22.) PROGRAMOZÁSI TÉTELEK ALKALMAZÁSA A PYTHONBAN

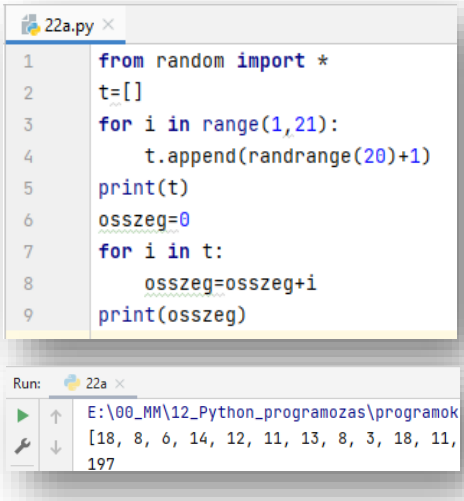
Ebben a fejezetben fogunk olyan dolgokkal találkozni, amelyeket már tanultunk, ezért felfoghatjuk ismétlésnek. És természetesen elő fog fordulni új anyag is.
Ebben a tananyagban az úgynevezett „programozási tételek”-et nézzük át!
Melyek az:

• Összegzés	• Kiválasztás	• Szélsőérték meghatározása
• Megszámolás	• Másolás	• Metszet
• Eldöntés	• Kiválogatás	• Unió
• Keresés	• Szétválogatás	• Rendezések

Tehát a következő „programozási tételek” a feladatok során legtöbbször előforduló algoritmusokat tartalmazza. Mindegyik „tételhez” tartozni fog egy rövid magyarázat, az általános mondatszerű leírás egy szövegdobozban, és maga a python példa kódja és képernyőképe. Legtöbbször látjuk majd, hogy a python-ban sokkal egyszerűbben meg tudjuk oldani a kódolást.
A feladatok megoldásánál (mondatszerű leírásnál és a kódírásnál is) tömböket fogunk használni, a t[ ] tömbnek pedig n darab eleme lesz. A tömbök elemeinek indexe 0-tól n-1-ig fog menni. Ahol több tömbbel dolgozunk ott az első tömb az a[ ], amelynek n eleme van, a második tömb a b[ ], amelynek m elem van. Harmadik tömb neve pedig a c[ ]. Az a[ ] tömb ciklus változója szokásosan i, a b[ ] tömbbé j, a harmadik c[ ] tömbbé k. De ahol szükséges használjuk az x,y,z változókat is.
A mondatszerű leírásnál is a tömbök indexelését 0-val kezdjük. Az értékadást egy darab egyenlőségjel (=) jelenti, az egyenlőség vizsgálatot két egyenlőségjel (==) jelzi, a nem egyenlőt egy kisebb és egy nagyobb jel jelzi (<>).
A python program kód írásánál a tömbökbe véletlen számokat fogunk generálni és azokkal fogunk dolgozni.

(22a.py)
Összegzés
Magyarázat:
Az egyik legegyszerűbb „programozási tétel” az összegzés.
Adott egy 20 elemű tömb, melyet feltöltünk 1-20 –ig vél. számokkal.
Majd egy for ciklussal végig megyünk a tömb elemein és egy „osszeg” nevű változóba mindig hozzáadjuk az aktuális értéket.
Tehát egy **tömb elemértékeinek összegét** számoljuk ki.
Mondatszerű leírás:

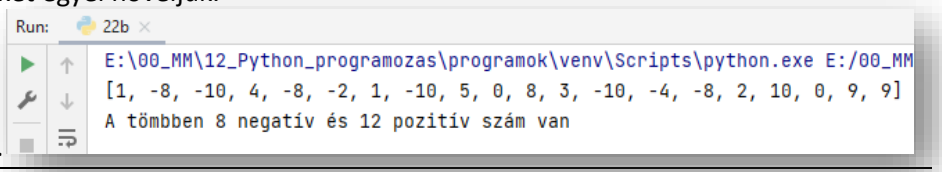
```
osszeg = 0
ciklus i = 0 .. n -1
    osszeg = osszeg + t[i]
ciklus vége
ki osszeg
```



(22b.py)
Megszámolás
Magyarázat:
Adott feltételek alapján a **tömb bizonyos elemeit megszámoljuk**.
Pl.: Megszámoljuk mennyi negatív és pozitív szám van a tömbben.
Létrehozunk egy tömböt, amit feltöltünk -10-től +10-ig számokkal.
Majd indítunk egy for ciklust, végig a tömbön, ahol egy if feltétellel megvizsgáljuk hogy az adott érték kisebb-e mint 0. Mert ha kisebb akkor a „negativ” változó értékét egyel növeljük.
Különben a „pozitiv” változó értékét növeljük egyel.
Végül kiíratjuk a minta szerint.

Mondatszerű leírás:

```
szamlalo = 0
ciklus i = 0 .. n - 1
   ha t[i] < 0 akkor
      szamlalo = szamlalo + 1
   ha vége
ciklus vége
ki szamlalo
```



A megszámolás python kódja:

```python
from random import *
t=[]
negativ=0
pozitiv=0
for i in range(1,21):
    t.append(randrange(21)-10)
print(t)
for i in t:
    if i<0:
        negativ += 1
    else:
        pozitiv += 1
print('A tömbben %d negatív és %d pozitív szám van' % (negativ,pozitiv))
```

(22c.py)

Eldöntés

Magyarázat: *Mondatszerű leírás:*

Az eldöntés esetében azt vizsgáljuk, hogy **szerepel-e egy bizonyos tulajdonságú elem az adatsorban**. A válasz igen/nem (True vagy False lehet.)

Létrehozunk egy tömböt. Amit feltöltünk 1-20-ig terjedő véletlen számokkal. Majd „talalt” néven létrehozunk egy változót, amit „False”-ra állítunk.

Aztán indítunk egy „while” ciklust, ami mindaddig fut, amig nem talál egy olyan számot, amit ha elosztunk hárommal nullát nem kapunk eredményül.

Mert, ha talált ilyet, akkor a „találat” változót igazra állítjuk. Közben egy „i” változót felhasználva léptetjük a tömb indexét. Ami egyébként fontos lesz a keresésnél.

A listát bejárhattuk volna egy for ciklus segítségével, de felesleges végighaladnunk az összes elemen.

A fenti program amint megtatálja az első olyan elemet, amely megfelel a feltételeknek, befejezi a lista átvizsgálását.

Hiszen ha már találtunk egy ilyen elemet, megvan a válasz.

```
van = 0
ciklus i = 0 .. n-1
  ha tomb[i] = ker_ertek akkor
     van = 1
  ha vége
ciklus vége
ki van
```

```python
from random import *
t=[]
for i in range(1,11):
    t.append(randrange(21))
print(t)
talalat = False
i=0
while i < len(t) and not talalat:
    if t[i] % 3 == 0:
        talalat = True
    i= i + 1
if talalat:
    print('Van a listában hárommal osztható szám.')
else:
    print('Nincs a listában hárommal osztható szám.')
```

```
E:\00_MM\12_Python_programozas\programo
[19, 7, 7, 12, 18, 16, 8, 18, 14, 8]
Van a listában hárommal osztható szám.
```

(22d.py)

Keresés

Magyarázat: *Mondatszerű leírás:*

Most ugyanazt vizsgáljuk, mint az előbb, csak még azt is keressük, hogy a **megtalált elem hányadik helyen van**.

Annyi dolgunk van, hogy elmentjük az előző programot másként és kiegészítjük egyetlen sorral. Mivel az előbb az „i” változóval lépkedünk előre egyesével, itt csak fel kell használni a kiíratásnál.

```
van = 0
ciklus i = 0 .. n-1
  ha tomb[i] = ker_ertek akkor
     van = 1
  ha vége
  i = i+1
ciklus vége
ki tomb[i]
```

```python
if talalat:
    print('Van a listában hárommal osztható szám.')
    print('Az első szám a listából, ami osztható hárommal: ',t[i-1])
```

```
E:\00_MM\12_Python_programozas\programok\venv\Script
[9, 18, 9, 16, 1, 16, 11, 5, 18, 13]
Van a listában hárommal osztható szám.
Az első szám a listából, ami osztható hárommal:  9
```

(22e.py)

Kiválasztás

Magyarázat:

A kiválasztás tételt akkor használjuk, ha **tudjuk, hogy a keresett értéket tartalmazza a tömb**. Ezért azt nem vizsgáljuk, hogy vége van-e a tömbnek. Ezért most fixen feltöltünk 1 és 20 közötti számokkal egy „t” tömböt, majd addig megyünk egy ciklussal az elemeken végig, amíg meg nem találja.
Végül kiíratjuk, hogy **hányadik elem**.

```
t=[7,15,3,6,11,18,14,7,4,9,12,13,17]
i=0
while t[i] != 12:
    i += 1
print('A 12-es szám a(z) %d -dik elem a lisában.' % (i+1))
```

```
Run: 22e
E:\00_MM\12_Python_programozas\programok\ve
A 12-es szám a(z) 11 -dik elem a lisában.
```

Mondatszerű leírás:

```
i = 0
ciklus amíg tomb[i] <> keresett
   i = i + 1
ciklus vége
ki i + 1
```

(22f.py)

Másolás

Magyarázat:

Ennél a hasznos „programozási tétel”-nél **egy sorozat elemeit átmásolom egy másik sorozatba, miközben valamilyen átalakítást végzek az egyes elemeken**.
Ebben a példában egy „a” nevű tömbben felsorolt számokat duplázzuk meg egy „b” nevű listába.

Mondatszerű leírás:

```
ciklus i = 1 .. n
   b[i] = művelet(a[i])
ciklus vége
```

```
a=[7,15,3,6,11,18,14,7,4,9,12,13,17]
b=[]
i=0
for i in range(len(a)):
    b.append(a[i]*2)
print(a)
print(b)
```

```
Run: 22f
E:\00_MM\12_Python_programozas\programok\venv\Scrip
[7, 15, 3, 6, 11, 18, 14, 7, 4, 9, 12, 13, 17]
[14, 30, 6, 12, 22, 36, 28, 14, 8, 18, 24, 26, 34]
```

(22g.py)

Kiválogatás

Magyarázat:

Az egyik tömb elemeit **átteszem egy másik tömbbe, ha megfelelnek egy adott feltételnek**. Tehát kiválogatom valamilyen feltétel szerint.
Adott a és b tömb. Az a tömb egész számokat tartalmaz 1 és 20 között. Az a tömbből a 10, vagy annál kisebb számokat átrakom b tömbbe.

Mondatszerű leírás:

```
j = 0
ciklus i = 0 .. n - 1
   ha a[i] <= 10
      b[j] = a[i]
      j = j + 1
   ha vége
ciklus vége
```

```
a=[7,15,3,6,11,18,14,7,4,9,12,13,17]
b=[]
i=0
for i in range(len(a)):
    if a[i] <= 10:
        b.append(a[i])
print(a)
print(b)
```

```
Run: 22g
E:\00_MM\12_Python_programozas\programok\venv\S
[7, 15, 3, 6, 11, 18, 14, 7, 4, 9, 12, 13, 17]
[7, 3, 6, 7, 4, 9]
```

(22h.py)

Szétválogatás

Magyarázat:

Két tömbbe válogatjuk szét egy tömb elemeit. Adott „a” tömb, amely egész számokat tartalmaz 1 és 20 között. A „b” és „c” tömb pedig üres. Az „a” elemeit „b” tömbbe rakjuk ha egyenlők vagy kisebbek 10-él, különben c-ben tároljuk.

Mondatszerű leírás:

```
j = 0
k = 0
ciklus i = 0 .. n-1
   ha a[i] < 5
      b[j] = a[i]
      j = j + 1
   különben
      c[k] = a[i]
      k = k + 1
   ha vége
ciklus vége
```

```python
a=[7,15,3,6,11,18,14,7,4,9,12,13,17]
b=[]
c=[]
i=0
for i in range(len(a)):
    if a[i] <= 10:
        b.append(a[i])
    else:
        c.append(a[i])
print(a)
print(b)
print(c)
```

```
E:\00_MM\12_Python_programozas\programok\venv\Sc
[7, 15, 3, 6, 11, 18, 14, 7, 4, 9, 12, 13, 17]
[7, 3, 6, 7, 4, 9]
[15, 11, 18, 14, 12, 13, 17]
```

(22i.py)

Szélsőérték meghatározása

Mondatszerű leírás:

```
min=t[0]
ciklus i = 1 .. n-1
   ha t[i] < min akkor
      min = t[i]
   ha vége
ciklus vége
ki min
```

```
max = t[0]
ciklus i = 1 .. n - 1
   ha t[i]> max akkor
      max = t[i]
   ha vége
ciklus vége
ki max
```

Magyarázat:

Tulajdonképpen ez a **legkisebb és/vagy legnagyobb érték kiválasztása egy listából**.

A példában generáljuk 10 db véletlen számot 1 és 100 között.

Majd keressük ki a minimum és a maximum értékeket a tömbből.

Ennél a „tételnél” arra kell figyelni, hogy a program elején a minimum értéket az előfordulható legnagyobbra, a maximum értéket az előfordulható legkisebbre kell állítani, mert az első összehasonlításnál már cserélődni kell az értékeknek (példa).

Vagy egy másik megoldásként a minimum és maximum értékeket a tömb első elemére állítjuk be (mondatszerű leírás).

```python
from random import *
t=[]
for i in range(1,11):
    t.append(randrange(100)+1)
print(t)
min=100
max=0
for i in range(len(t)):
    if t[i] < min:
        min = t[i]
for i in range(len(t)):
    if t[i] > max:
        max = t[i]
print('A legkisebb elem: ',min)
print('A legnagyobb elem: ',max)
```

```
E:\00_MM\12_Python_programozas\programok
[47, 25, 62, 71, 33, 35, 2, 10, 13, 81]
A legkisebb elem:  2
A legnagyobb elem:  81
```

(22j.py)

Metszet

Magyarázat: *Mondatszerű leírás:*

Két tömb azonos elemeinek kiválogatása egy harmadik tömbbe.

Nem olyan régen tanultuk a halmazokat. Ott két halm. metszetét az „print(a & b) paranccsal oldottuk meg. Itt ez nem működik, mert most egy külön listába szeretném eltárolni a közös értékeket.

Ezért két egymásba ágyazott ciklussal kell megoldani a feladatot.

Egyesével megyünk végig az „a” tömbön azon belül pedig megvizsgáljuk a „b” összes elemével, hogy van-e egyezés. Mert ha igen, akkor hozzáadjuk a „c” listához.

Értelmezzük a programot!

```
k = 0
ciklus i = 0 .. n-1
   j = 0
   ciklus amíg j<m és b[j]<>a[i]
      j = j + 1
   ciklus vége
   ha j<m akkor
      c[k] = a[i]
      k = k + 1
   ha vége
ciklus vége
```

```python
a=[12,9,7,15,13,5,8,11,19,3,1,10]
b=[2,4,6,8,10,12,14,16,17,18,20]
c=[]
i=0
for i in range(len(a)):
    j=0
    for j in range(len(b)):
        if a[i]==b[j]:
            c.append(a[i])
print(a)
print(b)
print(c)
```

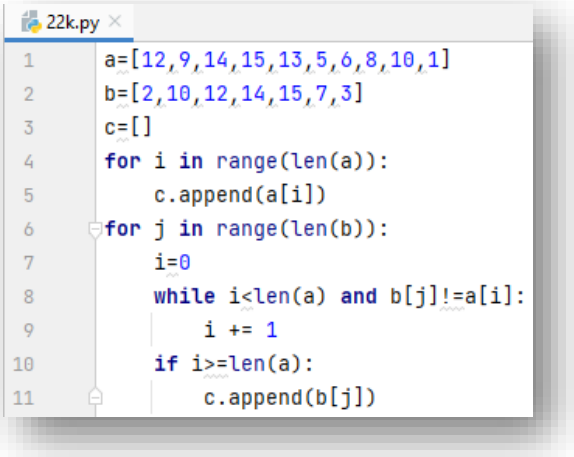
```
E:\00_MM\12_Python_programozas\programok\ve
[12, 9, 7, 15, 13, 5, 8, 11, 19, 3, 1, 10]
[2, 4, 6, 8, 10, 12, 14, 16, 17, 18, 20]
[12, 8, 10]
```

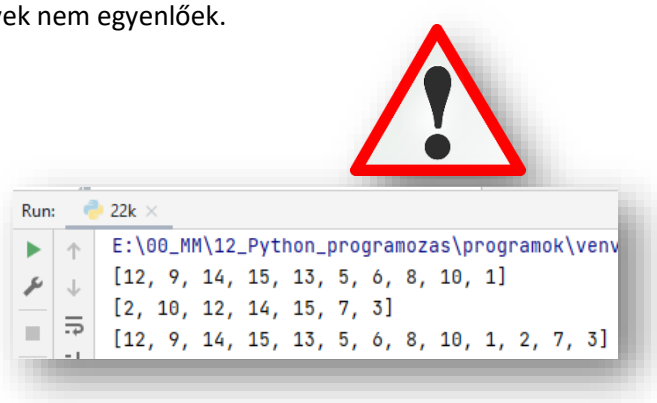
(22k.py)
Unió

Magyarázat: *Mondatszerű leírás:*

A és B tömb minden elemét szeretnénk C tömbbe tenni.
Először C-be tesszük az A összes elemét, majd ami nem szerepel A-ban, azt beletesszük C-be.
A mondatszerű leírás átalakítása python nyelvre mindig kihívás.
Először generálunk „a” és „b” tömböt 1 ás 20 közötti számokkal!
Létrehozunk egy üres „c” tömböt.
Aztán beletesszük egy for ciklussal az „a” tömb tartalmát a „c” tömbbe.
Uniónál arra kell figyelni, hogy metszet számai ne szerepeljenek kétszer a „c” tömbben.
Ezért a „b” tömb elemein egyesével végig megyünk egy hátultesztelő ciklussal és csak azokat a számokat adjuk hozzá a „c” tömbhöz, amelyek nem egyenlőek.

```
ciklus i = 0 .. n-1
   c[i] = a[i]
ciklus vége
k = n
ciklus j = 0 .. m-1
   i = 0
   ciklus amíg i<n és b[j]<>a[i]
      i = i + 1
   ciklus vége
   ha i>=n akkor
      c[k] = b[j]
      k = k + 1
   ha vége
ciklus vége
```

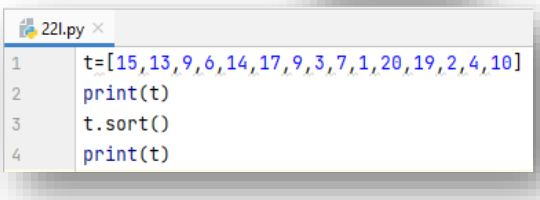


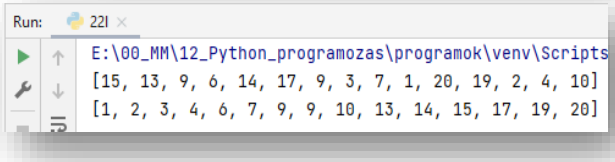


(22l.py)
Rendezések

Magyarázat:

A programozásban az egyik legfontosabb művelet a rendezés.
Amikor **egy rendezetlen tömb elemeit növekvő sorba rendezzük**.
Ennek a megoldására számtalan „programozási tétel” áll rendelkezésünkre.
A python nyelven ezt a problémát nagyon gyorsan és egyszerűen megoldhatjuk a short() metódussal.





Ezek a különböző módszerek a gyorsaság, bonyolultság és memória igény függvényében változhatnak.
Példák a **rendezés típusai**ra:

- **Buborékrendezés**
- **Rendezés cserével**
- **Rendezés maximum kiválasztással**
- **Minimum kiválasztásos rendezés**
- **Rendezés beszúrással**
- **Shell rendezés**
- **Gyorsrendezés**
- **Összefésüléses rendezés**

Ezeket a rendezéses „tételeket”, és azok mondatszerű leírását érdemes megtanulni, megérteni, mert a későbbi tanulmányok során szükség lehet rájuk. Mind elméleti, mind gyakorlati téren. Az Interneten nagyon sok helyen utána lehet nézni.