Programozás Python nyelven 2.

IV. Modulok
V. Objektum orientált programozás
VI. String, List, Dictionary, Set objektumok
VII. Fájlkezelés, kivételkezelés

4.1. Modulok

Miért használunk modulokat?

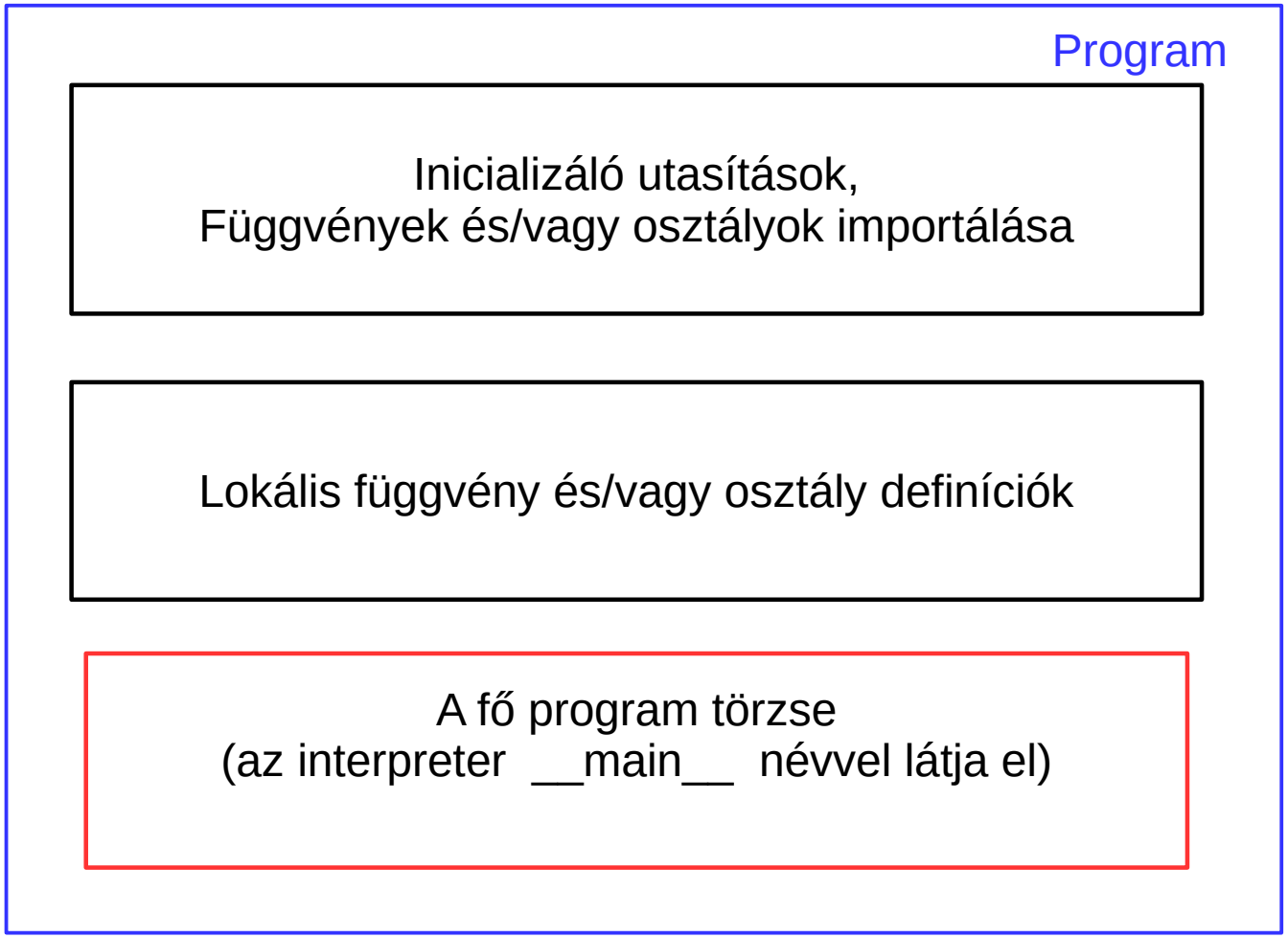
- a függvényekhez hasonlóan a modulok segítségével tudjuk a programunkat kisebb egységekre, alprogramokra osztani → dekompozíció (bonyolultabb probléma felosztása sok egyszerűbb problémára)
- így programunk átláthatóbb, egyszerűbb felépítésű lesz →
- könnyebben módosítható, hibakeresés, újra felhasználás egyszerűbb

Modul

- python kódot tartalmazó szöveges fájl → fájlnév kiterjesztése 'py'
- a fájl neve egyben a modul neve is (.py nélkül) !
  pl. valszam.py → modulnév: valszam
- egy modul tartalmazhat definíciókat, utasításokat, függvényeket, osztályokat (OOP, később) → tehát a függvénynél nagyobb egység
- a Python tartalmaz sok beépített, szabványos modult → ezeket betöltés (importálás) után lehet használni (de van amelyiket alapból betölti)
- illetve természetesen készíthetünk saját modulokat
- egy python alkalmazás egy főprogramból és több modulból állhat
- egy modul általában összetartozó dolgokat tartalmaz → globális változók, vagy egy függvénytár (pl. matematikai), vagy egy osztály csoport
- a legtöbb beépített függvény is külön modulokban van

4.2. Modulok

Egy Python nyelvű program szerkezete:



4.3. Modulok

Modul betöltése, használata

1. ha így töltjük be → import modulnév
 - a modulban lévő változók, függvények a modulnéven keresztül érhetők el →

pl. import time (bináris változat jön létre → modulnév.pyc)
 ltime=time.localtime() # modulban lévő függvény meghívása

 - vagy hozzárendelhetők helyi változóhoz, függvénynévhez

pl. import math
 kpi=math.pi # modulban lévő konstans elérése
 print(kpi)

2. de betölthetjük így is → from modulnév import függvénynév
 - az így betöltött függvényt, változót mint helyit látjuk → csak a nevét kell megadni

pl. from random import randrange
 x=randrange(6) # modulban lévő függvény meghívása

4.4. Modulok

Modul betöltése, használata

3. át is nevezhetjük betöltéskör (pl. rövidítjük)
   import modulnév as uj_név
   vagy from modulnév import függvénynév as uj_név

   pl. from random import randrange as randr
   x=randr(6)

4. így is betölthetjük a modul összes függvényét, változóját →
   from modulnév import * (globális névtérbe kerülnek)

   pl. from math import *

! kerülendő, csak ha nagyon sok mindent használunk a modulból

Modul futtatása

1. futtathatjuk közvetlenül, mint scriptként → így fő program lesz
→ a modulnak a __name__ attributuma (tulajdonsága) __main__ lesz

2. betölthetjük másik modulba (import) → így nem fő program lesz
→ a modulnak a __name__ attributuma (tulajdonsága) __modulnév__ lesz

4.5. Modulok

Modul futtatása

3. lehet egy törzs a modulban, amely csak akkor hajtódik végre, ha fő programként indítottuk a modult ! →

```
if __name__ == ' __main__ ' :
    utasítás1
    utasítás2
    ...
```

4. a modul legfelső szintjén álló változó és függvény deklarációk csak egyszer futnak le, ha a modult esetleg többször importáljuk akkor is !

5. modulok keresési útvonala → sys modul path változója (sys.path) → PYTHONPATH környezeti változó, alapértelmezett könyvtár (/usr/local/lib/python), aktuális könyvtár

új könyvtár felvétele → sys.path.append(' /…/... ')

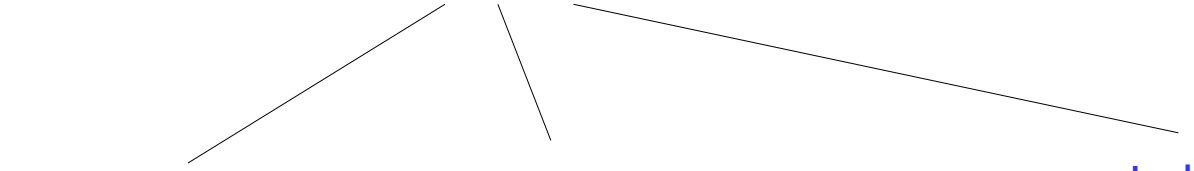
4.6. Modulok

Névterek és változók hatóköre

- **névtér** (namespace): itt kerülnek kötésre egymással → egy objektum – egy név

Binding (kötés)
Rebinding (átkötés)
Unbinding (törlés)

- több névtér létezik egyidejűleg



Beépített
(__builtin__)

állandó,
modul,
először ez jön
létre

Globális

állandó,
modul,
majd ez jön létre

Lokális

átmeneti,
függvény,
importált modul

4.7. Csomagok

csomagok

- a modulok csoportosítására szolgálnak → hierarchikus modul szerkezet

→ csomag1.csomag2.modul1
csomag1.csomag2.modul2
csomag1.csomag3.modul3
….

fizikai megvalósítás

- A csomag struktúrának megfelelő könyvtár struktúrát kell létrehozni a fájlrendszerben, és azokban elhelyezni a modulokat
- minden ilyen könyvtárban kell lennie egy → __init__ .py nevű fájlnak

pl.
```
csomag1/
        __init__ .py
        csomag2/
                __init__ .py
                modul1.py
                modul2.py
        csomag3/
                __init__ .py
                modul3.py
```

4.8. Modulokkal kapcsolatos beépített függvények

dir(modul)

- a modulban definiált összes modul, változó és függvény nevét adja vissza listában (sztringek listája)

  pl. dir(math)

- nem írja ki az automatikusan betöltött neveket !
  → __builtins__ modul (beépített szabványos modul)

- alkalmazása a __builtins__ modulra → dir(__builtins__)

globals( )

- az adott helyről elérhető globális nevek szótára

locals( )

- az adott helyről elérhető lokális nevek szótára

reload(modulnév)

- újra importálja a modult → újra lefut a modul inicializáló része

4.9. Matematikai beépített függvények

A **math** modul sok matematikai függvényt (illetve állandót) bocsájt rendelkezésünkre, DE ! importálni kell a modult

from math import * vagy import math

pi → a π állandó értékét tárolja ez a változó

e → az e állandó értékét tárolja ez a változó

pl. print(math.e) → 2.718281828459045

sqrt(szám)

- **szám** négyzetgyökét adja vissza

pl. print(math.sqrt(25)) → 5.0

sin(szög)

- **szög** szinuszát adja vissza (szög radiánban !)

pl. print(math.sin(math.pi/4)) → 0.7071067811865475

cos(szög)

- **szög** koszinuszát adja vissza (szög radiánban !)

pl. print(math.cos(math.pi)) → -1.0

4.10. Matematikai beépített függvények

tan(szög)

- **szög** tangensét adja vissza (szög radiánban !)

pl. print(math.tan(math.pi/4)) → 1.0

asin(szám) acos(szám) atan(szám)

- **szám** inverz szinuszát/koszinuszát/tangensét adja vissza (radiánban !)

pl. print(math.atan(1)) → 0.7853981633974483 (radianban)
print(180*math.asin(1)/math.pi) → 90.0 (fokban)

exp(x)

- exponenciális függvény (e^x)

pl. print(math.exp(2)) → 7.389056098930649
print(math.e*math.e) → 7.389056098930649

log(x)

- természetes logaritmus függvény → ln(x)

pl. print(math.log(7.389056098930649)) → 2.0

4.11. Matematikai beépített függvények

log10(x)

- 10-es alapú logaritmus függvény → lg(x)

pl. print(math.log10(10000)) → 4.0

radians(fok) **degrees(rad)**

- átváltás → fok-radián illetve radián-fok

pl. print(math.radians(180)) → 3.141592653589793
print(math.degrees(math.pi/4)) → 45.0

factorial(szám)

- **szám** faktoriálisát adja vissza (x!)

pl. print(math.factorial(4)) → 24 (4*3*2*1)

trunc(szám)

- egész számra levágja a kapott számot

pl. print(math.trunc(34.56788)) → 34
print(math.trunc(-34.56788)) → -34

4.12. Random modul beépített függvényei

A **random** modul véletlen számok előállításához tartalmaz függvényeket, használatukhoz importálni kell a modult

from random import * (vagy import random)

random( )

- 0 és 1 közötti 'véletlen' valós számot ad vissza

  pl. veletl=random.random( )

randrange(vége) vagy **randrange(kezd,vége)**

- 0 és vége-1 vagy kezd és vége-1 tartományból ad 'véletlen' egész számot

pl. veletl=random.randrange(6) → 0,1,2,3,4,5 közül választ

randrange(kezd,vége,lépés)

- kezd és vége-1 tartományból ad 'véletlen' egész számot, DE nem minden egész jöhet szóba, csak azok, amelyek n*lépés távolságra vannak a kezd-től

pl. veletl=random.randrange(3,36,4)
→ 3,7,11,15,19,23,27,31,35 közül választ

print(random.randrange(2,10,2))
→ 2,4,6,8 közül választ

5.1. Objektumok, osztályok

Miért használunk objektumokat?

- segítségükkel, a függvényekhez hasonlóan, a programunkat tudjuk strukturálni, felosztani kisebb, átláthatóbb egységekre
- így programunk átláthatóbb, egyszerűbb felépítésű lesz
- újra felhasználás egyszerűbb lesz

Objektum

- modell a való világról → egy bonyolult „változó”
- a különböző objektumok („példányok”) egymástól függetlenek
  → egységbe zárás (encapsulation)
- az objektumok belső változói és működése el van rejtve a külvilágtól →
- csak meghatározott eljárásokkal/függvényekkel hozzáférhetők (metódusok)
- a meglévő objektumokból lehet újat létrehozni → leszármaztatás

Osztály

- „adat szerkezet” → objektum típus
- leszármaztatás → „szülő osztály” - „gyermek osztály”
- gyermek osztály: örökli a szülő osztály minden tulajdonságát, függvényeit plusz újakat is kaphat !

5.2. Objektumok, osztályok

Osztály létrehozása

- egy függvény definícióhoz hasonló, csak „def” helyett → class
- adatokat (attribútumok) és függvényeket (metódusok) tartalmaz

```
class  osztály_neve:
        'osztály leírása'
        osztály-változó definíciók        # osztály-attribútumok
        osztály-függvény definíciók          # metódusok
```

pl.
```
class  pont:                          # pont típus → x,y koordináták
        'egy pont'
        db=0                          # osztály-attribútum
        def koordinata_kiir(paraméterek):          # metódus
        ...
```

Objektum (példány) létrehozása

- hasonló mint egy függvény hívása →

```
objektum_név = osztály_név(paraméterek)
```

pl.
```
A_pont = pont(10,30)        # 1. objektum
B_pont = pont(20,60)        # 2. objektum
```

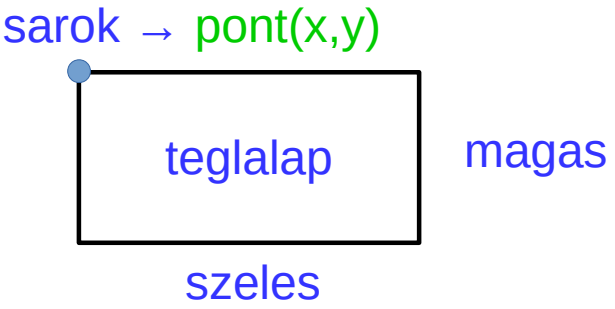
5.3. Objektumok, osztályok

Hasonlóság, egyediség

- ha két objektum tartalma ugyanaz → akkor is külön objektumok, és nem egyenlőek !
- ha két változó ugyanarra az objektumra mutat akkor egyenlő a két változó (alias)

Objektumok

- függvény paramétere lehet objektum
- függvény visszatérési értéke is lehet objektum
- egy objektumnak saját névtere van
- objektumban lehet objektum !




```
pl. class  pont:          # pont típus → x,y koordináták
        'egy pont'
    class  teglalap:          #téglalap típus →szélesség, magasság és
        'egy téglalap'        #          bal felső sarok x,y koordinátái
    tegla1 = teglalap( )          # téglalap példány létrehozása
    tegla1.szeles = 50            # szélesség példány-attribútumnak értékadás
    tegla1.magas = 30             # magasság példány-attribútumnak értékadás
    tegla1.sarok = pont( ) # bal-felső sarok példány-attribútum létrehozása
    tegla1.sarok.x = 20           # sarok példány-attribútum x attribútuma
    tegla1.sarok.y = 10           # sarok példány-attribútum y attribútuma
```

5.4. Attribútumok

Attribútum (változó attribútum)

- az objektum vagy osztály valamilyen jellemzője, amelyet egy változó tárol
- két típusa van: objektum (példány) illetve osztály attribútum

Osztály attribútum

- az osztályhoz kötődnek, nem a példányhoz !
- példány sem szükséges elérésükhöz →
- elérésük az osztálynéven keresztül
- felhasználása: példányok közös változójaként → pl. példányok darabszáma
- definíció:

```
class  osztály_neve:
        'osztály leírása'
        osztály-attribútum definíciók          # osztály-attribútumok
```

pl.
```
class  pont:
        'egy pont'
        db=0            # osztály-attribútum
```

- hivatkozás rá → osztály_neve.osztály-attribútum_neve

Pl. pont.db = 1

5.5. Attribútumok

Objektum (példány) attribútum

- objektum változó → a példányhoz kötődnek →
- egyedi változók, bár a nevük ugyanaz
- elérésük a példányon keresztül
- pl.

```
class  pont:                          # pont típus → x,y koordináták
     'egy pont'

A_pont = pont( )        # 1. objektum létrehozása
B_pont = pont( )        # 2. objektum létrehozása
A_pont.x = 10                # 1. objektum  x koordinátája
A_pont.y = 30                # 1. objektum  y koordinátája
B_pont.x = 21                # 2. objektum  x koordinátája
B_pont.y = 7                 # 2. objektum  y koordinátája
```

- az előző példa nem túl szép ! → bár a dinamikus értékadás miatt lehet így létrehozni változókat (objektum attribútumokat) DE nem így kell !! →
- speciális metóduson (konstruktor) keresztül hozzuk létre az objektum attribútumokat !!
- módosítani sem célszerű így az attribútumokat !
  → mert akkor nem zárt az objektum → módosításuk metódusokon keresztül

5.6. Attribútumok

Speciális osztály attribútumok

- minden osztály alapértelmezetten tartalmaz ilyeneket

__dict__

Tartalmazza az osztály attribútumait, és azok értékeit → szótár

__doc__

Dokumentációs karakterlánc

__module__

A modul neve, amelyben az osztály definiálva van

pl. az előző példában létrehozott „pont” osztály és A_pont, B_pont objektumok esetén →

```
print(pont.__doc__)          # kiír→  egy pont
print(pont.__module__)       # kiír→  __main__
print(pont.__dict__)         # kiír→
                {'__module__':'__main__','__doc__':'egy pont'}
```

5.7. Metódusok

Metódus (függvény attribútum)

- speciális függvény, egy objektumhoz vagy osztályhoz tartozik
- az osztályt látja el funkcionalitással → objektumok „viselkedése”
- a metódusok által tudunk az objektumokkal/osztályokkal különböző feladatokat, műveleteket elvégezni, vagy rajtuk valamilyen műveletet végezni
- ugyanabba az egységbe (objektum) lesznek bezárva így az adatok és az azokat kezelő függvények is
- hasonlóan kell definiálni mint egy függvényt, De egy osztály definíció belsejében
- a metódus első paraméterének mindig egy példány hivatkozásnak kell lennie ! → célszerű ennek mindig a „self” nevet adni

```
pl.  class  ido:                    # idő osztály létrehozása (típus)
         'idő osztály'
          def  ora_kiir(self):            # metódus definíció
               print('Idő: %d óra, %d perc'  % (self.ora,self.perc))
     …
     most = ido( )       # idő példány létrehozása
     most.ora = 10            # ora példány-attribútumnak értékadás
     most.perc = 25           # perc példány-attribútumnak értékadás
     …
     most.ora_kiir()     # idő kiíratása  → 10 óra, 25 perc
```

5.8. Metódusok

Konstruktor metódus

- automatikusan végrehajtódik egy példány létrehozásakor ! →
- felhasználása: példány attribútumok létrehozása, kezdőértékek adása
- neve kötelezően ! → '__init__( )'

```
pl.  class  ido2:                          # idő2 osztály létrehozása (típus)
          'jobb idő osztály'

          def  __init__(self):             # konstruktor metódus definíció
               self.ora = 12          # ora példány-attribútum
               self.perc = 0          # perc példány-attribútum
          def  ora_kiir(self):             # metódus definíció
               print('Idő: %d óra, %d perc' % (self.ora,self.perc))
    …
    most2 = ido2( )       #  példányosítás → beállítódik ora és perc attribútum is
    most2.ora_kiir()      # idő kiíratása  → 12 óra, 0 perc
    most2.perc = 42       # perc attribútum módosítása (így  nem ajánlott)
    most2.ora_kiir()      # idő kiíratása  → 12 óra, 42 perc
```

5.9. Metódusok

Konstruktor metódus paraméterei

- több paramétere is lehet → egy példány létrehozásakor adhatunk át argumentumokat, így tudunk kezdő értékeket átadni

```
pl.  class  ido3:                          # idő3 osztály létrehozása (típus)
         'még jobb idő osztály'

         def  __init__(self,hh=0,mm=0):      # konstruktor metódus definíció
              self.ora = hh                  # ora példány-attribútum
              self.perc = mm                 # perc példány-attribútum
         def  ora_kiir(self):                # metódus definíció
              print('Idő: %d óra, %d perc' % (self.ora,self.perc))
    …
    most3 = ido3(12,42 ) #  példányosítás → beállítódik ora és perc is
    most3.ora_kiir()     # idő kiíratása  → 12 óra, 42 perc
    most3.ora = 8        # óra attribútum módosítása (így  nem ajánlott)
    most3.ora_kiir()     # idő kiíratása  → 8 óra, 42 perc
```

5.10. Metódusok

Statikus metódus

- olyan metódus, amelyet példány nélkül használunk →
  az osztályhoz tartozik, osztály metódus
- meghívása az osztálynéven keresztül (példány sem kell) →
  osztály_neve.osztály-metódus_neve
- objektum megadása is lehetséges:
  osztály_neve.osztály-metódus_neve(objektum)

pl.

```
class  matek:                          # osztály létrehozása (típus)
    'matek osztály'
     def  negyzet(szam):                    # statikus metódus definíció
         return szam*szam

x = matek.negyzet(4 )
print(x)
```

Destruktor metódus

- speciális metódus → példányok törlésekor hajtódik végre !

- neve kötelezően ! → '__del__( )'

5.11. Metódusok

1. minta feladat

- metódusok és attribútumok használatának bemutatása

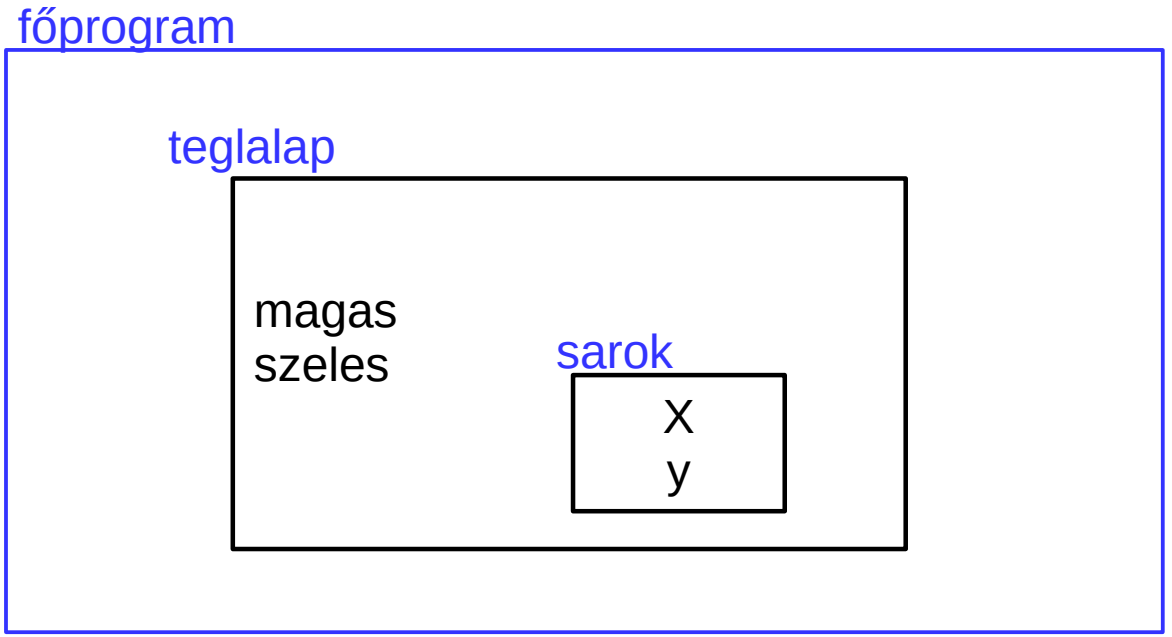
```
class  pont2:                          # pont2 osztály létrehozása
      'pont osztály'

      db =0           # osztály-attribútum
      def  __init__(self,xk=0,yk=0):          # konstruktor metódus definíció
            self.x = xk            # x példány-attribútum
            self.y = yk            # y példány-attribútum
            pont2.db = pont2.db+1    # darabszám növelése
      def  kiir(self):             # metódus definíció
            print('koordináta, x: %d, y: %d'  % (self.x,self.y))

 Apont = pont2(10,30)       # 1. objektum
 Bpont = pont2(20,60)       # 2. objektum
 print('pontok száma: %d' % (pont2.db))     # kiír → pontok száma: 2
```

5.12. Névterek

Névterek
- minden osztálynak saját névtere van, az itteni változók → az osztályváltozók
- minden objektumnak saját névtere van, az itteni változók → a példányváltozók
- az osztályok használhatják, DE nem módosíthatják a főprogram szintjén definiált változókat
- az objektumok használhatják, DE nem módosíthatják az osztályok szintjén és a főprogram szintjén definiált változókat
- minden modulnak, és függvénynek is saját névtere van



5.13. Öröklés, polimorfizmus

Öröklés

- egy meglévő osztályt felhasználhatunk új osztály létrehozására → objektum orientált programozás (OOP) egyik előnye
- az új, leszármaztatott osztály örökli a szülő osztály valamennyi jellemzőjét ! (attribútumok, metódusok)
- de ! A leszármaztatott osztály rendelkezhet néhány eltérő vagy kiegészítő funkcionalitással → leszármaztatás (derivation) →
- plusz saját attribútumokat, metódusokat kaphatnak, vagy az öröklött metódusok megváltoztathatóak (más viselkedés)
- több szülő osztály is lehet ! (többszörös öröklés)
- definiálása:

```
class  új_osztály_neve(szülő_osztály_neve):
    ...
```

pl.

```
class  pont3(pont2):
    'spec. pont osztály'
```

5.14. Öröklés, polimorfizmus

2. minta feladat

```
# öröklés bemutatása

    class  pont2:                          # pont2 osztály létrehozása
        'pont osztály'
        db =0           # osztály-attribútum
        def  __init__(self,xk=0,yk=0):          # konstruktor metódus definíció
            self.x = xk           # x példány-attribútum
            self.y = yk           # y példány-attribútum
            pont2.db = pont2.db+1    # darabszám növelése
        def  kiir(self):              # metódus definíció
            print('koordináta, x: %d, y: %d'  % (self.x,self.y))

    class  pont3(pont2):                              # pont3 osztály létrehozása
        'spec. pont osztály'
        ptipus =3           # új, plussz osztály-attribútum

     Apont = pont2(10,30)       # 1. objektum
     Bpont = pont3(20,60)       # 2. objektum
     print('pontok száma: %d' % (pont3.db))     # kiír → pontok száma: 2
```

5.15. Öröklés, polimorfizmus

Polimorfizmus

- ha a leszármaztatott osztályban létrehozunk egy ugyanolyan nevű metódust mint a szülő osztályé → akkor felülírjuk a metódust ! →
- polimorfizmus: ugyanazon néven hívott metódus különbözőképpen fog viselkedni attól függően, hogy melyik osztályból hívtuk meg ! (lényegében más metódusok, csak ugyanaz a nevük)

pl.

```
class  szulo:
    def kiir(self):
        print('szülő')

class  gyerek(szulo):
    def kiir(self):
        print('gyerek')
```

- de a leszármaztatott osztályban is el tudjuk érni a szülő osztály felülírt metódusát (a példány hivatkozást át kell adni) →
  pl. a 'gyerek' osztályból a 'szulo' osztály kiir metódusa
  → szulo.kiir(self)

5.16. Objektumokkal kapcsolatos beépített függvények

hasattr(objektum,attribútum)
- van-e az objektumnak ilyen nevű attribútuma ?

getattr(objektum,attribútum)
- attribútum értékének lekérdezése

setattr(objektum,attribútum)
- attribútum értékének beállítása (vagy új attribútum létrehozása így !)

delattr(objektum,attribútum)
- attribútum törlése

dir(osztály) vagy **dir(példány)**
- osztály/példány attribútumok neveinek (változó + metódus) listázása

vars(osztály)
- osztály attribútumok neveinek, és azok értékeinek listázása

isinstance(objektum,osztály) vagy **isinstance(objektum,típusobjektum)**
- az objektum az osztály (vagy típusobjektum) példánya-e ?
  (1-t ad vissza ha igen)

issubclass(osztály1,osztály2)
- osztály1 az osztály2 alosztálya-e ? (1-t ad vissza ha igen)

6.1. Sztring objektumok metódusai

split( ) vagy **split('szeparátor')**

- darabolás listává, alap szeparátor (elválasztó) a szóköz

pl.
```
sztri1 = 'egy szoveg, nem?'
li1 = sztri1.split( )          #→ ['egy', 'szoveg,', 'nem?']
li2 = sztri1.split(',')        #→ ['egy szoveg', ' nem?']
```

join(lista)

- sztring lista egyesítése,
  a szeparátor karakterre kell alkalmazni !!

pl.
```
li3 = ['egy', 'ketto', 'harom']
sztri2= '_'.join(li3)          → 'egy_ketto_harom'
```

index('karakter')

- 'karakter' első előfordulása a sztringben

pl.
```
sztri1 = 'egy szoveg, nem?'
print(sztri1.index('g'))       → 1
```

find('részsztring')

- 'részsztring' első előfordulásának pozíciója a sztringben

pl.
```
sztri1 = 'egy szoveg, nem szoveg?'
print(sztri1.find('szov'))     → 4
```

6.2. Sztring objektumok metódusai

strip( )

- szóköz eltávolítás sztring elejéről és végéről

pl. sztri1 = ' egy szoveg, nem? '
sztri2 = sztri1.strip( ) → 'egy szoveg, nem?'

replace('ch1','ch2')

- 'ch1' karakter kicserélése 'ch2' karakterre a sztringben

pl. sztri1 = 'egy szoveg, nem?'
print(sztri1.replace('g','-')) → 'e-y szove-, nem?'

count('részsztring')

- 'részsztring' előfordulásainak száma a sztringben

pl. sztri1 = 'egy szoveg, szoveg, ket szoveg?'
print(sztri1.count('szov')) → 3

lower( ) upper( )

- sztring kisbetűssé illetve nagybetűssé alakítása

6.3. Lista objektumok metódusai

sort( )

- rendezés

pl.
```
li1 = [3,4,5,6,9,8,2,1]
li1.sort()
print(li1)                  #   → [1,2,3,4,5,6,8,9]
```

reverse( )

- elemek sorrendjének megfordítása

pl.
```
li2 = [3,4,5,6,9,8,2,1]
li2.reverse()
print(li2)                  #   → [1,2,8,9,6,5,4,3]
```

extend(sorozat**)**

- sorozat elemeinek hozzáfűzése a listához

pl.
```
li3 = [3,4,5,6]
li3.extend([7,8,9])
print(li3)                  #   → [3,4,5,6,7,8,9]
```

count(elem**)**

- elem hányszor szerepel a listában

index(elem**)**

- elem első előfordulásának indexe

6.4. Lista objektumok metódusai

remove(elem**)**

- elem törlése

pl.
```
li4 = [3,4,5,6,8,9]
li4.remove(5)
print(li4)                # → [3,4,6,8,9]
```

append(elem**)**

- új elem hozzáfűzése a lista végéhez

pop(index**)** vagy **pop( )**

- visszaadja és törli ! a kért indexű elemet, vagy az utolsó elemet !

pl.
```
li4 = [3,4,5,6,8,9]
li4.append(10)
print(li4)                # → [3,4,5,6,8,9,10]
li4.pop()
print(li4)                # → [3,4,5,6,8,9]
```

insert(index,elem**)**

- új elem beszúrása a listába (index mutatja a helyet)

clear()

- a lista elemeinek törlése → üres lista lesz ! pl. li4.clear()

6.5. Lista objektumok metódusai

copy()

- lista másolása

pl.

```
li5 = [3,4,5,6,7,8]
li6 = li5.copy()
print(li6)          #  → [3,4,5,6,7,8]
li5.clear()
print(li5)          #  → []
print(li6)          #  → [3,4,5,6,7,8]
```

Lista veremként (LIFO)

elem felvétele lista végére → lista1.append(újelem)
elem kivétele a lista végéről → elem= lista1.pop( )

Lista sorként (FIFO)

elem felvétele listába (végére) → lista1.append(újelem)
legrégebbi elem kivétele a listából → elem= lista1.pop(0)

6.6. Lista létrehozása

1. Elemei megadásával pl. li5 = [3,4,5,6,7,8]

2. üres lista feltöltésével

pl.

```
li5 = []
li5.append(2)
li5.append(5)
...
```

3. másolással lista_másolat = lista_eredeti[:]

pl.

```
li5 = [3,4,5,6,7,8]
li6 = li5[:]
```

4. list(sorozat) egy sorozatból list() metódussal

- listából **list(egy_lista)**

pl.

```
li5 = [3,4,5,6,7,8]
li6 = list(li5)          # lista létrehozása másik listából (másolás ez is)
print(li6)               #   → [3,4,5,6,7,8]
li5.clear()
print(li6)               #   → [3,4,5,6,7,8]
```

6.7. Lista létrehozása

4. list(sorozat**)**

- sztringből **list(**egy_sztring**)**

pl.

```
st5 =  "ez egy string"
li6 = list(st5)              # lista létrehozása sztringből
print(li6)                   #   → ['e','z',' ','e','g','y',' ','s','t']
```

- tuple-ból **list(**egy_tuple**)**

pl.

```
tup5 =(3,4,5,6,7,8)
li7 = list(tup5)                 # lista létrehozása tuple-ből
print(li7)                       #   → [3,4,5,6,7,8]
```

- sorozatból **list(**range()**)**

pl.

```
li8 = list(range(1,10,2))          # 1-től 10-ig  kettesével
print(li8)                         #   → [1,3,5,7,9]
```

6.8. Lista létrehozása

5. listaépítő kifejezéssel (List comprehensions)

- **lista_név = [kifejezés(x) for x in sorozat]**
  a sorozat tagjai sorban behelyettesítődnek a kifejezésbe, és az eredmények kerülnek a listába

pl.

```
li9 = [2*y*y for y in range(1,5)]        # → [2*1*1,2*2*2,2*3*3,2*4*4]
print(li9)                               #   → [2,8,18,32]
```

összetett kifejezés is lehet ! pl. ami eredménye tuple lesz

pl.

```
li9 = [1,2,3,4]
li10 = [(x**2, x**3) for x in li9]
print(li10)                              #   → [(1,1),(4,8),(9,27),(16,64)]
```

- **lista_név = [kifejezés(x) for x in sorozat if feltétel]**
  a sorozat tagjai - ha megfelelnek a feltételnek ! - sorban behelyettesítődnek a kifejezésbe, és az eredmények kerülnek a listába

pl.

```
from math import log10 as lg
li9 = [4,10,30,100, 50,6]
li10 = [lg(x) for x in li9 if x>10]      # → [lg(30),lg(100),lg(50)]
print(li10)                              #   → [1.47712,2.0,1.69897]
```

Több változó, és több for is lehet !

6.9. Szótár objektumok metódusai

keys( )

- a kulcsok listáját adja meg

values( )

- az értékek listáját adja meg

pl.
```
szo4 = {'egy':'one','két':'two','há':'three'}
print(szo4.keys())          #  → ['egy','két','há']
print(szo4.values())        #  → ['one','two','three']
```

get(kulcs,hamisvissza)

- egy kulcshoz tartozó értéket adja meg (ha van ilyen kulcs) → ha nincs → akkor a második paraméter lesz a visszaadott érték !!

pl.
```
szo4.get( 'egy', 'none')
```

has_key(kulcs)

- a szótár tartalmazza-e az adott kulcsot ?

6.10. Szótár objektumok metódusai

items( )

- a szótárt transzformálja egy tuplekből álló listává → [(kulcs1, ertek1),(kulcs2, ertek2),(kulcs3, ertek3),……]

clear( )

- az összes elem törlése !! → üres szótár

copy( )

- szótár másolása ! → mert szotar1=szotar2 → nem hoz létre másolatot !!

up_date(masikszotar)

- felveszi az elemek közé egy másik szótár elemeit

7.1. Fájlkezelés

Munkavégzés fájlokkal

1. fájl megtalálása → elérési út, név (operációs rendszer elérése ! → később)

2. fájl megnyitása
 - hogy használni tudjunk egy fájlt, először meg kell nyitni → open(fájl,mód)

    a függvény egy fájl objektumot ad vissza, ezen keresztül tudjuk elérni a fájlt

3. fájl olvasása/írása → read( ) vagy write() függvény használata
 - általában csak az egyiket egyszerre
 - sorban vagy bizonyos pozíciótól

4. fájl lezárása
 - használat után a fájlt be kell zárni → close()

A fájl tartalmát mindig tekinthetjük karakter sorozatnak →
sztring kezelő függvényekkel feldolgozhatóak

7.2. Fájlkezelés

Szekvenciális írás fájlba

- fájl megnyitása → open(fnév,mód)

  a függvény egy file objektumot ad vissza

  pl. fajl1=open('/home/py/proba1.txt','w')

- sztring írása a fájlba →
  A file objektum write('szöveg') metódusával

  pl. fajl1.write("első szöveg")

- karakter láncok listájának írása a fájlba →
  A file objektum writelines('sztring_lista') metódusával

  pl. fajl1.writelines(["szó1","szó2","szó3"])

- fájl lezárása
  → A file objektum close( ) metódusával

  pl. fajl1.close( )

Megnyitási módok
'r' → olvasásra
'w' → írásra (régi törlődik !)
'a' → hozzáfűzésre
'r+' → olvasásra és írásra
'w+' → írásra és olvasásra
'a+' → hozzáfűzésre és olvasásra

Bináris módok →
hasonlóak csak plusz 'b'
pl. 'rb' → olvasásra
'wb' → írásra

Egy sorba ír !!

7.3. Fájlkezelés

Fájl szekvenciális olvasása

- fájl megnyitása → open(fnév,mód) függvénnyel

  pl. fajl2=open('/home/valaki/proba1.txt','r')

- egy sor beolvasása →
  A file objektum readline( ) metódusával, pl. sorbe=fajl2.readline( )

- meghatározott számú karakter olvasása →
  A file objektum read(darab) metódusával,

  pl. adatbe=fajl2.read(10) → 10 karakter beolvasása (az aktuális pozíciótól)

- teljes fájl beolvasása !→
  A file objektum readlines( ) metódusával, (nincs spec. karakter konverzió !)
  pl. fajlbe=fajl2.readlines( ) → a sorok listáját adja vissza

  vagy
  A file objektum read( ) metódusával, pl. fajlbe=fajl2.read( )

- fájl lezárása
  → A file objektum close( ) metódusával, pl. fajl2.close( )

7.4. Fájlkezelés

1. mintafeladat

Fájl másolása

```
# 1. mintafeladat, fájl másolása

def fmasol(forras,cel):
    fbe=open(forras,'r')
    fki=open(cel,'w')
    while 1:            #  végtelen ciklus
        adat=fbe.read(50)   # következő 50 byte beolvasása
        if adat=='':        # ha nincs mit beolvasni → kilépés
            break
        else:               # ha volt beolvasott adat
            fki.write(adat)     # → kiírás a másolatba
    fbe.close()
    fki.close()

# fő program
fmasol('f1.txt','f2.txt')       #   'f1.txt'  fájlnak léteznie kell !!
```

7.5. Szöveg fájl kezelése

Szövegfájl → karakterek, betűközök, sorvége jel

Létrehozás, írás

pl.
```
f1=open('fajl3.txt','w')
f1.write('első sor \n masodik sor \n')
f1.write('harmadik sor \n')
f1.close( )
```

Beolvasás

```
f1=open('fajl3.txt','r')
adat= f1.readline( )        # egy sor beolvasása → sztring
sorok= f1.readlines( )      # sorok  beolvasása → sztring lista
f1.close( )
```

7.6. Szöveg fájl kezelése

2. mintafeladat

Fájl másolása, szűréssel → '#' karakterrel kezdődő sorok kihagyása !

```
# 2. mintafeladat, fájl szűrése
def fszuro(forras,cel):
    fbe=open(forras,'r')
    fki=open(cel,'w')
    while 1:            #  végtelen ciklus
        adat=fbe.readline()      # következő sor beolvasása
        if adat=='':             # ha nincs mit beolvasni → kilépés
            break
        else:                    # ha volt beolvasott adat
            if adat[0]!='#':
                fki.write(adat)        # → kiírás a másolatba
    fbe.close()
    fki.close()
# fő program
fszuro('f1.txt','f2.txt')      #   'f1.txt'  fájlnak léteznie kell !!
```

7.7. Változók mentése és beolvasása

A fájlok írására és olvasására szolgáló függvények sztringeket fogadnak illetve adnak vissza → a helyes visszaalakítás valamilyen más változó típusra nem mindig egyszerű !

Ezért egyéb változók írására, olvasására célszerű a pickle modult használni → import pickle

mentés fájlba → pickle.dump(változó, fájlobjektum)

olvasás fájlból → pickle.load(fájlobjektum)

Írás

```
import pickle
f1=open('fajl4.txt','w')
pickle.dump(30,f1)
pickle.dump(12.25,f1)
pickle.dump(2325,f1)
f1.close( )
```


Beolvasás

```
f1=open('fajl4.txt','r')
szam1= pickle.load( f1)   # egész ! (30)
szam2= pickle.load( f1)   # float ! (12.25)
szam3= pickle.load( f1)   # egész ! (2325)
f1.close( )
```

7.8. File objektumok egyéb metódusai

seek( )

- mozgás a fájlban → a kurzor pozíció állítása

fájlobjektum.seek(eltolás, honnan)

honnan → 0 – fájl elejétől, 1 – aktuális pozíciótól,
2 – fájl végétől

tell( )

- a kurzor fájlon belüli pozícióját adja meg

pl.

```
fajl1=open('f1.txt','w+')        # megnyitás írásra és olvasásra
fajl1.write('012345678901234 5')
mosthol=fajl1.tell( )
print(mosthol)                   # → 16
fajl1.seek(8,0)
mosthol=fajl1.tell( )
print(mosthol)                   # → 8
adatbe=fajl1.read(3)             # 3 karakter beolvasása
print(adatbe)                    # → 90
fajl1.close( )
```

7.9. Standard bemenet, kimenet

A sztenderd kimenet (képernyő) illetve bemenet (billentyűzet) és hibacsatorna (szintén képernyő) elérése a sys modulon keresztül lehetséges (import sys)

kimenet → stdout
bemenet → stdin
hibacsatorna → stderr

Írás (print helyett)

```
sys.stdout.write(valami)
```

Olvasás (raw_input helyett)

```
be = sys.stdin.readline( )
```

7.10. Fájlrendszer kezelése

A különböző operációs rendszerek eléréséhez többféle modul létezik.

- általános → os modul
- Unix → posix modul
- Windows → nt modul
- Macintosh → mac modul

os modul beépített függvényei

rename( )

- fájl átnevezése

```
os.rename('regi-nev','uj-nev')
```

remove( )

- fájl törlése

access(út,mód)

- hozzáférési engedély vizsgálata

chmod(út,mód)

- hozzáférési engedély módosítása

umask(maszk)

- maszk beállítása

7.11. Fájlrendszer kezelése

os modul beépített függvényei, mappák kezelése

getcwd( )
- aktuális mappa lekérdezése

chdir(útvonal)
- könyvtár (mappa) váltás → pl. Linux esetén # os.chdir('/home/kis')

mkdir(mappa) vagy **mkdir(mappa, mód)**
- mappa létrehozása (az aktuálisban !) # pl. os.mkdir('m1')

listdir(mappa)
- mappa tartalmának listázása → lista !

pl. (Windows esetén)

```
os.mkdir('m1\m8')
os.mkdir('m1\hh')
li1 =  os.listdir('m1')
print(li1)                  # →  [ 'hh', 'm8' ]
```

rmdir(mappa)
- mappa törlése # pl. os.rmdir('m1\m8')

7.12. Fájlrendszer kezelése

os.path modul beépített függvényei, elérési utak kezelése

exists(útvonal)

- létezik-e az útvonal

pl. print(os.path.exists('m1\hh')) # → True

isdir(útvonal)

- az útvonalnév könyvtár-e

pl. print(os.path.isdir('m1\hh')) # → True

isfile(útvonal)

- az útvonalnév fájl-e

basename(útvonal)

- a teljes elérési útból a fájlnevet adja vissza

pl. (Windows esetén)

```
szt1=os.path.basename('m1\hh\zz1.txt')
print(szt1)                    #→  zz1.txt
```

dirname(útvonal)

- a teljes elérési útból a mappák nevét adja vissza

split(útvonal)

- az útvonalat ketté bontja könyvtár névre és bázis névre

7.13. Fájlrendszer kezelése

os.path modul beépített függvényei, elérési utak kezelése

splitdrive(útvonal)
- az útvonalat ketté bontja meghajtó névre és útvonal névre
  Windows esetén ! (tuple formában adja vissza)

splitext(útvonal)
- a fájl nevét ketté bontja a tényleges alap névre és a kiterjesztésre
  (tuple formában adja vissza)

join(útvonal1, útvonal2, ...)
- két vagy több útvonalat egyetlen útvonalnévvé egyesít

getsize(fájl)
- a fájl hosszát adja meg (bájtban)

getatime(fájl)
- a fájl utolsó hozzáférési ideje

getmtime(fájl)
- a fájl utolsó módosítási ideje