(PTIA1201) Elemi programozás

Dr. Facskó Gábor, PhD
tudományos főmunkatárs
facskog@gamma.ttk.pte.hu

Pécsi Tudományegyetem, Természettudományi Kar, Matematikai és Informatikai Intézet, 7624 Pécs, Ifjúság utja 6.
Wigner Fizikai Kutatóközpont, Űrfizikai és Űrtechnikai Osztály, 1121 Budapest, Konkoly-Thege Miklós út 29-33.
`https://facsko.ttk.pte.hu`

2024. október 18.

Programozási tételek, egyszerű feladatok

- ~~Maximum/minimum kiválasztás~~
- ~~Tömb elemeinek összege~~
- ~~Buborék rendezés~~
- ~~Másodfokú egyenlet megoldása~~
- Elem kiválasztása
- Megszámlálás
- ~~Rekurzió~~
  - ~~Faktoriális~~
  - ~~Fibonacci-sorozat~~

Input, Állományok kezelése

- ~~Input: a = input()~~
- ~~Open, read~~
- Lorem ipsumos példaprogram

Osztályok, objektum orientált programozás I

- Az osztály absztrakt fogalom, konkrét életbeli objektumok, objektumcsoportok formai leírása. Osztályokkal olyan objektumokat formalizálhatunk, amelyek azonos tulajdonságokkal és viselkedéssel rendelkeznek.
  class OsztalyNeve(object):
  osztály törzse...
- Közvetlenül az osztály neve után, zárójelek között megadhatjuk, hogy melyik osztályból öröklődünk.
- A Python támogatja a többszörös öröklődést, így egy osztály akár egyszerre több másik osztályból is öröklődhet.
- Object ősosztály - nem kell már kiírni
- Példa: A Szuperhos osztály elkezdése

```
class Szuperhos:
pass
```

Osztályok, objektum orientált programozás II

- Adattagok, metódusok (tagfüggvényeket). A . (pont) operátorral érhető el
- Pythonban az adattagokat a konstruktoron belül hozzuk létre és inicializáljuk őket.
- A metódusok függvények az osztályon belül, így a def kulcsszóval hozzuk létre őket. Minden metódusnak van egy kötelező első paramétere, amit expliciten ki kell írni minden metódus fejlécében. Ezt általában self-nek nevezzük el, és ezzel magára az objektumra tudunk hivatkozni. Az aktuális objektumra hivatkozó azonosítót nem kötelező self-nek elnevezni, a név tetszőleges is lehet.
  Példa: Metódus létrehozása az osztályon belül

```
class OsztalyNeve:
    def metodusNeve(self, param):   # a self után jönnek a "tényleges"
        print("Ez egy metódus a következő paraméterrel:", param)
```

Osztályok, objektum orientált programozás III

- A konstruktor a konkrét objektumpéldányok létrehozásakor lefutó, speciális metódus. Pythonban az __init__ metódus a konstruktornak, ezt használjuk az adattagok létrehozására és inicializálására. Szintaxisa (a szögletes zárójelek közötti részek elhagyhatók):

```
class OsztalyNeve:
    def __init__(self, [param1, param2, ...]):
    konstruktor utasítások... (pl. adattagok inicializálása)
```

Példa: A szuperhősökről eltároljuk a nevüket és a szupererejüket. Hozzunk létre egy olyan konstruktort a Szuperhos osztályban, amely paraméterül kapja a nevet és a szupererőt, és ezekkel az értékekkel inicializálja az adattagokat!

```
class Szuperhos:
    def __init__(self, nev, szuperero):
        self.nev = nev                    # adattagok létrehozása és inic
        self.szuperero = szuperero
```

Osztályok, objektum orientált programozás IV

- Pythonban nincs function overload. Ha azt szeretnénk elérni, hogy egy metódust többféle, eltérő paraméterezéssel is tudjunk használni, akkor használjuk a default függvényparamétereket.
  Példa: Írjuk át a Szuperhos osztály konstruktorát úgy, hogy a szupererő paraméter értékét ne legyen kötelező megadni, alapértéke legyen 50!

```
class Szuperhos:
    def __init__(self, nev, szuperero=50):
        self.nev = nev
        self.szuperero = szuperero
```

- Objektumok létrehozása: példányosítás:
  `objektumNeve = OsztalyNeve([param1, param2, ...])`
  Példa: Példányosítsuk a Szuperhos osztályunkat!

Osztályok, objektum orientált programozás V

```
class Szuperhos:
    def __init__(self, nev, szuperero=50):
        self.nev = nev
        self.szuperero = szuperero
        print("-- Szuperhős létrehozva. --")  # a szemléletesség kedvé
hos1 = Szuperhos("Thor", 70)
```

- Láthatóságok, getterek, setterek, property-k Javában az adattagok és metódusok elérhetőségét különböző láthatósági módosítószavakkal tudtuk megadni (public, protected, private, "package private" láthatóságok).
- Pythonban nincsenek a láthatóság szabályozására szolgáló módosítószavak.

Osztályok, objektum orientált programozás VI

- Konvenció alapján az adattag neve előtti egyszeres alulvonás azt jelzi, hogy az adattag nem publikus használatra van szánva ("private adattag"). Viszont ettől az adattag kívülről továbbra is elérhető lesz!
  Példa: Jelezzük, hogy a Szuperhos osztály adattagjait nem publikus használatra szánjuk!

```
class Szuperhos:
    def __init__(self, nev, szuperero=50):
        self._nev = nev
        self._szuperero = szuperero
```

- Pythonban is készíthetünk az adattagokhoz hagyományos getter, illetve setter metódusokat.

Osztályok, objektum orientált programozás VII

- getter az adattagok értékének lekérdezésére, míg a setter az adattagok értékének beállítására szolgáló metódus. Ezeket nem publikus adattagok esetén használjuk. Példa: Írjunk hagyományos gettert és settert a Szuperhos osztály _szuperero adattagjához!

```
class Szuperhos:
    def __init__(self, nev, szuperero=50):
        self._nev = nev
        self._szuperero = szuperero

    def get_szuperero(self):            # getter metódus
        return self._szuperero

    def set_szuperero(self, ertek):     # setter metódus
        self._szuperero = ertek
```

Osztályok, objektum orientált programozás VIII

```
# === példányosítás ===
hos1 = Szuperhos("Thor", 70)
hos1.set_szuperero(100)                # setter hívás
print(hos1.get_szuperero())            # getter hívás
```

- A Pythonban property-ket szoktunk használni getterek és setterek megvalósítására. A get property szintaxisa:

```
@property\\
def adattag1(self):
return self._adattag1
\end{verbatim
A set property szintaxisa:\\
\begin{verbatim}
@adattag1.setter
def adattag1(self, ertek):
    self._adattag1 = ertek
```

Osztályok, objektum orientált programozás IX

```
\end{verbatim
Példa: Cseréljük le a Szuperhos osztályban a \_szuperero adattaghoz kés
\begin{verbatim}
class Szuperhos:
    def __init__(self, nev, szuperero=50):
        self._nev = nev
        self._szuperero = szuperero
    @property
    def szuperero(self):              # get property
        return self._szuperero
    @szuperero.setter
    def szuperero(self, ertek):       # set property
        self._szuperero = ertek
hos1 = Szuperhos("Thor", 70)
hos1.szuperero = 100                    # set property hívás
```

Osztályok, objektum orientált programozás X

```
print(hos1.szuperero)                    # get property hívás
```

- Fontos, hogy a property és az adattag neve mindig eltérő legyen. A fenti példában a get és set property-k neve szuperero (alulvonás nélkül), az adattag neve pedig ettől eltérő módon _szuperero (alulvonással).
- Feladat: A fenti mintájára készítsünk get és set property-t a _nev adattaghoz is!
- Objektumok kiíratása. Printtel: ronda.
- Átdefiniálás, átdifináljuk a __str__ metódust.

```
class OsztalyNeve:
    # ...
    def __str__(self):
    return "Ez a szöveg fog megjelenni az objektum kiíratásakor."
```

- Példa: Írjuk meg a Szuperhos osztályban a szöveggé alakítást megvalósító metódust! A metódus írja ki az adattagok értékét!

Osztályok, objektum orientált programozás XI

```
class Szuperhos:
    def __init__(self, nev, szuperero=50):
        self._nev = nev
        self._szuperero = szuperero
    #...

    def __str__(self):
        return self._nev + " egy szuperhős, akinek szuperereje " + str(
```

```
hos1 = Szuperhos("Thor", 70)
print(hos1)
Thor egy szuperhős, akinek szuperereje 70
```

Osztályok, objektum orientált programozás XII

- Operator overloading: aperator overloading segítségével kiterjeszthetjük a megszokott operátoraink működését. Például a + (plusz) operátort Pythonban használhatjuk két egész szám összeadására, illetve két string összefűzésére is. Ez azért lehetséges, mert a + operátor ki van terjesztve az int és az str osztályokban egyaránt. Amikor egy operátor különböző osztályok példányaira használva másként viselkedik, operator overloading-nak nevezzük.
- Pythonban a saját osztályainkban is lehetőségünk van bizonyos operátorok működését felüldefiniálnunk, ha felülírjuk a nekik megfelelő metódus működését.
- Néhány speciális metódus, és az operátorok, amelyeket felüldefiniálhatunk vele:
  Operator overload függvény A függvényt meghívó kifejezés

Osztályok, objektum orientált programozás XIII

```
__eq__ (egyenlőség) obj1 == obj2
__ne__ (nem egyenlőség) obj1 != obj2
__add__ (összeadás) obj1 + obj2
__sub__ (kivonás) obj1 - obj2
__iadd__ (megnövelés) obj1 += obj2
__isub__ (csökkentés) obj1 -= obj2
__lt__ (kisebb, mint) obj1 < obj2
__gt__ (nagyobb, mint) obj1 > obj2
__le__ (kisebb vagy egyenlő) obj1 <= obj2
__ge__ (nagyobb vagy egyenlő) obj1 >= obj2
```

- Példa: Definiáljuk felül az $==$ és $+$ operátorok működését a Szuperhos osztályban!

Osztályok, objektum orientált programozás XIV

```
class Szuperhos:
    def __init__(self, nev, szuperero=50):
        self._nev = nev
        self._szuperero = szuperero
    def __str__(self):
        return self._nev + " egy szuperhős, akinek szuperereje " + str

    # két szuperhős akkor lesz egyenlő, ha a nevük és a szupererejük me

    def __eq__(self, masik_hos):
        return self._nev == masik_hos._nev and self._szuperero == masik

    # két szuperhős összeadása során a szupererejük összeadódik

    def __add__(self, masik_hos):
```

Osztályok, objektum orientált programozás XV

```
            uj_szuperero = self._szuperero + masik_hos._szuperero
            uj_szuperhos = Szuperhos("Megahős", uj_szuperero)
            return uj_szuperhos

#=== tesztelés ===

hos1 = Szuperhos("Thor", 70)
hos2 = Szuperhos("Hulk", 80)
hos3 = Szuperhos("Hulk", 80)
hos4 = hos1 + hos2
print(hos2 == hos3)
print(hos4)
```

Osztályok, objektum orientált programozás XVI

- Típusellenőrzés: Mivel a Python nem fordított nyelv, így statikus típusellenőrzés nincs benne. Sajnos ezt csak dinamikusan, futásidőben tudjuk ellenőrizni, az isinstance(obj, type) függvénnyel. A függvény pontosan akkor ad vissza igazat, ha az obj objektum type típusú.
  Példa: Az __add__ metódus utasításait csak Szuperhos típusú paraméter esetén hajtsuk végre! Eltérő típus esetén írassunk ki hibaüzenetet!

```
class Szuperhos:
    def __init__(self, nev, szuperero=50):
        self._nev = nev
        self._szuperero = szuperero

    #...

    def __add__(self, masik_hos):
        if isinstance(masik_hos, Szuperhos):   # típusellenőrzés
```

Osztályok, objektum orientált programozás XVII

```
            uj_szuperero = self._szuperero + masik_hos._szuperero
            uj_szuperhos = Szuperhos("Megahős", uj_szuperero)
            return uj_szuperhos
        else:
        print("Egy szuperhőst csak egy másik szuperhőssel lehet összeac
```

- Az isinstance() függvényt beépített típusokra is használhatjuk.

```
print(isinstance(42, int))
print(isinstance(42, str))
```

Vége

Köszönöm a figyelmüket!