Programozás Python nyelven 1.

I. Python nyelv alapjai, adattípusok, változók
II. Elágazások, ciklusok
III. Függvények

1.1. Python nyelv, bevezetés

Python nyelv:

- magas szintű nyelv
- nem fordító program van, hanem Python értelmező-futtató környezet szükséges egy python kód futtatásához
- Jelenleg kétféle Python verzió elérhető/használatos → 3-as és a 2-es (ez a régi) → pontosabban a 3.x.y és a 2.a.b verziók

(jelenleg: Python 3.10.5)

Vannak jelentős különbségek közöttük → az egyikben megírt-fejlesztett program valószínűleg nem fog futni a másikon (változtatás nélkül) !!

- A telepített futtató környezet tartalmaz egy egyszerű fejlesztő környezetet → IDLE a neve → egy szövegszerkesztő (editor) és egy python parancssor (shell) részekből áll → az IDLE shellben egyesével lehet utasításokat futtatni → a '>>>' prompt után kell beírni az utasítást, majd enter leütése
- a forráskódot tartalmazó python fájl kiterjesztése 'py' →tehát egy python forrás fájl neve pl. program1.py
- Python fejlesztő környezet többféle is van→ pl. Eric, Pyzo, ... de elég egy egyszerű programozó szövegszerkesztő is, pl. Geany

1.2. Python nyelv, bevezetés

Fontos szintaktikai szabályok:

- az utasítások végét a sorvége jel mutatja → nem kell plusz speciális karakter !!
  (pl. mint C nyelv esetén a ';')
- az utasítások és utasítás blokkok (több, összetartozó utasítás egymás után) határait a **sortörés** definiálja
- utasításblokkok jelölése → behúzásokkal (azonos számú space-szóköz !!!)

```
utasítás1
utasítás2:
    utasítás3
    utasítás4
    utasítás5
    utasítás6
utasítás7
```

utasítás2-höz tartozó utasításblokk

- megjegyzések írása a programban → # jel után

```
utasítás1    # megjegyzés1
# megjegyzés2
utasítás2
```

1.3. Python nyelv, bevezetés

Adat típusok

Python nyelven sokféle típusú adattal dolgozhatunk, most csak a néhány leggyakrabban használt:

- string → több karakterből álló szöveg (sztring) tárolására szolgál,
  pl. „hello” „kettő” 'ez is sztring'
  → idézőjelek vagy aposztrófok közé kell írni !!
- int → egész szám pl. 2 245
- float → valós szám (egészek + törtek), lebegőpontos ábrázolással, pl. 23.45
- bool → logikai (boolean), két érték → True (1) és False (0)
- list → lista, több (akár különböző típusú) értéket tárol → egy általános tömb,
  pl. [2,4,6,8,10] [1,3,'öt',7,9,11]

Változó

- egy adatot tárol, értéke (a tárolt adat) általában változik a program futása során
- van neve (betűkből, számokból és az aláhúzás karakterből állhat) → igazából egy memória rekeszt címez meg (a RAM-ban tárolódnak)
- van típusa ! (milyen típusú adatot adatot tárol), és persze értéke (az adat, amit tárol)
- kis- és nagybetű különbözőnek számít !!
- normál változókat kisbetűvel kezdjük ! pl. szam1 sebesseg
- nagybetűkkel általában a konstansokat (értékük nem változik) nevezzük el
- aláhúzás karakterrel speciális változók kezdődnek !

1.4. Változók, értékadás

Értékadás változónak

- értékadás, bármikor a programban '=' használatával

```
szam1=5              #  szam1 értéke 5 lesz (int), amíg meg nem változtatjuk
szoveg='start'       #  szoveg értéke start lesz (string)
szam2=12.25          # szam2 értéke 12.25 lesz (float)
```

- tehát az egyenlőségjel általában a program nyelveknél értékadást jelent !! NEM egyenlőséget, egyenletet mint matekban !!

Változó értékének megváltoztatása

```
x=5          #  x értéke 5 lesz, amíg meg nem változtatjuk
x=8          #  x értéke 8 lesz mostantól
x=x+1        #  x értékét 1-el növeljük !!
```

Ez nem egyenlőség ! (nem találnánk olyan x számot, amelyre igaz lenne), hanem → a baloldali változó (x) felveszi a jobboldalon lévő kifejezés (x+1) értékét ! → tehát ez egy értékadás ! → ahol a változó előző értékével is tudunk számolni !
Vesszük x jelenlegi értékét (8), majd hozzáadunk 1-et és a kapott eredmény (9) lesz x változó új értéke

1.5. Változók, értékadás

Változó típusa, címe

- dinamikus típusadás → értékadáskor lesz típusa a változónak
  (a hozzárendelt értéktől függ, hogy milyen lesz, és akár változhat is !!!)
  → tehát előre nincs deklarálva, nem kell előre megadni

```
x=12.5          # x típusa float (valós) lesz
y='egy'         # y típusa string lesz
x='ketto'       # x típusa mostantól string lesz
```

- típus lekérdezése → type() függvény használatával
  type(változónév) pl. type(x)

- változó memória címének lekérdezése → id() függvény használatával
  id(változónév) pl. id(x)

Értékadás

- többszörös értékadás is lehetséges

```
x=y=10          # x értéke 10 lesz, és y értéke is 10 lesz mostantól
a,b=3,4         # a értéke 3 lesz, és b értéke 4 lesz
```

1.6. Műveletek, operátorok, kifejezések

Aritmetikai műveletek

- a négy alapművelet operátorai: + - * /
- hatványozás: **
- maradékos osztás: %
- egész osztás: // (Python 3. verzióban !!)

Értékadás (általánosabban) → változónév=kifejezés;

először kiértékelődik az egyenlőség jel jobb oldalán lévő kifejezés → és a kapott értéket veszi fel a baloldali változó

```
szam1=4+5*2                 # szam1 értéke 14 lesz !!
                            # (először a szorzás lesz elvégezve)
szam3=(szam1-2)/2     # szam3 értéke (14-2)/2= ??
```

Python 3 → 6.0 lesz (valós)!! Python 2 → 6 lesz (egész)!!

```
szam1=9%4             # szam1 értéke 1 lesz
                            # (az egész osztás maradéka)
szam1=szam1+2         # szam1 értéke 2-vel növelődik ! → 3
szam3=szam1**3        # szam3 értéke 3^3=27 lesz
```

1.7. Műveletek, operátorok, kifejezések

Relációs műveletek

- nagyobb: >
- kisebb: <
- egyenlő: ==
- nem egyenlő: !=
- nagyobb vagy egyenlő: >=
- kisebb vagy egyenlő: <=

Logikai műveletek, operátorok

- ÉS: and pl. a and b
- VAGY: or pl. c or d
- NEM: not pl. not d

```
pl. x<10          # igaz értéket ad ha x kisebb mint 10
    y==5          # igaz értéket ad ha y egyenlő 5-el
    (x>10) and (x<20)   # igaz értéket ad ha x 10 és 20 közé esik
                        # (nagyobb mint 10 ÉS kisebb mint 20)
```

Hozzárendelő operátorok (értékadás)

- sima értékadás: = pl. x=25 # x értéke 25 lesz
- növelés: += pl. x+=5 # x értéke 5-el növelődik (mint x=x+5)
- csökkentés: -= pl. x-=2 # x értéke 2-vel csökken (mint x=x-2)
- szorzás: *= pl. x*=3 # x értéke 3-szorosára nő (mint x=x*3)
- osztás: /= pl. x/=4 # x értéke negyedére csökken (mint x=x/4)
- maradék: %= pl. x%=2 # x értéke → mint x=x%2

1.8. Kiíratás a képernyőre

Print() függvény

- segítségével tudunk kiírni szövegeket, vagy kifejezések, változók értékeit a képernyőre
- **szöveg kiíratása**

pl. print('hello') # megjelenik → hello

régebbi Python verzióknál zárójelek nélkül (is) működött !!

pl. print 'hello'

- **kifejezés értékének kiíratása**

pl. print(4*15+25) # megjelenik → 85

- **változó értékének kiíratása**

pl. szam=24
print(szam) # megjelenik → 24

- több érték kiíratása is lehetséges egyszerre →

print ('hello ',szam)

1.9. Számok

Egész számok

int → 16 bites vagy 32 bites (vagy 64 bites ??) egész

értékadás történhet nemcsak 10-es számrendszerben !

- szam1=5 # 10-es számrendsz. → szam1 értéke 5
- szam2=0x24 # 16-os számr. → szam2 értéke 24_{16} lesz (36)
- szam3=0b1010 # 2-es számr. → szam3 értéke 1010_2 lesz (10)

Valós számok

float → valós szám (egészek + törtek)

- 8byte-os lebegőpontos ábrázolással
  - → 1 bit előjel + 52 bites mantissza (törtrész) + 11 bites karakterisztika (10 hatvány kitevője)
  - → 12 értékes számjegy !
  - → ábrázolható legkisebb szám: $\mathbf{10^{-308}}$
  - → ábrázolható legnagyobb szám: $\mathbf{10^{308}}$

- megadása többféleképpen lehetséges, pl.

10.55 .04 (0.04)

20. (20.0) # mert csak 20 → ez egész számot ad meg !!

2e4 ($2*10^4$ =20000.0) 3.5e-3 ($3.5*10^{-3}$ = 0.0035)

1.10. Számok

Komplex számok

complex → a+bj ahol 'a' és 'b' valós számok (vagy egészek)
→ valós rész (real) → 'a'
→ képzetes rész (imag) → 'b'

- Komplex változó megadása
  pl. komp1=4+5j komp2=2-1j komp3=-2.5+0.5j

- Műveletek
  pl. komp1 + komp2 = 6+4j
  komp1 - komp3 = 6.5+4.5j

- Egy már megadott komplex szám valós vagy képzetes részét külön-külön is le lehet kérdezni !
  → komp1.real → # komp1 valós értéke → 4.0
  → komp1.imag → # komp1 képzetes értéke → 5.0

1.11. Számok

1. mintafeladat

Téglalap kerületének és területének számítása
(a program lefordítva, futtatva Linux alatt, Geany szövegszerkesztőt és Python 3.10.5 értelmezőt használva)

```
#!/usr/bin/python
# -*- coding:Utf-8 -*-
# 1. mintafeladat

Aoldal=10
Boldal=15
Kerulet=2*Aoldal+2*Boldal;
Terulet=Aoldal*Boldal
print('Kerület: ',Kerulet)
print('Terület: ',Terulet)
```

Képernyőn kiírva

Kerület: 50
Terület: 150

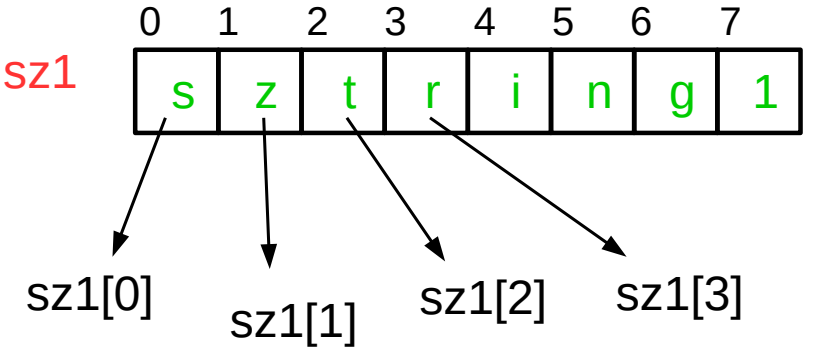
1.12. String

String (karakterlánc)

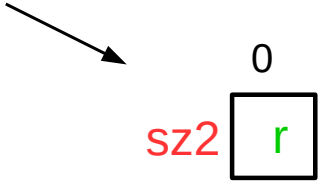
string
- összetett adattípus → szekvencia (elemek rendezett együttese)
- határolása → aposztrófokkal ('ez sztring') vagy idézőjelekkel ? („ez is sztring”)
- objektum ! és nem módosítható !!
- az egyes karakterek elérése indexeléssel → sztring_név[index]

pl.

sz1= 'sztring1'



sz2=sz1[3]



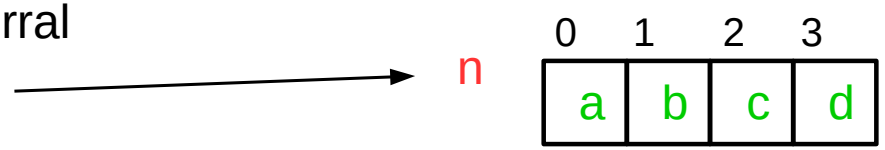
- speciális karakterek a sztringben → \ után → pl. \n (sortörés)
- háromszoros idézőjelekkel vagy aposztrófokkal bármilyen speciális karakterláncot lehet kreálni

1.13. String

Műveletek sztringekkel

- **összefűzés** → + operátorral

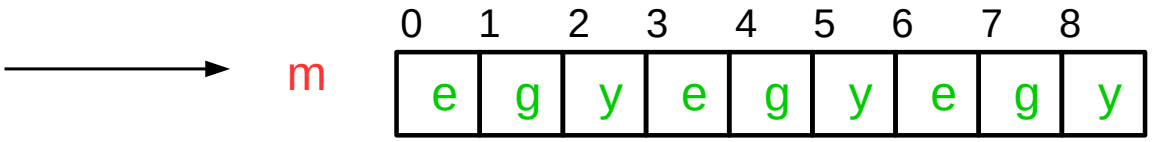
  pl. n = 'ab' + 'cd '



- **többszörözés** (ismétlés)

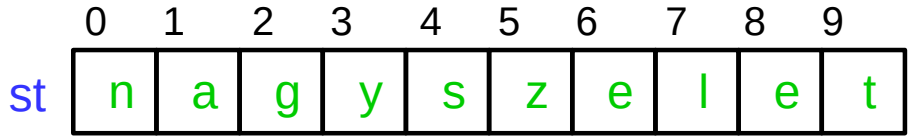
  * operátorral

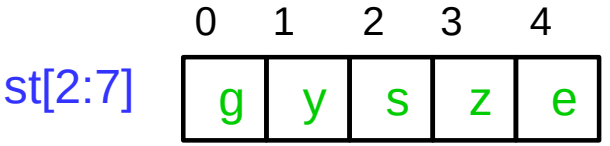
  pl. m = 'egy' *3

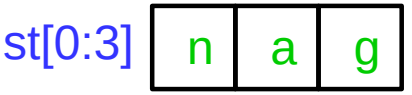


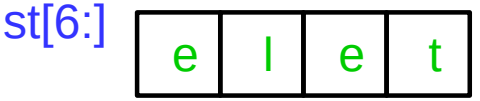
- **szeletelés (slicing)** → **string_neve[tól:ig]**

  pl. ha st = 'nagyszelet ' akkor









1.14. String

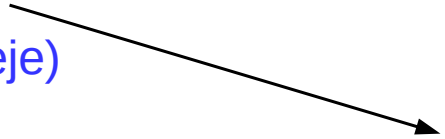
2. mintafeladat

Hozzunk létre egy „egynegy” nevű sztringet amely az '1' '2' '3' és '4' karaktereket tartalmazza egymás után, 5-ször ismételve ('123412341234…..'), majd hozzunk létre egy „eleje” nevű sztringet amely az „egynegy” sztring első 7 karakterét tartalmazza. Írjuk ki őket a képernyőre ! „eleje” sztringhez adjunk hozzá egy '?' karaktert !

```
#!/usr/bin/python
# -*- coding:Utf-8 -*-
# 2. mintafeladat
egynegy = '1234' * 5
eleje = egynegy[0:7]
print('egynegy: ',egynegy)
print('egynegy eleje: ',eleje)
eleje = eleje +  '?'
print('eleje újra: ',eleje)
```


Képernyőn kiírva

```
egynegy: 1234123412341234
egynegy eleje: 1234123
eleje újra: 1234123?
```



A sztring nem módosítható !!
De mégsem hibás ez a sor ! → ilyenkor egy új változó jön létre, ugyanolyan névvel
(a régi többé nem elérhető)

1.15. String

Karakterlánc előállítása formázással

formázósztring % (változók)

3. mintafeladat

```
szam1=25
string1='piros'
sz2='a sorszáma: %d, a színe: %s '  % (szam1,string1)
print(sz2)
```

Képernyőn kiírva

a sorszáma: 25, a színe: piros

- a zárójelben megadott változók sorban behelyettesítődnek a ”%” jelek helyére
- a ”%” jelek után álló betűk → konverziós markerek → megadják hogy a változók milyen formátumban értendők

konverziós markerek

%d vagy %i → előjeles egész (decimális)
%f vagy %g → valós szám (float)
%8.2g → valós szám,
mezőszélesség:8, tizedesjegyek:2

%s sztring
%c karakter
%x hexa szám

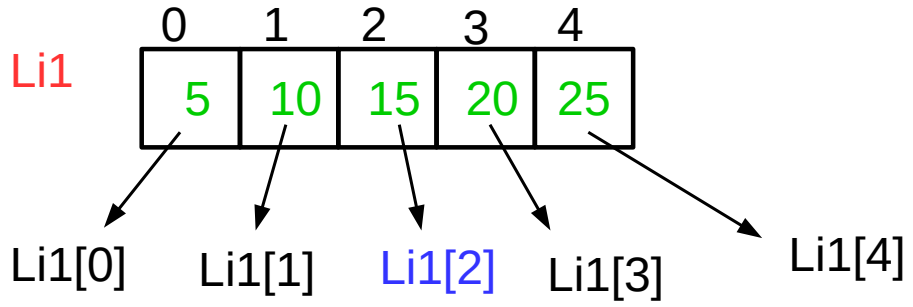
1.16. Lista

Lista

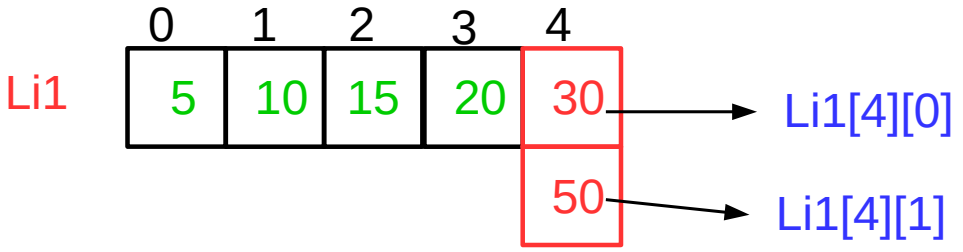
list
- összetett adattípus → szekvencia ez is (elemek rendezett együttese)
- általános tömb → az elemei lehetnek különböző típusúak !
- határolása, szögletes zárójelekkel → [„ez”,”egy”,”lista”], és ez is → [4,15,'hat',4]
- üres lista → [ ]
- objektum, módosítható !
- az egyes elemek elérése indexeléssel → lista_név[index]
- a listák egymásba ágyazhatóak (lista eleme lehet maga is lista)

pl.

Li1= [5,10,15,20,25] →



Li1[4]= [30,50] →

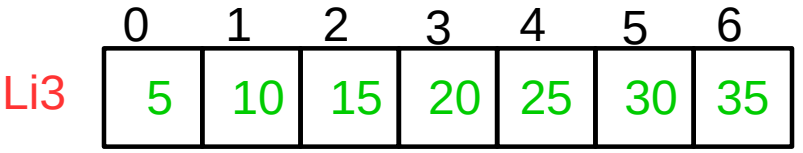


1.17. Lista

Műveletek listákkal

- **összefűzés** → + operátorral

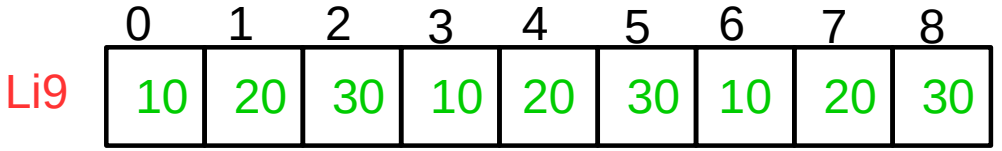
pl. Li1 = [5,10,15,20]
Li2 = [25,30,35]
Li3 = Li1 + Li2



- **többszörözés** (ismétlés)

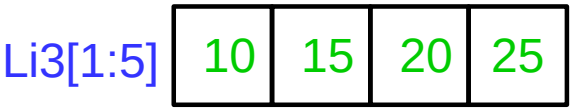
* operátorral

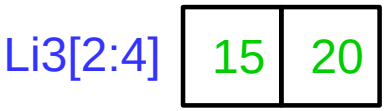
pl. Li4 = [10,20,30]
Li9 = Li4*3



- **szeletelés (slicing)** → **lista_neve[tól:ig]**

pl. ha Li3 = [5,10,15,20,25,30,35] akkor



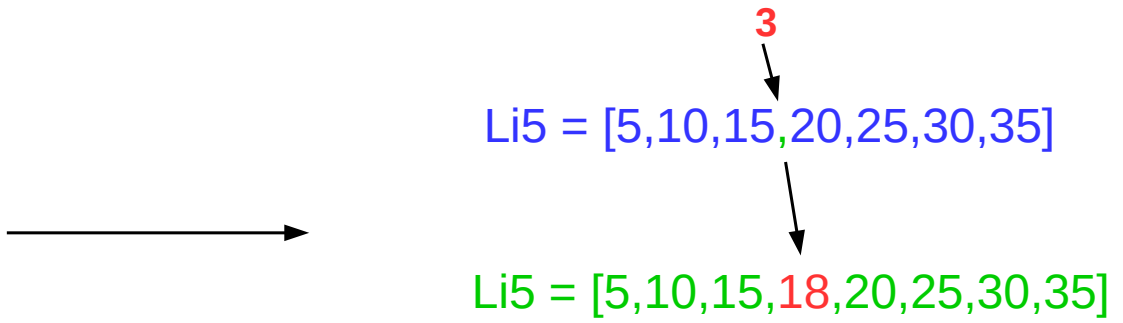


1.18. Lista

Műveletek listákkal

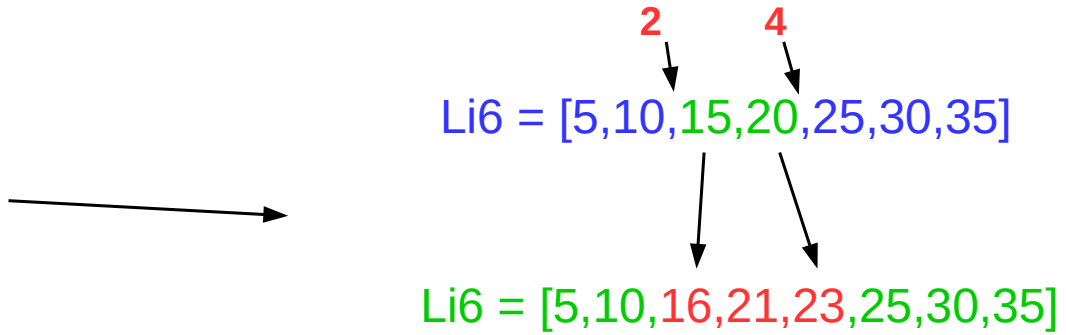
Beszúrás listába

Li5[3:3] = [18]



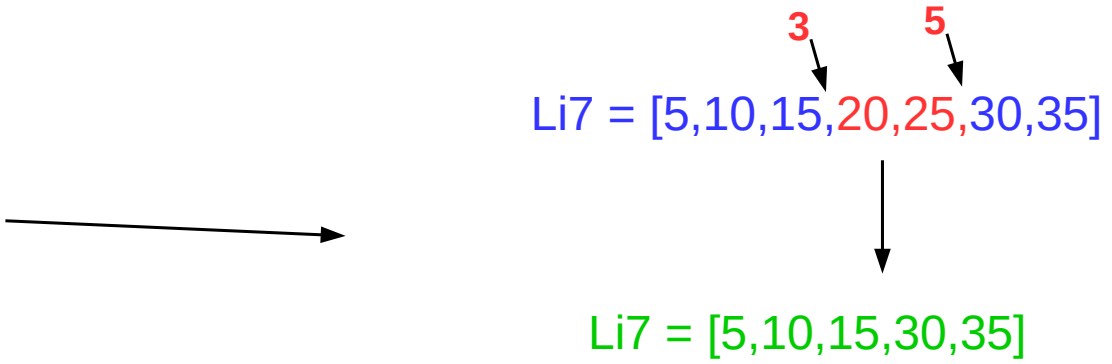
Helyettesítés listában

Li6[2:4] = [16,21,23]



Törlés listából

Li7[3:5] = [ ]



1.19. Lista

4. mintafeladat

Hozzunk létre egy „egyot” nevű listát amely az 1, 2, 3, 4 és 5 számokat tartalmazza egymás után, 2-szer ismételve
[1,2,3,4,5,1,2,3,4,5], majd hozzunk létre egy „masolat” nevű üres listát, amelybe másoljuk bele szeletelést és helyettesítést használva az „egyot” listát. Írjuk ki őket a képernyőre ! Végül a „masolat”-ból töröljük a középső 4 számot !

```
egyot = [1,2,3,4,5] *2
print('egyot: ',egyot)
masolat = [ ]
masolat[0:0]= egyot[0:10]
print('masolat: ',masolat)
masolat[3:7]= [ ]
print('masolat újra: ',masolat)
print('egyot újra: ',egyot)
```


Képernyőn kiírva

```
egyot: [1,2,3,4,5,1,2,3,4,5]
masolat: [1,2,3,4,5,1,2,3,4,5]
masolat újra: [1,2,3,3,4,5]
egyot újra: [1,2,3,4,5,1,2,3,4,5]
```

1.20. Lista

5. mintafeladat

A 4. mintafeladatot próbáljuk megoldani „egyszerűbb” másolással!

```
egyot = [1,2,3,4,5] *2
print('egyot: ',egyot)
masolat =  egyot
print('masolat: ',masolat)
masolat[3:7]= [ ]
print('masolat újra: ',masolat)
print('egyot újra: ',egyot)
```

→

Képernyőn kiírva

```
egyot: [1,2,3,4,5,1,2,3,4,5]
masolat: [1,2,3,4,5,1,2,3,4,5]
masolat újra: [1,2,3,3,4,5]
egyot újra: [1,2,3,3,4,5]
```

---

masolat = egyot →

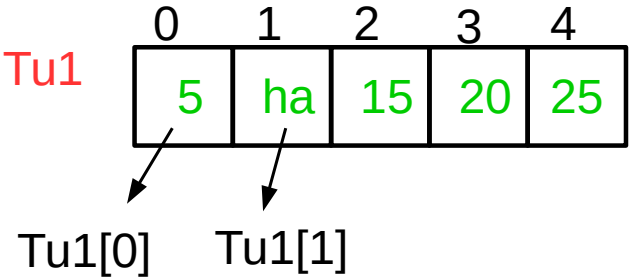
Ilyenkor nem jön létre új lista változó !! Csak lesz egy másik neve is ugyan annak a listának !! → mindkét névvel az eredetit látjuk, az eredeti listát módosítottuk !!

1.21. Tuple

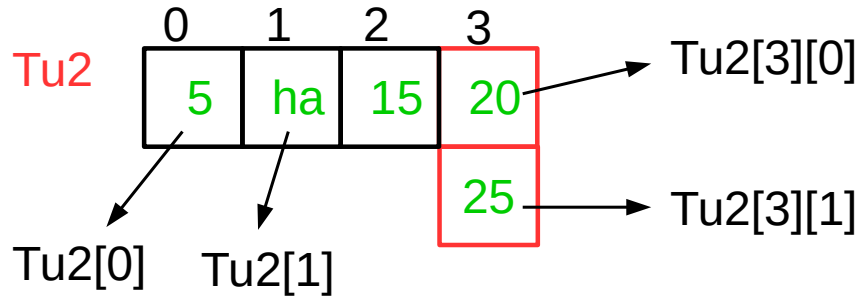
Tuple

olyan mint a lista, DE nem módosítható !!
- összetett adattípus → szekvencia ez is (elemek rendezett együttese)
- általános tömb → az elemei lehetnek különböző típusúak !
- határolása, zárójelekkel → („ez”,”egy”,”tuple”), ez is → (3,5,8,35),
  és ez is → (4,15,'hat',4), üres tuple → ( )
- az egyes elemek elérése indexeléssel → tuple_név[index]
- egymásba ágyazhatóak (tuple eleme lehet maga is tuple vagy lista)
- objektum ez is, de kevesebb memóriát foglal mint a lista !
- DE ! elemek hozzáadása lehetséges, és ha valamelyik eleme lista → az módosítható !!

pl.
Tu1= (5,'ha',15,20,25) →



pl.
Tu2= (5,'ha',15,[20,25]) →



1.22. Szótár

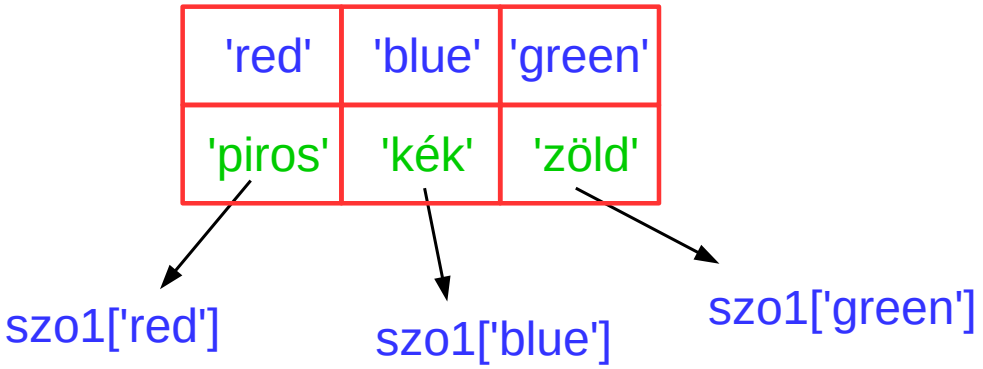
Szótár (dictionary)

dict

- összetett adattípus →de nem szekvencia (nem rendezett elemsorozat)
- elemei → kulcs:érték párok
- speciális index (a kulcs), ami nem csak szám lehet, hanem bármilyen
  nem módosítható adattípus → egész, valós, sztring, tuple
- határolása, kapcsos zárójelekkel →{'red':'piros','blue':'kék','green':'zöld'}
- objektum, módosítható ! üres szótár→ { }
- értékek bármilyen adattípusok lehetnek
- az egyes elemek elérése indexeléssel → szótár_név[kulcs]

pl.

szo1= {'red':'piros','blue':'kék','green':'zöld'}



1.23. Szótár

Műveletek szótárakkal

Elem beírása szótárba

szótár_név[kulcs]=érték

pl. szo1= {'red':'piros','blue':'kék','green':'zöld'}

szo1['white']= 'fehér'

'red'	'blue'	'green'	'white'
'piros'	'kék'	'zöld'	'fehér'

Elem törlése szótárból

del szótár_név[kulcs]

pl. del szo1['red']

~~'red'~~	'blue'	'green'	'white'
~~'piros'~~	'kék'	'zöld'	'fehér'

Teljes szótár törlése !! → del szótár_név

2.1. Vezérlő struktúrák

- A tevékenységek végrehajtási sorrendjét határozzák meg
- alapvető vezérlő struktúrák:
  - szekvencia
  - kiválasztás
  - ismétlés

1. szekvencia

Egymást követő utasítások sorozata.
Alap esetben a program utasítások egymás után hajtódnak végbe (ahogy a forráskódban egymás után szerepelnek)

2. kiválasztás (elágazás)

Feltételes végrehajtás, döntés. → egy ideig eltérő utasítások hajtódnak végre, attól függően hogy egy feltétel igaz vagy hamis volt → elágazások létrehozása a programban → **if**

3. ismétlés (ciklus)

Ismétlődő feladatok, utasítások végrehajtása
→ ciklusok → **while, for**

2.2. Feltételes utasítás

1. if … else...

- szintaktikája:

```
if feltétel:
    igaz ág utasításai
else:
    hamis ág utasításai
```

az else rész elhagyható !

- mindig csak az egyik ág utasításai hajtódnak végre, a feltétel logikai értékétől függően !

Logikai feltételben használatos operátorok

- összehasonlító → nagyobb: > kisebb: < egyenlő: ==
  nem egyenlő: != nagyobb vagy egyenlő: >= kisebb v. egyenlő: <=

- logikai → ÉS: and VAGY: or NEM: not

- tagságot vizsgáló operátor → in pl. x in sorozat → x in (1,2,3,4,5)
  # x tagja-e a sorozatnak ?

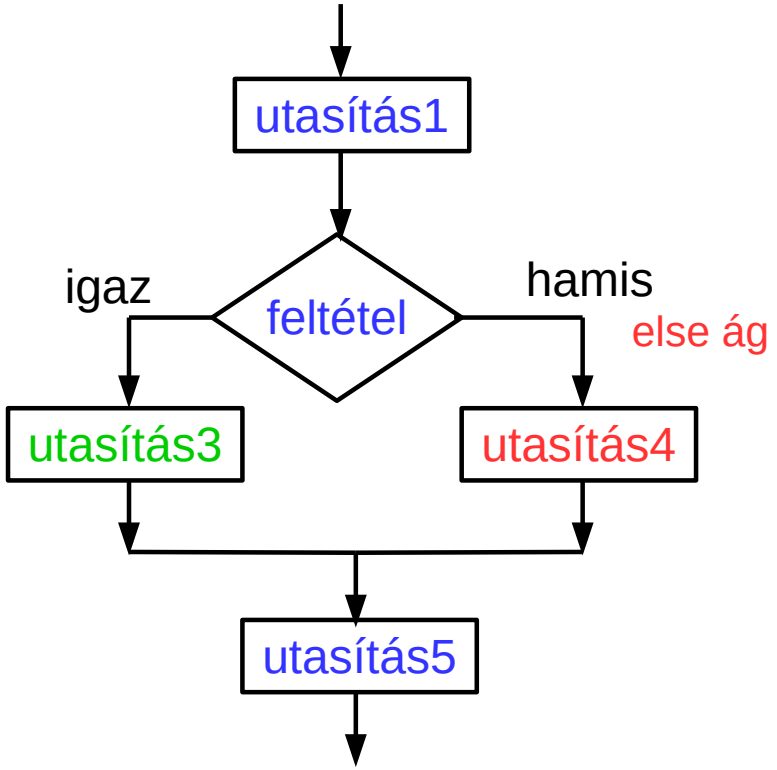
- azonosítást vizsgáló operátor → is
  pl. objektum1 is objektum2 → x is y
  # x és y ugyanaz az objektum (címük azonos) ?

2.3. Feltételes utasítás

2. if...else... utasítás használata

- használata

```
utasítás1
if feltétel:
    utasítás3
else:
    utasítás4
utasítás5
```



pl.

```
x=a-b
if x<0:
    print('x kisebb mint 0 !')
    y=1
else:
    print('x nem kisebb mint 0 !')
    y=5
z=x+y
```

2.4. Feltételes utasítás

if...else... utasítás használata

pl. a programban folyamatosan számolni kell 1-től 8-ig, majd kezdeni elölről (1,2,3,4,5,6,7,8,1,2,3, ….)

```
        …
        ...
    if szamlal<8:
       szamlal=szamlal+1                  #  vagy → szamlal+=1
    else:
       szamlal=1
        …
```

megoldás másféleképpen, else ág nélkül:

```
        ...
    szamlal=szamlal+1                  #először növelünk
    if szamlal>8:
       szamlal=1                       #  ha túlléptük a határt →
        …                              #  kezdőérték beállítása újra
```

2.5. Feltételes utasítás

3. if … elif… else...

Többszörös elágazás (elif → else if)

- szintaktikája:

```
if feltétel1:
    feltétel1 igaz  utasításai
elif feltétel2:                    # több elif ág is lehet!
    feltétel1 hamis, de feltétel2 igaz  utasításai
else:
    mindkét feltétel hamis  utasításai
                                   # az  else rész elhagyható !
```

- mindig csak az egyik ág utasításai hajtódnak végre itt is

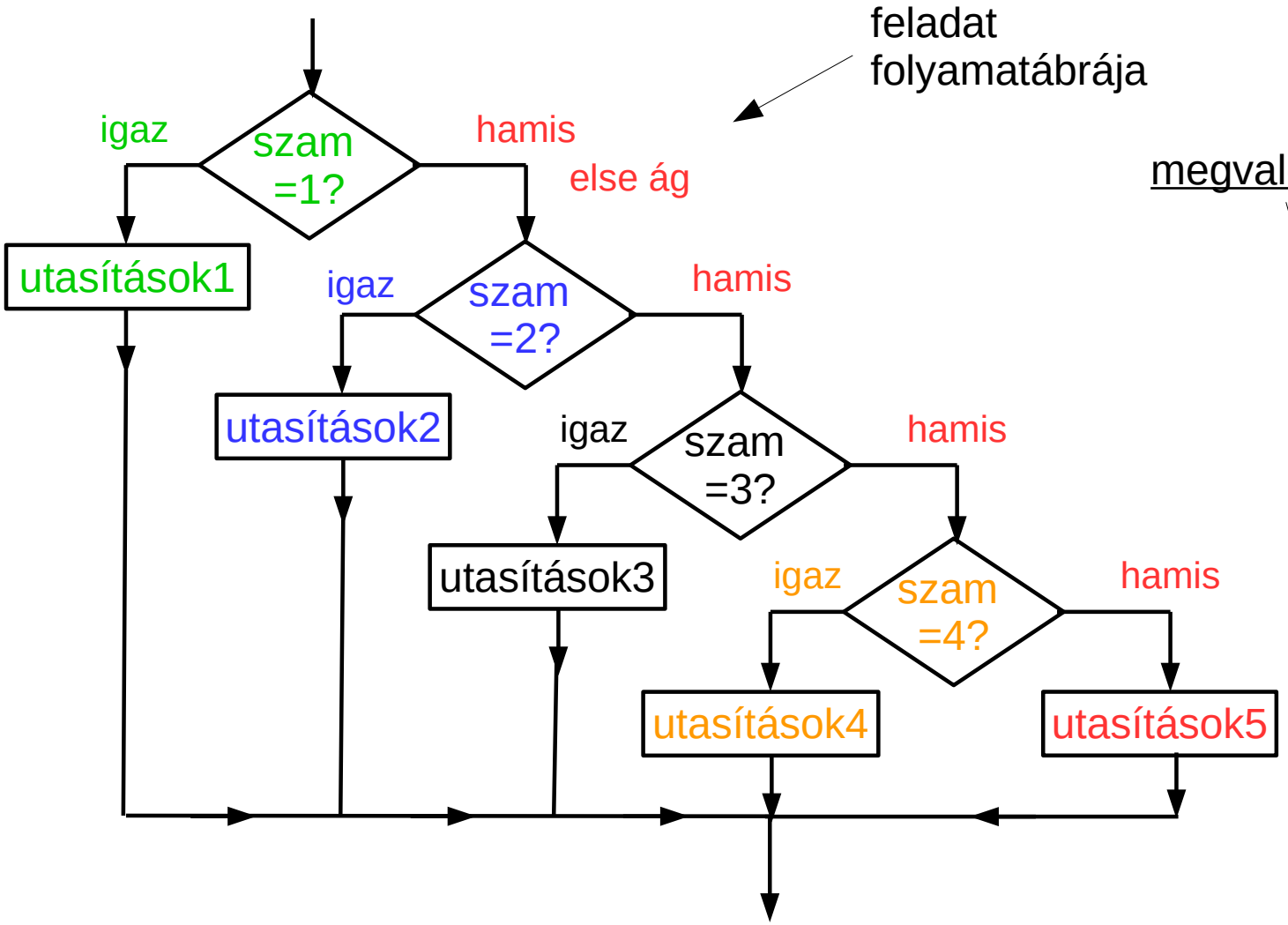
- használata pl.

```
if x>10:
    print('x nagyobb mint 10 !')
elif x==10:
    print('x egyenlő 10-el !')
else:
    print('x kisebb mint 10 !')
```

2.6. Feltételes utasítás, mintafeladatok

1. minta feladat

pl. egy szám különböző értékeitől függően mást kell csinálnia a programnak, legyen pédául 4 lehetséges érték (1,2,3,4)



megvalósítás

4 db egymásba ágyazott if...else

if...elif...else szerkezet

2.7. Feltételes utasítás, mintafeladatok

1. minta feladat (if...elif...else... használata)

pl. egy szám különböző értékeitől függően mást kell csinálnia a programnak, legyen pédául 4 lehetséges érték (1,2,3,4)

```
...
if szam==1:
        maxszam=90
        szamdb=5
elif szam==2:
        maxszam=45
        szamdb=6
elif szam==3:
        maxszam=35
        szamdb=7
elif szam==4:
        maxszam=80
        szamdb=20
else:   # a szám  nem jó
        print(' a szam értéke csak 1,  2,  3  vagy 4 lehet !!')
...
```

2.8. Feltételes utasítás, mintafeladatok

1. minta feladat (egymásba ágyazott if...else... szerkezetekkel)

pl. egy szám különböző értékeitől függően mást kell csinálnia a programnak, legyen pédául 4 lehetséges érték (1,2,3,4)

```
...
if szam==1:
    maxszam=90
    szamdb=5
else:                       # szam=2, 3 vagy 4, vagy más
    if szam==2:
        maxszam=45
        szamdb=6
    else:                           # szam= 3 vagy 4, vagy más
        if szam==3:
            maxszam=35
            szamdb=7
        else:                       # szam= 4, vagy más
            if szam==4:
                maxszam=80
                szamdb=20
            else:   # a szám  nem jó
                print(' a szam értéke csak 1,  2,  3  vagy 4 lehet !!')
...
```

2.9. Ciklus utasítás

Ciklus utasítás: ha többször ismételni akarunk utasításokat

- többféle is van ! → while, for

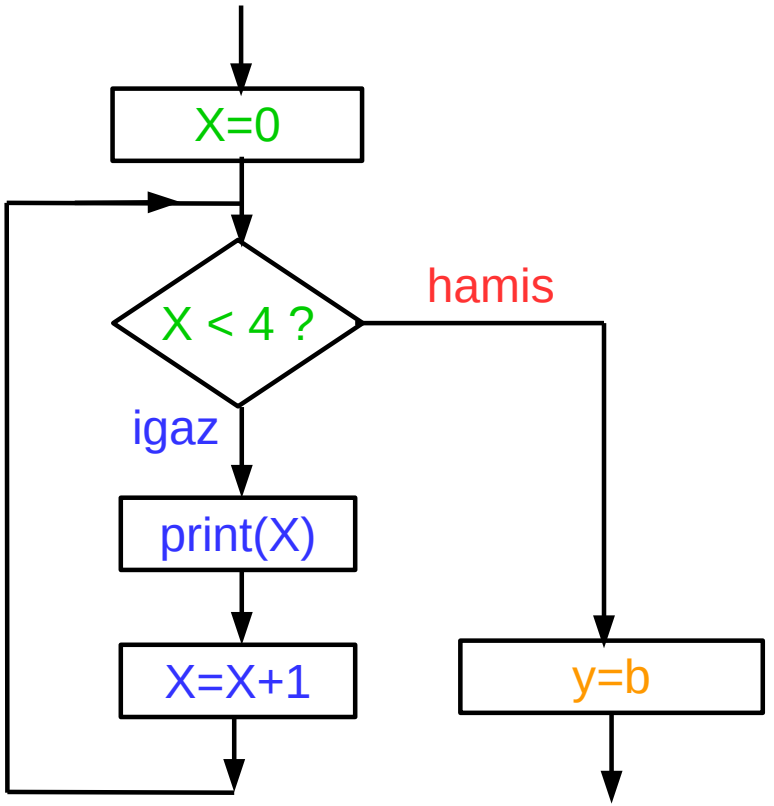
1. while ciklus

```
while feltétel:
        ismétlendő utasítások
```

amíg a feltétel igaz,
addig ismétli az utasításokat

pl.

```
...
x=0
while x<4:
    print(x)
    x=x+1
y=b
...
```



2.10. Ciklus utasítás

2. while működése

pl.

```
x=0
while x<4:
    x=x+1
    print(x)
```

Megfelel ennek:

```
x=0
x=x+1       #   x=0+1
print(x)    #   kiír → 1
x=x+1       #   x=1+1
print(x)    #   kiír → 2
x=x+1       #   x=2+1
print(x)    #   kiír → 3
x=x+1       #   x=3+1
print(x)    #   kiír → 4
```

van két speciális utasítás, amely módosítja a ciklusok lefutását!

break → kilépés a ciklusból !

continue → ugrás a ciklus elejére

2.11. Ciklus utasítás

3. for ciklus

```
for változó in szekvencia:
        ismétlendő utasítások
```

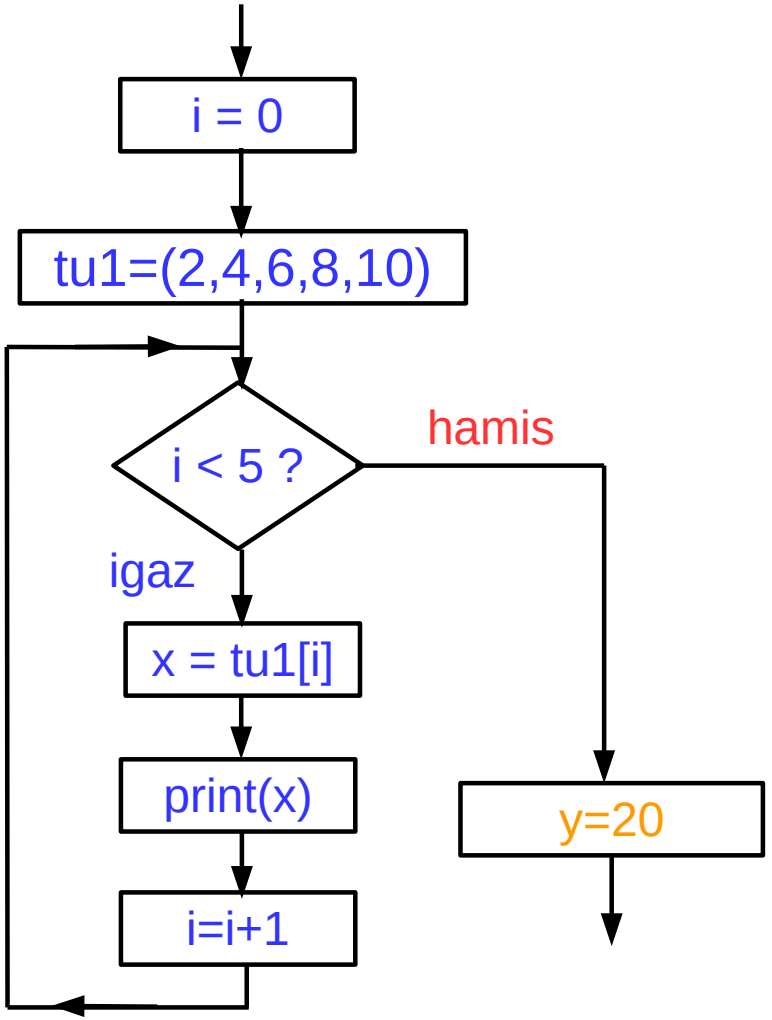
Végig lépked a szekvencián → a változó sorban felveszi a szekvencia elemeit

pl.

```
tu1=(2,4,6,8,10)
for x in tu1:
    print(x)
y=20
...
```

Képernyőn kiírva

2
4
6
8
10



2.12. Ciklus utasítás

2. mintafeladat

Írassuk ki az 1-10 egész számok négyzetének értékeit !
Megoldás for ciklussal

```
tu1=(1,2,3,4,5,6,7,8,9,10)
for x in tu1:
    li1= '%d négyzete = %d' % (x,x*x)
    print(li1)
```

Képernyőn kiírva

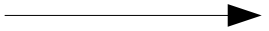
```
1 négyzete = 1
2 négyzete = 4
3 négyzete = 9
4 négyzete = 16
5 négyzete = 25
6 négyzete = 36
7 négyzete = 49
8 négyzete = 64
9 négyzete = 81
10 négyzete = 100
```

2.13. Ciklus utasítás

2. mintafeladat, másik megoldás

Írassuk ki az 1-10 egész számok négyzetének értékeit !
Megoldás while ciklussal

```
x=1
while x<11:
    li1= '%d négyzete = %d' % (x,x*x)
    print(li1)
    x=x+1
```



Képernyőn kiírva

```
1 négyzete = 1
2 négyzete = 4
3 négyzete = 9
4 négyzete = 16
5 négyzete = 25
6 négyzete = 36
7 négyzete = 49
8 négyzete = 64
9 négyzete = 81
10 négyzete = 100
```

2.14. Ciklus utasítás

3. mintafeladat

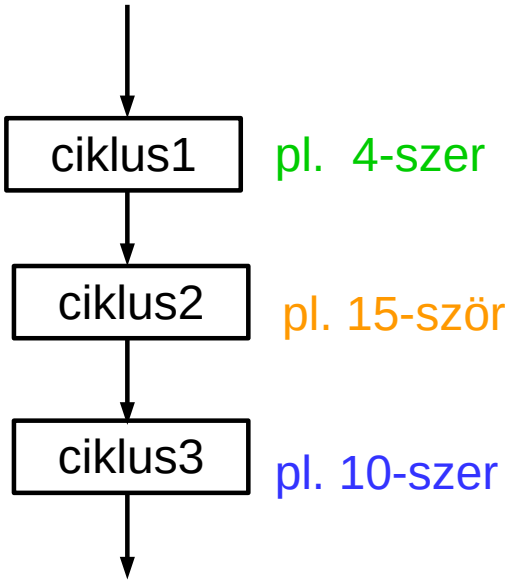
Adjuk össze az 1 és 101 közötti páros számokat
(tehát 2+4+6+8+……..+100)

```
szam=2              #  2 a legelső páros szám
osszeg=0     # ez tárolja a számok összegét
while szam<101:           #  100-ig megyünk
      osszeg=osszeg+szam        # a következő szám hozzáadása
      szam=szam+2     # a következő páros szám előállítása
st1= 'Az  1 és 101 közötti páros számok összege = %d' % (osszeg)
print(st1)
```

2.15. Több ciklus

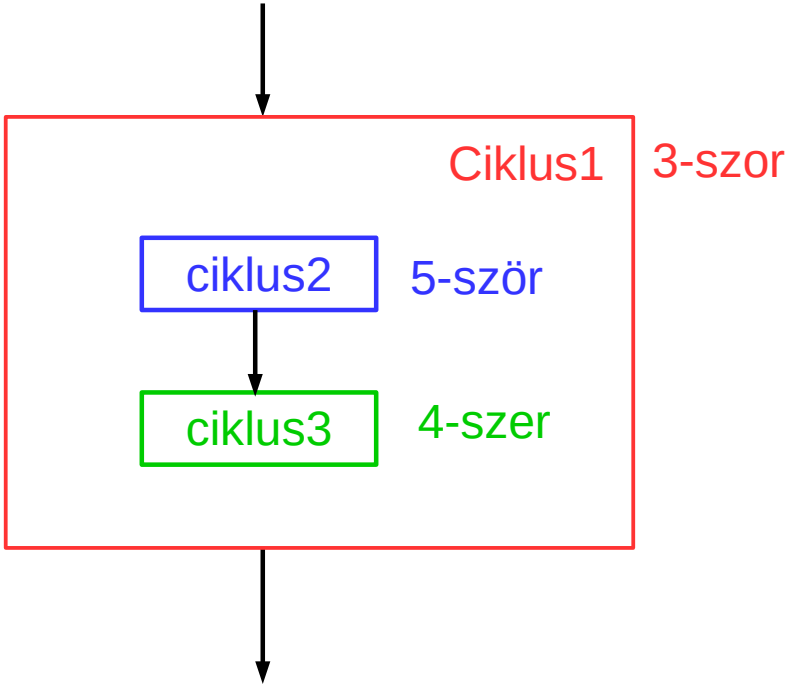
Több ciklus egymás után

- Akkor használunk ilyen szerkezeteket, ha egymás után különböző dolgokat kell ismételni,
- ciklusként természetesen akár for, akár while is használható, bármilyen kombinációban



Ciklus a ciklusban

- nemcsak egymás után lehetnek ciklusok, hanem egymáson belül is



3.1. Függvény

- Miért használunk függvényeket?

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

- Függvények

  - vannak előre definiált, beépített függvények:
    a Python nyelv könyvtári függvényei,
    + esetleg a használt fejlesztési környezet saját függvényei
    ezeket csak használni kell, pl. az eddigiekben már használt print() függvény
  - de természetesen létrehozhatunk saját függvényeket is

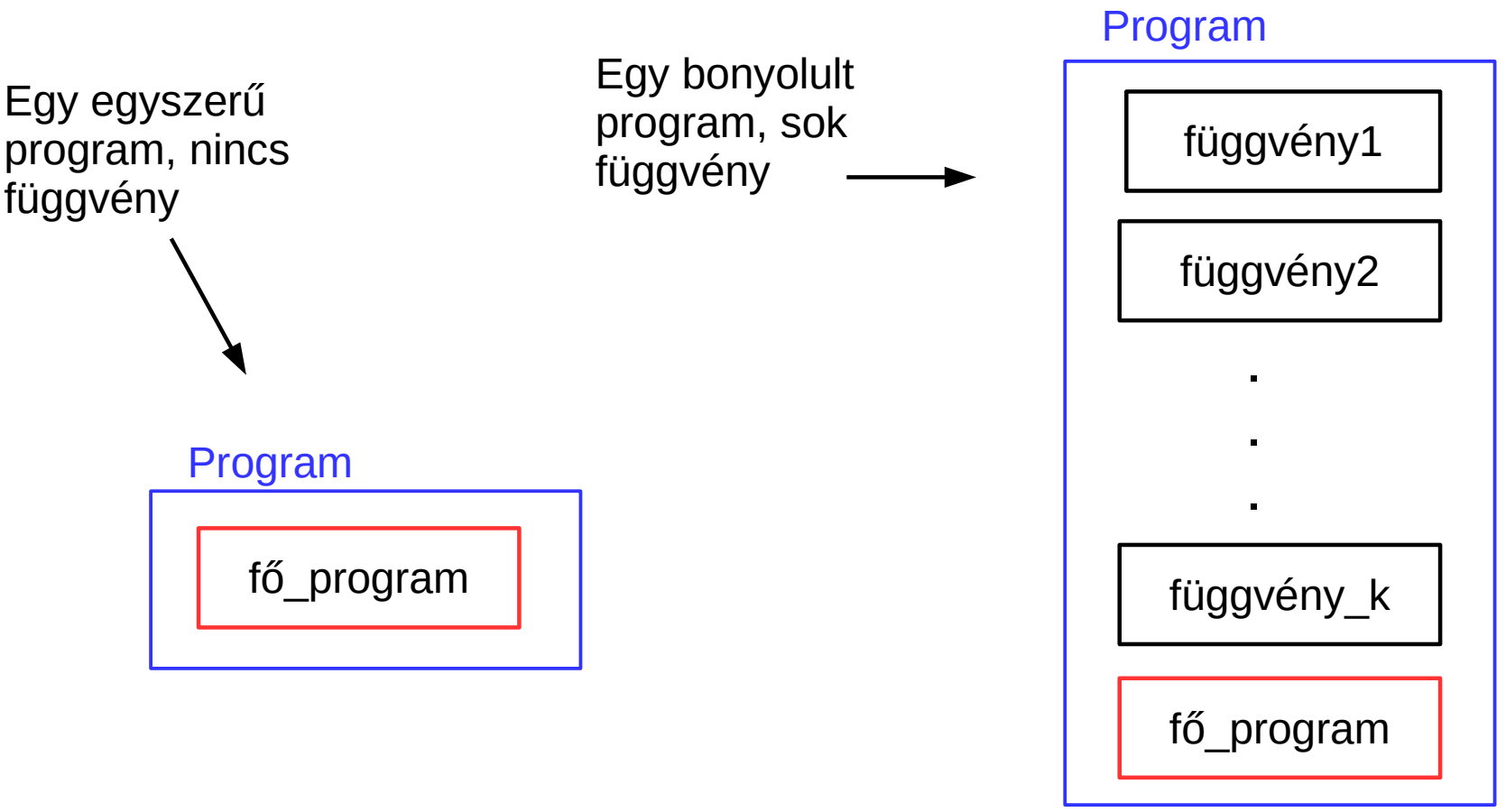
- Függvények típusai

  - valódi függvény (van visszatérési érték)
  - eljárás (procedure) → nincs visszatérési érték, „csak csinál valamit”
  - objektum metódus → objektumhoz tartozó függvény (később, OOP)

3.2. Függvény

- Egy Python nyelvű program szerkezete:

  - függvényekből (alprogramokból) állhat,
  - Python nyelven függvényen belül is lehet függvényt létrehozni !!



3.3. Függvény

- Saját függvények létrehozása

  - egy függvény fejrészét ('def' kulcsszó után a függvény neve, és zárójelek között a formális paraméterek) a függvény törzse követi

```
def  függvény_neve(paraméterek):        # fejrész
     'függvény leírása'
     utasítás1                          # a függvény törzse
     utasítás2
     ....
     return eredmény                    # eredmény vissza adása a hívónak
                                        # nem kötelező
```

- Kapcsolat egy függvény és a program többi része között

  - a függvény bemenete → a paraméterek (argumentumok) , segítségükkel tudunk adatokat átadni a függvénynek (amikor az aktuális paraméterekkel meghívjuk)
  - a függvény kimenete → a visszatérési érték, segítségével tudunk adatot visszaadni a hívó programrésznek → return érték

3.4. Függvény használata

- függvény létrehozása →

```
pl.  def fuggv1(par1,par2):      # par1, par2 → formális paraméterek
         print(par1+par2)
         print(par1*par2)
```

- függvény hívása → függvénynév(aktuális_paraméterek)

pl. fuggv1(10,30) → a függvény lefut, par1=10 és par2=30 értékekkel

- elképzelhető, hogy nincs paramétere a függvénynek
- ha van paramétere, akkor ugyanannyi darab (és ugyanolyan típusú) aktuális paramétert kell átadnunk mikor meghívjuk !
- argumentumként függvény hivatkozás is átadható (függvény név)

- érték visszaadása a return utasítással → pl. return 10
- ha return nincs → nem igazi függvény, csak alprogram, eljárás
- a visszatérési érték felhasználása → hozzárendelés egy változóhoz

pl. szam=atlag(10,36,60) → a függvény vissza adott értéke 'szam' változóba kerül

- sokszor meghívható egy függvény, más-más argumentumokkal → ez csökkenti a sorok számát (a függvényt csak egyszer kell leírni)

3.5. Függvény használata

- Függvény hívása, majd vissza:

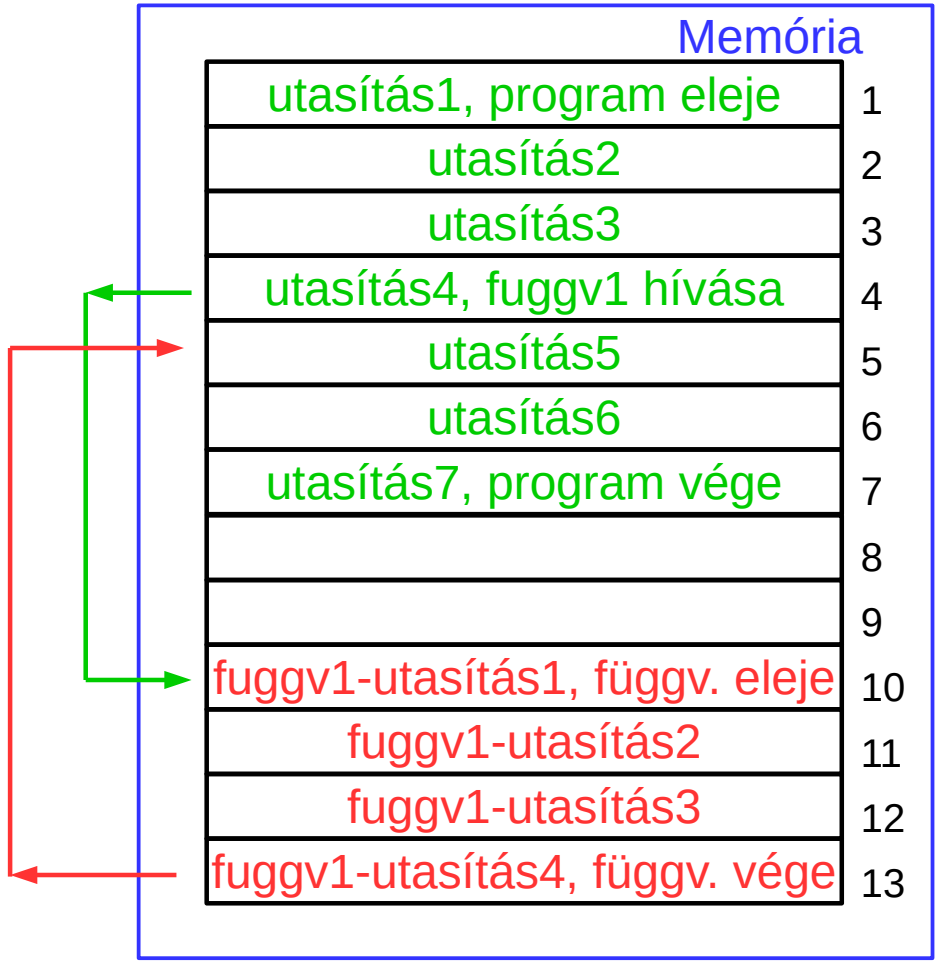
  - lényegében ugró utasítással történik ! (gépi kód szinten)
  - alapvetően az utasítások végrehajtása ugyanis egymás után történik, ahogyan a memóriában következnek egymás után (olyan sorrendben, ahogyan mi leírtuk az utasításokat)

  - függvény hívás esetén azonban nem a következő memória rekeszben lévő utasítás hajtódik végre, hanem az adott memória címre ugrunk, és ott folytatjuk tovább

fő_program

fuggv1_függvény

A program végrehajtás sorrendje ebben az esetben tehát (melyik számú memóriarekeszben lévő utasítás hajtódik végre)
→ 1-2-3-4-10-11-12-13-5-6-7



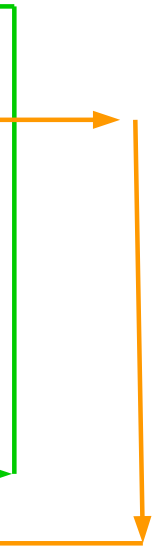
3.6. Függvény használata

1. mintafeladat

3 szám átlagát kiszámoló függvény

```
def atlag3(a, b, c):        # 5. meghívás → a=10, b=20, c=30
        atl=(a+b+c)/3       # 6. atl=60/3=20
        return atl          # 7. vissza a hívóhoz ! → 20 visszaadása

        # itt indul a program !!
sz1=10      # 1.
sz2=20      # 2.
sz3=30      # 3.
sz123=atlag3(sz1,sz2,sz3)  # 4. → atlag3 meghívása → ugrás
                           # 8. → sz123=20
st1= '%d és %d és %d átlaga: %d' % (sz1,sz2,sz3,sz123)
print(st1)
```



3.7. Függvény argumentumok

1. paraméterek típusa

- a dinamikus típus adás miatt → a függvény paraméterek típusa csak meghíváskor dől el (az aktuális paraméterek típusa határozza meg) !! → ez rugalmas, DE hibaforrás is lehet !! → mást kapunk, mint amire számítunk

2. mintafeladat

```
def kiir3szor(a):
        print(a)
        print(a)
        print(a)
# fő program
kiir3szor(5)
kiir3szor('OK')
```

Képernyőn kiírva

```
5
5
5
OK
OK
OK
```

3.8. Függvény argumentumok

2. paraméterek alapértelmezett értéke

- a függvény paramétereknek adhatunk alapértelmezett értékeket → híváskor nem kötelező ezeket megadni
- az alapértelmezett értékek nélküli paramétereknek kell elől állniuk !!

3. mintafeladat

```
def szorzotabl(alap,tol=1,ig=10):
    n=tol
    while n<=ig:
        szorz=n*alap
        str = '%d x %d = %d' % (n,alap,szorz)
        print(str)
        n=n+1

# fő program
szorzotabl(4,3,5)
szorzotabl(6,8)
szorzotabl(2)
```

Képernyőn kiírva

```
3 x 4 = 12
4 x 4 = 16
5 x 4 = 20
8 x 6 = 48
9 x 6 = 54
10 x 6 = 60
1 x 2 = 2
2 x 2 = 4
3 x 2 = 6
4 x 2 = 8
5 x 2 = 10
6 x 2 = 12
7 x 2 = 14
8 x 2 = 16
9 x 2 = 18
10 x 2 = 20
```

3.9. Függvény argumentumok

3. kulcsszavas argumentumok

- ha mindegyik paramétereknek adtunk alapértelmezett értékeket →
  híváskor ha a paraméterek neveit megadjuk
  → sorrendjük bármilyen lehet !

4. mintafeladat

```
def szorzotabl2(alap=2,tol=1,ig=10):
    n=tol
    while n<=ig:
        szorz=n*alap
        str = '%d x %d = %d' % (n,alap,szorz)
        print(str)
        n=n+1

# fő program
szorzotabl2(tol=4,ig=8)
szorzotabl2(ig=5,alap=3)
```

Képernyőn kiírva

```
4 x 2 = 8
5 x 2 = 10
6 x 2 = 12
7 x 2 = 14
8 x 2 = 16
1 x 3 = 3
2 x 3 = 6
3 x 3 = 9
4 x 3 = 12
5 x 3 = 15
```

3.10. Függvény argumentumok

4. változó hosszúságú argumentumlisták

nem kulcsszavas

```
def fuggv_nev(arg1, arg2, …, *args):
    ...
```

Változó hosszúságú tuple → ebbe kerülnek a pluszban megadott paraméterek

kulcsszavas

```
def fuggv_nev(arg1, arg2, …, *args, **kwargs):
    ...
```

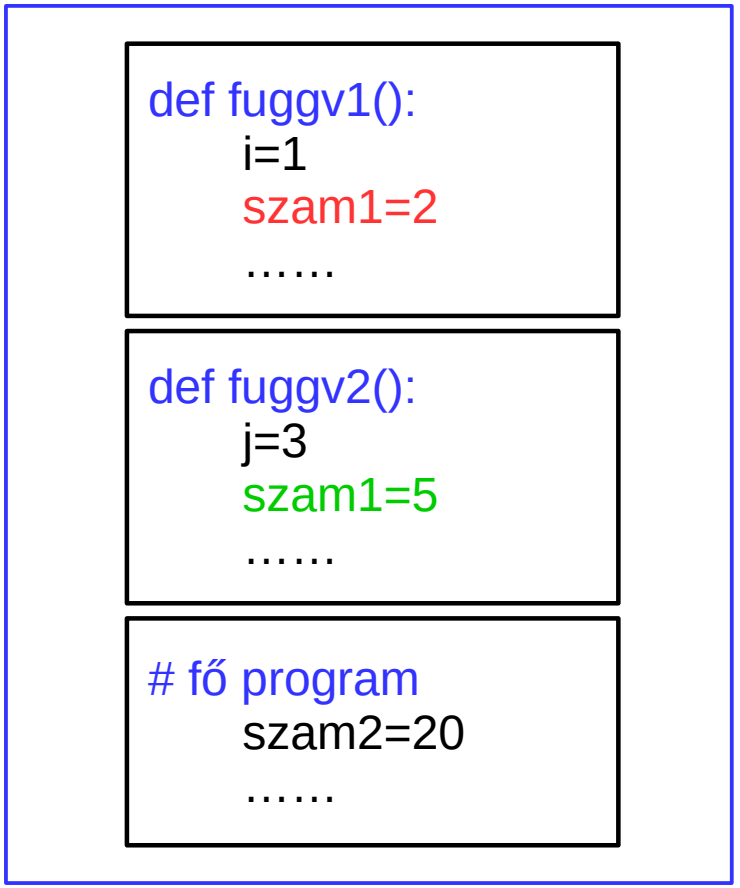
Szótárba kerülnek → a végén még megadott kulcsszavas paraméterek

3.11. Helyi és globális változók

- Helyi változók

  - A függvényen belül definiált változók → a függvény helyi (saját) lokális változói → más függvények ezeket nem érik el !!
  - különböző függvényeknek lehetnek azonos nevű helyi változói → de ezen változók teljesen függetlenek egymástól !!

  (hasonlóan mint egy Miskolcon lévő Petőfi utcának semmi köze egy Debrecenben lévő Petőfi utcához)



szam1 és szam1 változó 2 különböző változó !! → semmi közük egymáshoz

fuggv1 számára 'j' változó nem létezik

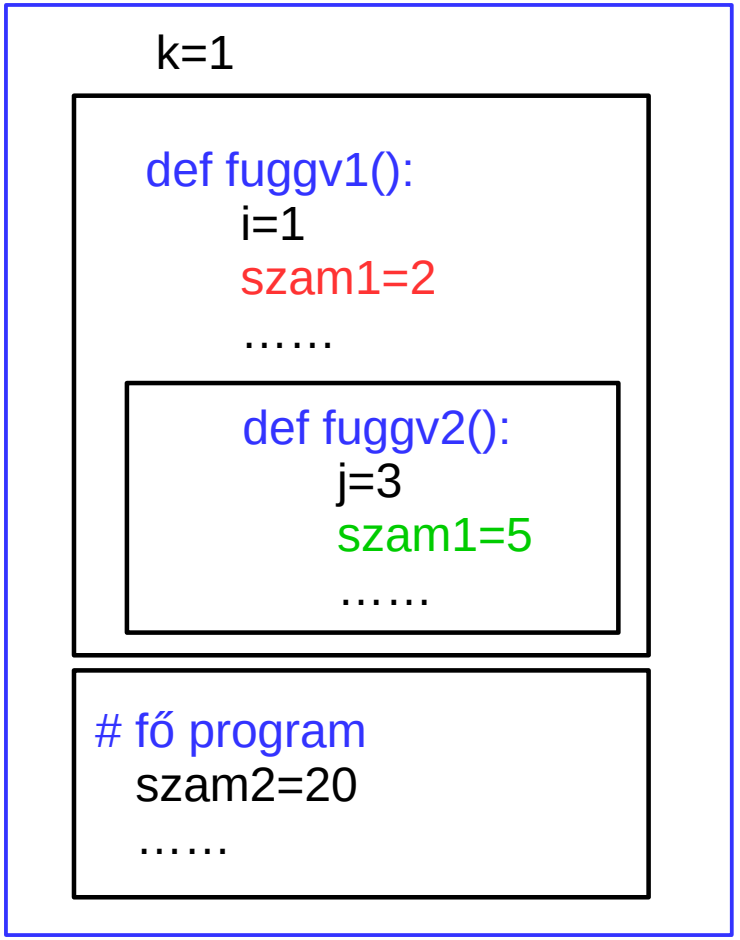
fuggv2 számára 'i' változó nem létezik

Fő program számára 'j' és 'i' változó nem létezik

3.12. Helyi és globális változók

- Globális változók

  - A függvényeken kívül definiált változók → a program globális változói
    → minden függvény (és a fő program) eléri ezeket !!
  - DE az ugyanolyan nevű lokális változó a függvényen belül „elfedi” a globális változót !! → függvényen belül a globális változó simán nem módosítható ! → csak külön definíció után → global változó_név



'k' és 'szam2' globális változók

fuggv2 számára 'i' változó globális, fuggv1 számára lokális, a főprogram számára nem létezik

'szam1' fuggv1 számára lokális, a főprogram számára nem létezik. Viszont fuggv2 számára globális lenne, ha nem lenne saját, lokális 'szam1' változója (ez elfedi 'szam1' változót !)→ tehát szam1 és szam1 változó két különböző változó lesz most is

'j' változó fuggv2 lokális változója, a fő program és fuggv1 számára nem létezik

3.13. Helyi és globális változók

5. mintafeladat

```
def atlag3(a, b, c):
        atl=(a+b+c)/3
        print(atl)
        print(szam)
        return atl

# fő program !!
atl=10
szam=60
sz123=atlag3(10,20,30)      #   atlag3 meghívása
print(atl)
print(szam)
```

'atl' és 'szam' globális változók DE atlag3 függvényben van helyi 'atl' változó is !! → a globális változót elfedi

Képernyőn kiírva

```
20.0
60
10
60
```

3.14. Lambda formák

- kisméretű, névtelen függvények
- paraméterük sok lehet, de csak egy értéket adhatnak vissza →
  kifejezés formájában !
- mivel nevük nincs → közvetlenül nem hívhatók meg →
  általában akkor használjuk őket ha függvény argumentumra van szükség
- lambda forma létrehozása →

```
lambda par1,par2, ...parx : kifejezés
        # a kifejezés értéke lesz a visszatérési érték !
```

pl. `lambda a,b : a*b`

- hozzárendelhetők változóhoz ! → ekkor meghívhatók →

pl. `x=lambda a,b : a*b`

hívása → y=x(10,30) → a lambda függvény lefut (a=10 és b=30) → y=300

- az argumentum tuple és szótár is lehet

s=lambda *tupl : kifejezés → pl. s=lambda *t1 : 2*t1

z=lambda **szot : kifejezés

3.15. Beépített függvények

előre definiált, a nyelvbe beépített függvények
nagyon sok létezik, érdemes kategorizálni őket →

- általános jellegű, valamilyen fontos funkciót ellátó függvények
- számokkal kapcsolatos függvények
- sztringekkel kapcsolatos függvények
- sorozatokkal (lista, tuple)…
- fájl műveletekkel….
- ...

<u>általános jellegű beépített függvények</u>

print( )

- képernyőre íratja ki a kapott paramétert

type( )

- egy változó típusát adja vissza

pl. x='egy szoveg'
y=25.6
print type(x) → < type 'str' >
print type(y) → < type float' >

3.16. Beépített függvények

általános jellegű beépített függvények

input( )

- beolvasás billentyűzetről (a vége → 'enter')

pl. nev = input('Írja be a nevét: ')

mindig karakterláncot ad vissza ! → nekünk kell átalakítani a kívánt formára

!! A 2-es Pythonban olyan típust adott vissz, amit a felhasználó beírt !!
ott volt egy másik függvény is → **raw_input( )**
→ az adott vissza minden esetben karakterláncot

apply(fv_név,par1,par2,... **)**

- egy függvényt lehet vele meghívni (ha futás közben dől el hogy melyiket)

id( )

- egy változó címét adja vissza

pl. y=(23,5,78,124,3,45,7)
print(id(y))

3.17. Beépített függvények

általános jellegű beépített függvények

comp(obj1,obj2**)**
- két objektum összehasonlítása

min( )
- egy sorozat legkisebb elemét adja vissza

pl.
```
y=(23,5,78,124,3,45,7)
print(min(y))          → 3
```

max( )
- egy sorozat legnagyobb elemét adja vissza

pl.
```
x='egy szoveg'
print(max(x))          → z
print(max(y))          → 124
```

sum( )
- egy számsorozat (tuple vagy list) öszegét adja vissza

pl.
```
y=(1,2,3,4,5)
print(sum(y))          → 15
```

3.18. Szám típusok beépített függvényei

int( )

- egész számmá alakít (számot vagy sztringet)

pl. szam = int('13')
szam2 = int(24.5) → 24

float( )

- lebegőpontos számmá alakít (számot vagy sztringet)

pl. szam3 = float('13.5')

complex(sztring**)** vagy **complex(**val,kep**)**

- komplex számmá alakít

pl. komp1 = complex('13+5j')
komp2 = complex(13,5)

coerce(obj1,obj2**)**

- azonos numerikus típussá konvertálja őket → tupleként adja vissza

pl. tup1 = coerce(13.5,5)
print(tup1) → (13.5, 5.0)

3.19. Szám típusok beépített függvényei

pow(alap,kitevő**)** vagy **pow(**alap,kitevő,mod**)**

- hatványozás (+ osztás maradéka !)

pl. szam1 = pow(3,4) # 3^4 = 9*9

szam2 = pow(3,4,2) # 3^4 % 2 = 81 % 2 = 1

print(szam1) → 81

print(szam2) → 1

divmod(obj1,obj2**)**

- egész osztás + maradékos osztás → tupleként adja vissza

pl. tup1 = divmod(13,2) # 13 / 2 = 6 és 13 % 2 = 1

print(tup1) → (6, 1)

round(valós,tizedesjegyek**)**

- lebegőpontos számot kerekít

pl. szam3 = round(13.3456723,4)

print(szam3) → 13.3457

abs( )

- abszolút értéket ad vissza

pl. print(abs(-13)) → 13

3.20. Szám típusok beépített függvényei

hex(egész **)**

- hexa konverzió (egész számot)

pl. print(hex(43)) → 0x2b

bin(egész **)**

- bináris konverzió (egész számot)

pl. print(bin(43)) → 0b101011

oct(egész **)**

- oktális konverzió (egész számot)

pl. print(oct(43)) → 053

ord(karakter**)**

- karakter-egész konverzió (ASCII kód)

pl. print(ord('B')) → 66

chr(kód**)**

- ASCII kód - karakter konverzió

pl. print(chr(65)) → 'A'

3.21. Szekvenciák, szótárak beépített függvényei

len( )

- sztring, lista, tuple, szótár méretét (elem számát) adja vissza

pl.
```
x='egy szoveg'
y=(23,5,78,124,3,45,7)
print(len(x))        → 10
print(len(y))        → 7
```

range(ig**)** vagy **range(**tól,ig**)** vagy **range(**tól,ig,lépés**)**

- számokból álló listaszerű valami létrehozása (2-s pythonban lista volt)

pl.
```
li1 = range(5)       → 0,1,2,3,4
li2 = range(3,9)     → 3,4,5,6,7,8
li3 = range(3,13,2)  → 3,5,7,9,11
```

str( )

- sztringgé alakít

pl.
```
sztr1 = str(13.56)   → '13.56'
```

repr( )

- sztringgé alakít

pl.
```
sztr1 = repr(13.56)   → '13.56'
```

3.22. Szekvenciák, szótárak beépített függvényei

filter(logikai_függv, sorozat_be**)**

- a bemenő sorozatot egy logikai függvény alapján szűri →
  a szűrt sorozatot adja vissza

DE ! 3-as Pythonban objektumot ad vissza ! → át kell alakítani !

```
# 6. mintafeladat
def nagyobb5(a):
    if a>5:
        return 1   #  igaz, ha az átadott szám > 5
    else:
        return 0      #  hamis, ha az átadott szám <= 5

# fő program !!
li1=[0,1,2,3,4,5,6,7,8,9]    #    lista létrehozása
li2 = filter(nagyobb5,li1)    #    li1 szűrése
li3 = list(li2)    #  lista típusúvá konvertálás
print(li1)
print(li3)
```

Képernyőn kiírva

```
[0,1,2,3,4,5,6,7,8,9]
[6,7,8,9]
```

3.23. Szekvenciák, szótárak beépített függvényei

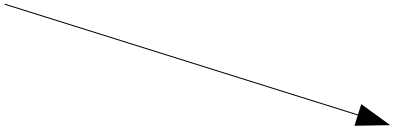
map(függvény, sorozat_be**)**

-leképzés → a bemenő sorozat minden elemén végrehajtja ugyanazt a műveletet → a számított sorozatot adja vissza

DE ! 3-as Pythonban objektumot ad vissza ! → át kell alakítani !

```
# 7. mintafeladat
#   négyzetre emelés

li1= [0,1,2,3,4,5,6,7,8,9]   #    lista létrehozása
li2 = (map((lambda a : a*a),li1)
li3=list(li2)
print(li1)
print(li3)
```



Képernyőn kiírva

[0,1,2,3,4,5,6,7,8,9]
[0,1,4,9,16,25,36,49,64,81]

3.24. Szekvenciák, szótárak beépített függvényei

map(függvény, sorozat1,sorozat2**)**

-leképzés → lehet több bemenő sorozat is !→ több argumentumú függvény kell

```
# 8. mintafeladat
#   két sorozat összeadása

li1=[0,1,2,3,4,5]
li2=[4,7,9,11,13,15]
li3 = list(map((lambda a,b : a+b),li1,li2))
print(li3)
```

Képernyőn kiírva

[4,8,11,14,17,20]