(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 25.

Programozási tételek, egyszerű feladatok

- ~~Maximum/minimum kiválasztás~~
- ~~Tömb elemeinek összege~~
- ~~Buborék rendezés~~
- KIválasztás
- Megszámlálás
- Keresés
- Eldöntés
- ~~Másodfokú egyenlet megoldása~~
- Elem kiválasztása
- Megszámlálás
- Lineáris egyenletrendszerek megoldása Gauss-eliminációval

Osztályok, objektumok, metódusok, konstruktor I

- Az osztály absztrakt fogalom, valós 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.
- Az Object ősosztály nem kell már kiírni
- Példa: A Szuperhos osztály elkezdése

```
class Szuperhos:
  pass
```

Osztályok, objektumok, metódusok, konstruktor II

- Az osztály adattagokat és metódusok (tagfüggvényeket) tartalmaz. Ezek a . (pont) operátorral érhetőek el
- Pythonban az adattagokat a konstruktoron belül hozzuk létre és inicializáljuk őket. (A konstruktor az a metódus, ami létrehozza és inicializálja az adott objektumot.)
- A metódusok függvények az osztályon belül, így a def kulcsszóval hozzuk létre őket. Minden metódusnak van egy kötelező első paramétere, amit 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 az igazi paramét
        print("Ez egy metódus a következö paraméterrel:", param)
```

Osztályok, objektumok, metódusok, konstruktor III

- A konstruktor a konkrét objektumpéldányok létrehozásakor lefutó, speciális metódus. Pythonban az __init__ metódus a konstruktornak, ezt használjuk az adattagok létrehozására és inicializálására:

```
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 inicializálása
        self.szuperero = szuperero
```

Osztályok, objektumok, metódusok, konstruktor 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
```

Osztályok, objektumok, metódusok, konstruktor V

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

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

Láthatóságok, getterek, setterek, property-k I

- Javában, C++-ban és C#-ban az adattagok és metódusok elérhetőségét különböző láthatósági módosítószavakkal tudtuk megadni (public, protected, private láthatóságok).
- Pythonban nincsenek a láthatóság szabályozására szolgáló módosítószavak.
- Az adattag neve előtti egyszeres alulvonás azt jelzi, hogy az adattag nem publikus, de ettől még az adattag 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
```

Láthatóságok, getterek, setterek, property-k II

- Pythonban is készíthetünk az adattagokhoz hagyományos getter, illetve setter metódusokat.
- getter az adattagok értékének lekérdezésére, míg a setter az adattagok értékének beállítására szolgáló metódus. Ezeket nem publikus adattagok esetén használjuk.

Láthatóságok, getterek, setterek, property-k III

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

Láthatóságok, getterek, setterek, property-k IV

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

```
@property
def adattag1(self):
   return self._adattag1
```

- A set property szintaxisa:

```
@adattag1.setter
def adattag1(self, ertek):
    self._adattag1 = ertek
```

Láthatóságok, getterek, setterek, property-k V

- Példa: Cseréljük le a Szuperhos osztályban a _szuperero adattaghoz készített hagyományos gettert és settert property-kre!

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

Láthatóságok, getterek, setterek, property-k VI

- 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 I

- Printtel ronda. Átdefináljuk a __str__ metódust.

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

Objektumok kiíratása II

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

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

    #...

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

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

Operator overloading I

- Operator 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.

Operator overloading II

- Néhány speciális metódus, és az operátorok, amelyeket felüldefiniálhatunk vele:

```
  Operator overload függvény A függvényt meghívó kifejezés
__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 egyenlö) obj1 <= obj2
__ge__ (nagyobb vagy egyenlö) obj1 >= obj2
```

Operator overloading III

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

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

    # két szuperhös akkor lesz egyenlö, ha a nevük és a szuperérejük megegyezik
    def __eq__(self, masik_hos):
        return self._nev == masik_hos._nev and self._szuperero == masik_hos._szuperero

    # két szuperhös összeadása során a szuperérejük összeadódik
    def __add__(self, masik_hos):
        uj_szuperero = self._szuperero + masik_hos._szuperero
        uj_szuperhos = Szuperhos("Megahös", 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)
```

Típusellenőrzés I

- Nincs Pythonban statikus típusellenőrzés. Ezt csak dinamikusan, futásidőben lehet ellenőrizni, az isinstance(obj, type) függvénnyel.
- 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):   # típusellenörzés
            uj_szuperero = self._szuperero + masik_hos._szuperero
            uj_szuperhos = Szuperhos("Megahös", uj_szuperero)
            return uj_szuperhos
        else:
        print("Egy szuperhöst csak egy másik szuperhössel lehet összeadni.")
```

- Az isinstance() függvényt beépített típusokra is használhatjuk.

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

Vége

Köszönöm a figyelmüket!