UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM
Department of Software Engineering

Szkriptnyelvek - 4. gyakorlat

Cservenák Bence

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM
Department of Software Engineering

- A dictionary egy kulcs-érték párokból álló leképezés
  - mint Javában a map vagy PHP-ban az asszociatív tömb
- Létrehozás (példa):

```
ures = dict()          # üres dictionary létrehozása
jegyek = { "ASD123": 3, "FOOB4R": 5, "NEP4LF": 3 }
```

  - a példában a kulcs a Neptun kód, az érték az érdemjegy
- `len(d)`: visszaadja a `d` dictionary hosszát

```
jegyek = { "ASD123": 3, "FOOB4R": 5, "NEP4LF": 3 }
print(len(jegyek))
```

```
3
```

- Egy dictionary adott kulcshoz tartozó értéke lekérhető…
  - …tömbindexeléssel: `dictionary_neve[kulcs]`
    - ha nincs ilyen kulcs, hibát kapunk(!)
  - …`get()` metódussal: `dictionary_neve.get(kulcs)`
    - ha nincs ilyen kulcs, **`None`** értékkel tér vissza

```
jegyek = { "ASD123": 3, "FOOB4R": 5, "NEP4LF": 3 }

print(jegyek["FOOB4R"])
print(jegyek.get("FOOB4R"))

# print(jegyek["KEKW55"])       # hiba!
print(jegyek.get("KEKW55"))
```

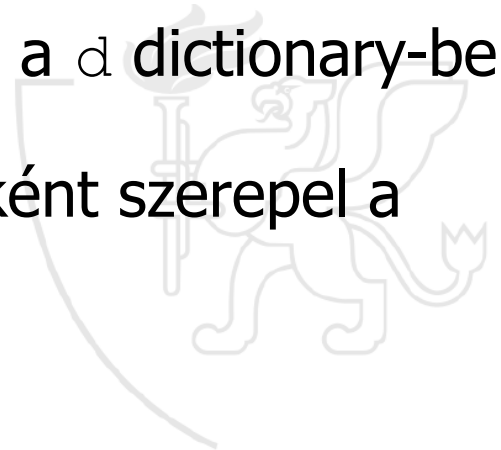
```
5
5


None
```

- Egy létező dictionary-hez új elemet adhatunk hozzá, ha az új elem kulcsára megadjuk az értéket

```
d = dict()
d[kulcs] = érték
```

- `del d[k]`: törli a `d` dictionary-ből a `k` kulcsú elempárt
- `elem in d`: igazat ad vissza, ha `elem` kulcsként szerepel a `d` dictionary-ben
- `elem in d.values()`: igazat ad vissza, ha `elem` értékként szerepel a `d` dictionary-ben



- Példa:

```
jegyek = { "ASD123": 3, "FOOB4R": 5, "NEP4LF": 3 }

jegyek["KEKW55"] = 4              # új kulcs-érték pár hozzáadása
del jegyek["ASD123"]              # adott kulcsú elem törlése
print(jegyek)

if "ASD123" in jegyek:            # adott kulcs keresése
   print("Az ASD123 Neptun kódú hallgató jegye: " + str(jegyek["ASD123"]))
else:
   print("Nincs ilyen hallgató!")

if 5 in jegyek.values():          # adott érték keresése
   print("Lett ötös ^^")
```

```
{'FOOB4R': 5, 'NEP4LF': 3, 'KEKW55': 4}
Nincs ilyen hallgató!
Lett ötös ^^
```

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM
Department of Software Engineering

- Dictionary bejárása
  - kulcsok bejárása:

```
for neptun in jegyek:
    print(neptun)
```

  - értékek bejárása:

```
for jegy in jegyek.values():
    print(jegy)
```

  - kulcsok és értékek együttes bejárása:

```
for neptun, jegy in jegyek.items():
    print(neptun, "érdemjegye:", jegy)
```

- A Python támogatja az objektumorientált paradigmát
- Osztály létrehozása:

```
class OsztalyNeve(object):
    osztály törzse…
```

  - zárójelek között megadjuk, hogy melyik osztály(ok)ból öröklődünk
  - a gyakorlaton nem tárgyaljuk az öröklődést, mindig az `object`-ből származtatunk
- Az osztályok adattagokat és metódusokat (tagfüggvényeket) tartalmazhatnak
  - ezek elérése a `.` (pont) operátorral történik
  - minden metódus első paramétere a **`self`**, amit expliciten ki kell írnunk a metódus fejlécében
    - ezzel magára az objektumra hivatkozhatunk (mint Javában a `this`)

- A konstruktor az osztály példányosítása során lefutó metódus
  - Pythonban az `__init__` metódus tekinthető konstruktornak
  - itt szoktuk létrehozni és inicializálni az adattagokat
  - Pythonban egy adattagra vagy metódusra való hivatkozásnál is mindig expliciten ki kell írni a **`self`**-et! (pl. **`self`**`.adattag`, **`self`**`.metodus()`)

```
class OsztalyNeve(object):

    def __init__(self, param1, param2, …):    # konstruktor
        self.adattag1 = param1   # adattagok inicializálása a paraméterekkel
        self.adattag2 = param2
        ...
```

- A példányosítás során az osztályból objektumot hozunk létre
  - átadjuk a paramétereket a konstruktornak (az első, `self` paramétert nem)

```
class OsztalyNeve(object):

    def __init__(self, param1, param2, …):
        self.adattag1 = param1
        self.adattag2 = param2
        ...

# osztály példányosítása (objektum létrehozása)

objektumNeve = OsztalyNeve(p1, p2, …)
```

- Pythonban **nincsenek** láthatósági módosítószók
  - Javában: `private`, `protected`, `public`, "package private" láthatóságok
- Konvenció alapján az adattag neve előtti egyszeres alulvonás (_) jelzi azt, hogy az adattag nem publikus használatra van szánva ("private")
  - viszont ettől még kívülről ugyanúgy elérhető lesz

```
class OsztalyNeve(object):

   def __init__(self, param1, param2, …):
      self._adattag1 = param1  # nem publikus használatra szánt adattagok
      self._adattag2 = param2
      ...
```

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM
Department of Software Engineering

- Pythonban is írhatunk az adattagokhoz gettereket és settereket
- Getter: "private" adattagok lekérdezésére szolgáló metódus

```
def getAdattag(self):
    return self._adattag
```

- Setter: "private" adattagok értékének beállítására szolgáló metódus

```
def setAdattag(self, ertek):
    self._adattag = ertek
```

- Az előző dián látható szintaxis Pythonban ritkán használatos, helyettük property-ket szokás használni getterek és setterek megvalósítására
  - getterek property-je:

```
@property
def adattag(self):
    return self._adattag
```

  - setterek property-je:

```
@adattag.setter
def adattag(self, ertek):
    self._adattag = ertek
```

  - ezeknek köszönhetően úgy tűnik, minta publikus adattagokkal dolgoznánk

Figyelem

A property és az adattag neve mindig eltérő legyen, különben végtelen rekurzióba futunk!

- Ha a `print()` utasítással kiíratjuk az objektumot, akkor valami ehhez hasonlót kapunk:

```
<__main__.OsztalyNeve object at 0x012B08D0>
```

- Ezt lehetőségünk van "szebbé tenni" az `__str__` metódus felüldefiniálásával
  - az objektum szöveggé alakítását valósítja meg (mint Javában a `toString()`)
  - egy stringgel tér vissza, ezt írjuk felül

```
class OsztalyNeve(object):

    ...

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

- Pythonban lehetőségünk van bizonyos operátorok működését felüldefiniálni, ha felülírjuk a nekik megfelelő metódus működését (operator overloading)
- Néhány fontosabb metódus, amelyekkel operátorokat lehet felüldefiniálni:
  - `__eq__`: egyenlőség (`obj1 == obj2` hívja meg)
  - `__neq__`: nem egyenlőség (`obj1 != obj2` hívja meg)
  - `__add__`: összeadás (`obj1 + obj2` hívja meg)
  - …

- Pythonban lehetőségünk van típusellenőrzésre is
- `isinstance(obj, type)`: igazat ad vissza, ha az `obj` objektum `type` típusú

```
class OsztalyNeve(object):

    def __init__(self, param1, param2):
        self.adattag1 = param1
        self.adattag2 = param2

objektumNeve = OsztalyNeve("first", "second")
print(isinstance(objektumNeve, OsztalyNeve))     # True
```

  - beépített típusokra is működik

```
print(isinstance(42, int))      # True
print(isinstance(42, str))      # False
```

```
class Sutemeny(object):
    def __init__(self, nev, szeletek = 8):              # konstruktor
        self.nev = nev
        self._szeletek = szeletek                       # "private" adattag (konvenció)

    @property
    def szeletek(self):                                 # getter (property-s megvalósítás)
        return self._szeletek

    @szeletek.setter
    def szeletek(self, szeletek):                       # setter (property-s megvalósítás)
        self._szeletek = szeletek

    def __str__(self):                                  # szöveggé alakítás
        return "Ez egy " + str(self.szeletek) + " szeletes " + self.nev

    def __eq__(self, masik_suti):                       # == operátor felüldefiniálása
        if isinstance(masik_suti, Sutemeny):
            return self.nev == masik_suti.nev and self.szeletek == masik_suti.szeletek
        return False
```

```
    def __add__(self, masik_suti):                    # + operátor felüldefiniálása
        if isinstance(masik_suti, Sutemeny):
            uj_szeletek = self.szeletek + masik_suti.szeletek
            uj_suti = Sutemeny("szupersüti", uj_szeletek)
            return uj_suti
        else:
            print("HIBA: Nem sütemény típusú paraméter!")

# főprogram

suti1 = Sutemeny("krémes")                            # példányosítás
suti2 = Sutemeny("brownie", 16)
print(suti1.szeletek)                                 # getter hívás
suti2.szeletek = 12                                   # setter hívás
print(suti1)                                          # __str__ hívás
print(suti2)
suti3 = suti1 + suti2                                 # __add__ hívás
print(suti3)
suti4 = Sutemeny("szupersüti", 20)
print(suti3 == suti4)                                 # __eq__ hívás
```

- Az előző diákon szereplő program outputja:

```
8
Ez egy 8 szeletes krémes
Ez egy 12 szeletes brownie
Ez egy 20 szeletes szupersüti
True
```