"])  # → Jozef
```

Načítanie objektu z JSON dát

```
class Osoba:
    def __init__(self, meno, vek, student):
        self.meno = meno
        self.vek = vek
        self.student = student

    @classmethod
    def from_json(cls, json_string):
        data = json.loads(json_string)
        return cls(**data)

osoba = Osoba.from_json('{"meno": "Jozef", "vek": 25, "student": true}')
```