(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. november 8.

(Érettségiből) kimaradó anyagrészek I

- Algebrai tört fogalma és alkalmazása. Lineáris törtfüggvények ábrázolása, jellemzése.
- **Abszolútt értéket tartalmazó egyenletek megoldása.**
- Másodfokú egyenleteknél a gyökök és együtthatók összefüggései. Másodfokú egyenletrendszer megoldása.
- Összefüggés két pozitív szám számtani és mértani közepe között.
- **Négyzetgyök alatt csak olyan elsőfokú polinomok, amelyek főegyütthatója 1, azaz $\sqrt{ax+b} = cx+d$ helyett a $\sqrt{x+c} = ax+b$ megoldása elegendő. (Eddig az ax+b alakú elsőfokú polinomok négyzetgyökét is vizsgálták.)**
- A függvény transzformációk közül az $f(cx)$ ábrázolása.
- Magasságtétel, befogótétel a derékszögű háromszögben.
- **Szög ívmértéke.**
- Logarimusfüggvény, logaritmus azonosságai, logaritmusos egyenletek.

(Érettségiből) kimaradó anyagrészek II

- **Függvény inverze.**
- **Az egyenes egyenletének normálvektoros és irányvektoros alakja, kör és egyenes kölcsönös helyzete a koordinátageometriában.**
- **Két vektor skaláris szorzata.**
- A valós számok halmazán értelmezett triginometrikus függvények értelmezése, ábrázolása és trigonometrikus egyenletek megoldása.

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

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!