A könyv szerkezete és használata

A könyv igyekszik egy új megközelítésben, a fontosabb koncepciók mentén tárgyalni a Python nyelvet. A könyv minden fejezete az adott koncepció felvezetésével kezd, majd végigveszi (mintegy referenciaszerűen) a nyelv jelöléseit.

- Aki most ismerkedik a nyelvvel és csak a programozási nyelv fontosabb koncepcióinak a megértése a célja, annak ajánlott a fejezetek elejére koncentrálni. A referenciarészt nyugodtan átugorhatja. Számára érdemes lehet még az A. függelék tanulmányozása.

- Aki most ismerkedik a nyelvvel azzal a céllal, hogy megtanuljon programozni, annak ajánljuk, hogy a fejezetek elejének olvasásakor próbálja is ki a példákat. A referenciarészt érdemes első alkalommal csak átfutni, majd később visszatérni rá részletes átolvasás céljából.

- Aki tapasztalt szoftverfejlesztő, annak a fejezetek elejét érdemes úgy olvasni, hogy arra figyel, hogy a Python nyelvben milyen új, más nyelvekben nem létező koncepciók vannak. Számára a fejezetek végén lévő referenciarészek szinte kötelező olvasmányok.

Az ábrák az UML jelölést[5] követik és a jelentésük rövid leírása az 1., a 3., a 4. és a 5. fejezetek végén találhatóak. A forráskódpéldákban a könyvre jellemző tördelési kihívások miatt néha eltérünk a Python kódolási szabványtól. Ezek leginkább rövid vagy rövidített nevekben és 40-50 karakter széles tördelésekben nyilvánulnak meg. A programkódpéldák koncepcionálisan önálló egységek, de feltételezik, hogy a könyvben korábban elhelyezkedő példák lefuttatásra kerültek, mert hivatkozhatnak az azokban szereplő definíciókra. A könyvben leírtak és az összeállított példák a könyv írásakor elérhető legfrissebb 3.8.2 Python verzió funkcióit veszik alapul. A példák nagy része működik a jelenleg széles körben elterjedt 3.6.x és 3.7.x verziókkal is. A példák futásának eredményeit a könyvben tudatosan nem közöljük. A könyv weblapján – `http://smartpython.axonmatics.com` – megtalálhatóak a könyvben ismertetett forráskódok, kiegészítések, visszajelzési lehetőségek, kapcsolódó tanfolyamok és további információk.

A könyvben kétfajta szövegdobozt fogunk használni. A ⚠ szimbólumokkal jelöltek fontos technikai részletek tartalmaznak. A ℹ szimbólumokkal jelöltek pedig érdekes plusz információkat hordoznak.

3. Az osztály – hogyan modellezzük a világot

„... »Think like an object.« Of course, this statement gets its real meaning by contrast with the typical approach to software development: »Think like a computer.«”

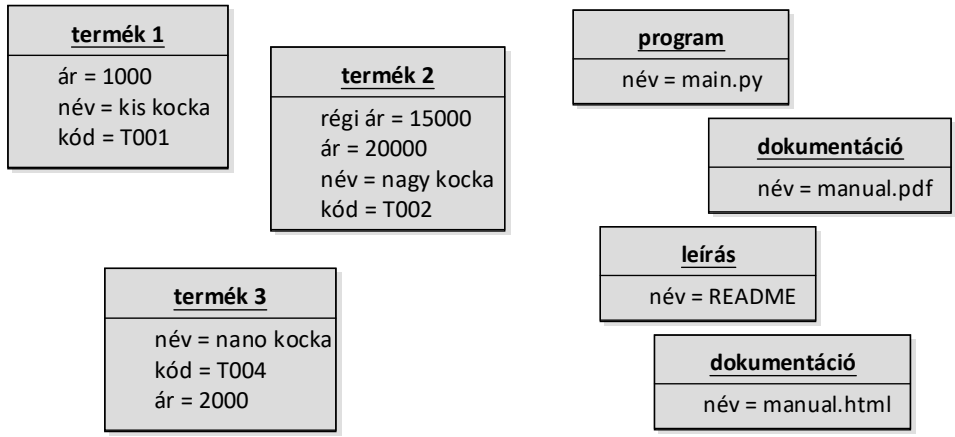
David West

3.1. Mi az objektumorientált programozás?

A programozási nyelvek tervezésénél az egyik fontos cél, hogy a programok szövegét a valóságban lévő fogalmakhoz nagyon hasonlóan tudjuk megfogalmazni. Az objektumorientált programozás vagy szemlélet erre tesz kísérletet azzal, hogy a program szövegében a valóságban létező objektumoknak megfeleltethető objektumokat hozhatunk létre. Az objektumok lehetnek például a valóságban létező termékek programban reprezentáló objektumok a 3.1a. ábra szerint; vagy lehetnek valamilyen elvontabb, technikai objektumok, mint például számítógépen elhelyezkedő fájlt reprezentáló objektumok a 3.1b. szerint.

Az objektumoknak példányváltozóik és metódusaik vannak. A példányváltozók az adott objektum változónevei. A metódusok pedig az adott objektumra jellemző viselkedést leíró függvények. Az előbbi példánál maradva a `Termek` objektumnak lehetnek olyan példányváltozói, mint: kód, név, ár, régi ár; valamint olyan viselkedése, hogy leértékelik a terméket.

> **i** A Python nyelv dokumentációjában előszeretettel használják az attribútum kifejezést, amely olyan neveket jelent, amelyek a pont után szerepelnek. Egy objektum attribútumai mind a példányváltozói, mind a metódusai. Hasonlóan, egy modul attribútumai a modulban definiált változónevek, függvények és osztályok. A példányváltozó helyett sokszor használják az adatattribútum elnevezést is.



(a) Termék objektumok

(b) Fájl objektumok

3.1. ábra. Objektumok

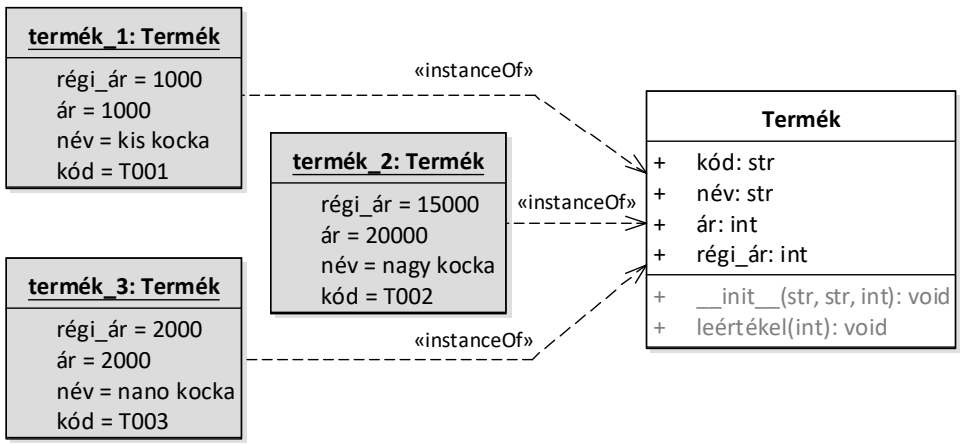
3.2. Mi az osztály?

Az osztály a hasonló objektumok közös szerkezetét írja le. A Python nyelvben ezek az osztályok (angolul *class*) az objektumok típusai (angolul *type*). Például, ha egy rendelés nyilvántartó programban a termékeket objektumként szeretnénk ábrázolni, akkor definiálunk egy Termék osztályt. Ez az osztálydefiníció azt írja le, hogy hogyan jönnek létre az új objektumok példányváltozói, és itt definiáljuk a metódusokat is, amelyek ezen példányváltozókat fogják olvasni vagy írni.

Az osztályok meghatározása a fejlesztés során kritikus: ha egyértelmű egy osztály felelősségi köre, vagyis, hogy a valóság mely aspektusait akarjuk modellezni, akkor a program sokkal érthetőbb lesz. Az osztályok felelősségi körének kijelölése csak a dokumentációban tud megjelenni, de ennek a döntésnek a következménye, hogy az osztályokban milyen példányváltozókat és metódusokat definiálunk. A termék-osztályunk teljesen másképp nézne ki, ha, mondjuk, vegyi anyagokkal dolgoznánk, és nemcsak a nevét, hanem az anyag veszélyességét és tárolási megszorításait is a termék példányváltozójaként kellene tárolni.

3.3. Objektumok létrehozása

Az eddigi példákban az adott osztályhoz tartozó objektumokat úgy hoztuk létre, hogy vagy értékként definiáltuk, vagy kifejezések eredményeként. Az objektumokat létrehozhatjuk osztályokból is. Az objektumok létrehozását nevezzük



3.2. ábra. Termék objektumok és az osztály

példányosításnak, mivel ez olyan, mintha egy elvont fogalom/kategória/általánosítás konkrét valós megjelenését/példányát hoznánk létre. Amikor osztályból szeretnénk létrehozni objektumot, akkor az osztálynév után teszünk egy zárójelet, mintha egy függvényt hívnánk. Az osztálytól függően adhatunk meg paramétereket, amelyek a létrejövő objektum példányváltozóinak az értékét befolyásolják általában.

Például, ha az `int` osztályt példányosítjuk, akkor létrejön egy `int` típusú objektum, amelynek 0 lesz az értéke. Az `int` típus példányosításakor egy lehetséges paraméter egy karaktersorozat lehet, ami ha számot tartalmaz, akkor azzal megegyező értékű szám fog létrejönni.

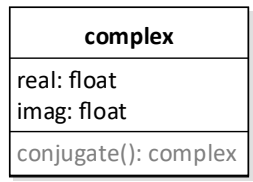
```
1 EGESZ_SZAM_0 = int()
2 EGESZ_SZAM_5 = int('5')
```

3.1. példa. Számobjektumok létrehozása

3.4. Példányváltozók és metódusok használata

A korábban már tárgyalt típusok közül a komplex számoknak van egy `real` és egy `imag` példányváltozója. Ezekre úgy tudunk hivatkozni, hogy a komplex számra hivatkozó változónév után teszünk egy pontot, és megadjuk a példányváltozó nevét: `x.real` és `x.imag`. A metódusok esetében a pont és metódusnév után zárójel következik. Ez a zárójel a függvényeknél látott módon állhat

üresen, vagy szerepelhetnek benne paraméterek. Például a komplex számoknak egy fontos metódusa van, a `conjugate()`.



3.3. ábra. Komplex szám osztály

```
I = complex(0, 1) # 0+1j
valos_resze = I.real
kepzetes_resze = I.imag
konjugaltja = I.conjugate()
```

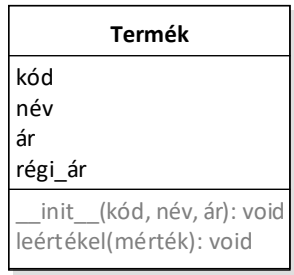
3.2. példa. Műveletek komplex számokkal

3.5. Osztályok definiálása

Fentebb láttuk, hogy hogyan történik egy osztály felhasználása. Nézzük meg egy példán keresztül, hogy hogyan kell saját osztályt definiálni. A 3.3. példában a 3.4. ábrán látható terméket reprezentáló osztályt fogjuk létrehozni. A 3.4. példában mutatjuk be ennek az osztálynak a példányosítását. Az osztály definíciója a `class` kulcsszóval kezdődik, és az osztály neve után kettőspont és maga az osztály elemeinek definícióit tartalmazó blokk következik. Az osztályhoz általában tartozik egy, a példány alapbeállítását végző inicializáló metódus `__init__()` néven (ez hasonló más programozási nyelvek konstruktor fogalmához). Az osztály definíciója tartalmaz egy `learaz()` metódust, amely a `Termek` típusú objektumok árral kapcsolatos példányváltozóit módosítja leértékelés esetén. Az osztályban definiált metódusok első paramétere mindig magára az objektumra hivatkozik, és a konvenciók szerint `self`-nek hívjuk.

```
class Termek:
    def __init__(self, kod, nev, ar):
        self.kod = kod
        self.nev = nev
        self.ar = ar
        self.regi_ar = ar

```



3.4. ábra. Termék osztály

```
8       def learaz(self, szazalek):
9           self.regi_ar = self.ar
10          uj_ar = self.ar * (1 - szazalek/100)
11          self.ar = round(uj_ar)
```

3.3. példa. Termék osztály definiálása

Az inicializáló metódus definiálja az újonnan létrejött objektum példányváltozóit a kezdőértékük beállításával. Ennek a metódusnak a paraméterei lesznek azok az értékek, amelyeket a példányosításnál az osztály neve utáni zárójelben megadunk a 3.4. példában látható módon.

```
1 k01 = Termek('K01', 'kocka', 1000)
```

3.4. példa. Termék objektum létrehozása

A 3.5. példa első sorában azt láthatjuk, hogyan definiálunk egy újabb `k02` objektumot. A második sorban kiírjuk mind a korábban definiált `k01`, mind a most definiált `k02` objektumok árait és neveit. A harmadik sorban megváltoztatjuk a `k01` objektum árát és a negyedik sorban a `k02` objektum nevét. Az utolsó sorban kiírjuk ismét mind a két objektum nevét és értékét. Ez a példa jól demonstrálja, hogy bár mindkét objektum példányváltozóit ugyanabban az osztálydefinícióban írtuk le, értékeik az objektumokra egyedien jellemzők és egymástól függetlenül változtathatók.

```
1 k02 = Termek('K02', 'kis kocka', 500)
2 print(k01.nev, k01.ar, k02.nev, k02.ar)
3 k01.ar = 1100
4 k02.nev = 'közepes kocka'
5 print(k01.nev, k01.ar, k02.nev, k02.ar)
```

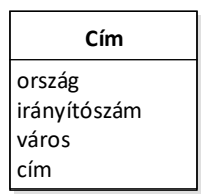
3.5. példa. Példák az objektum példányváltozóinak használatára

A 3.6. példa első sorában szintén kiírjuk az objektumok neveit és értékeit. A második sorban meghívjuk a leárazást végző metódust. Ez csak a `k01` példányváltozóit változtatja meg, ahogy ezt az utolsó sorban lévő kiírás alapján ellenőrizhetjük is.

```
1 print(k01.nev, k01.ar, k02.nev, k02.ar)
2 k01.learaz(30)
3 print(k01.nev, k01.ar, k02.nev, k02.ar)
```

3.6. példa. Példa az objektum metódusának használatára

Van olyan szélsőséges eset is, amikor egy osztály csak példányváltozókat tartalmaz. Ilyenre a 3.5. ábrán látható `Cim` osztály egy jó példa, mivel csak példányváltozókból áll: ország, irányítószám, város, cím. Az osztály definíciója a 3.7. példában látható.



3.5. ábra. Cím osztály

```
1 class Cim:
2     def __init__(self, orszag, irszam, varos, cim):
3         self.orszag = orszag
4         self.irszam = irszam
5         self.varos = varos
6         self.cim = cim
```

3.7. példa. Cím osztály definiálása

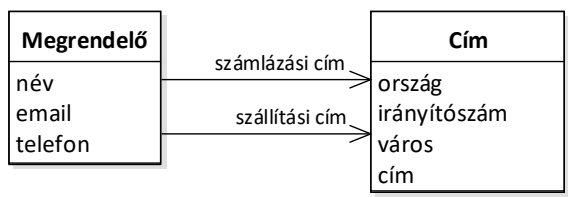
Az objektumoknak egyedi azonosítója van, amely nem változik a létrehozás után. Ezt az `id()` függvénnyel tudjuk lekérdezni, amelynek az eredménye egy egész szám. Ha egy adott időpillanatban két objektum azonosítója megegyezik, akkor az ugyanaz az objektum. Az objektumok metódusait és példányváltozóit a `dir()` függvénnyel kérdezhetjük le. A 3.8. példában láthatjuk a fenti két függvény meghívását és az eredményeik kiírását.

```
termek = Termek('K01', 'kocka', 1000)
print(id(termek), dir(termek))
```

3.8. példa. Egy objektum azonosítójának és attribútumainak a kiírása

3.6. Osztályok közötti kapcsolatok

A 3.6. ábrán azt láthatjuk, hogy egy megrendelőnek egy számlázási címe és egy szállítási címe van. Mind a két cím típusa – azaz az osztálya – `Cim`. A `Cim` osztály definícióját láthattuk az 3.7. példában. A `Megrendelo` osztály 3.9. példában szereplő definíciójából láthatjuk, hogy a címekre való hivatkozások sima példányváltozóként jelennek meg.



3.6. ábra. Megrendelő osztály

```
class Megrendelo:
    def __init__(self, nev, email, telefon,
                 szallitasi_cim,
                 szamlazasi_cim=None):
        self.nev = nev
        self.email = email
        self.telefon = telefon
        self.szallitasi_cim = szallitasi_cim
        self.szamlazasi_cim = szamlazasi_cim
```

3.9. példa. Megrendelő osztály

Egy vállalati rendszerben a rendeléseket szeretnénk modellezni osztályokkal a 3.7. ábra szerint. Az egyszerűség kedvéért feltételezzük, hogy egy megrendelésben csak egy termékből rendelnek tetszőleges darabszámot. Magának a rendelést modellező osztálynak a definícióját a 3.10. példában láthatjuk. A többi osztály definícióit már ismerjük a korábbi példákból.