Sor, verem adatszerkezet

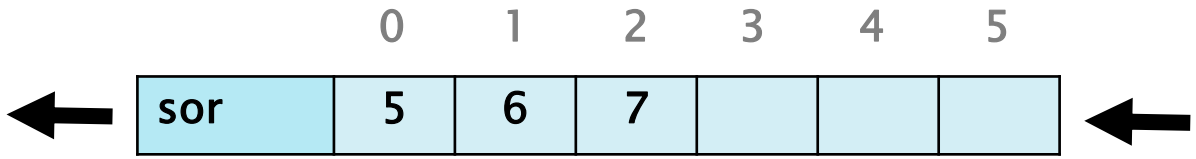
Dr. Szeghalmy Szilvia
Debreceni Egyetem, Informatikai Kar

A sor

- Tulajdonságok
  - Homogén
  - Speciális szekvenciális adatszerkezet.
  - Dinamikus
  - Megőrzi az elemek sorrendjét: First In First Out (FIFO)

- Reprezentáció: folytonos/szétszórt

- Folytonos
  - Rögzített sor (fix sor)
  - Vándorló sor
  - Ciklikus sor



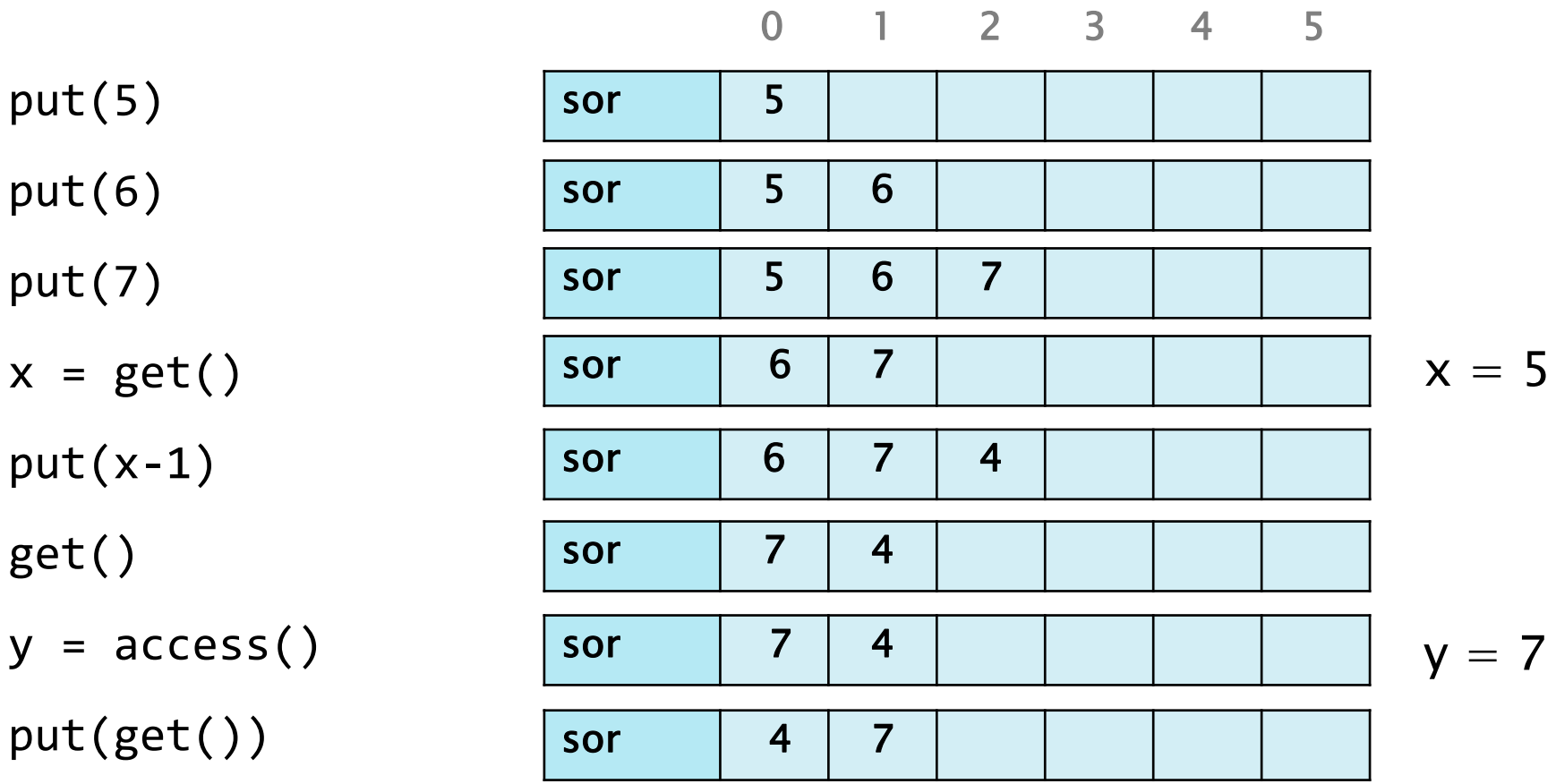
Műveletek áttekintése

- Létrehozás
- Elérés:
  - csak a legelső elem érhető el: x = **access()**
- Módosítás:
  - bővítés: a sor végénél: **put(x)**
  - törlés: a sor elejéről: x = **get()**
  - Csere: -

- Bejárás: -
- Keresés: -
- Rendezés: -

Rögzített sor

- Mi történik a sorral a műveletek hatására?



Létrehozás és segédfüggvények

```
MAX_MERET = 100
sor = np.empty(MAX_MERET, dtype = 'int')  # vagy akár 'object' (később)
elemszam = 0
```

Feladat: Írj függvényt, mely igaz értéket ad vissza, ha a sor üres, különben hamis értéket!

```
def empty():
    return elemszam <= 0            # v. elemszam == 0
```

Feladat: Írj függvényt, mely igaz értéket ad vissza, ha a sor tele van, különben hamis értéket!

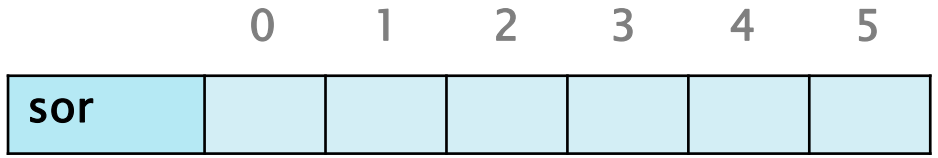
```
def full():
    return elemszam >= MAX_MERET    # v. elemszam == MAX_MERET
```

A put művelet

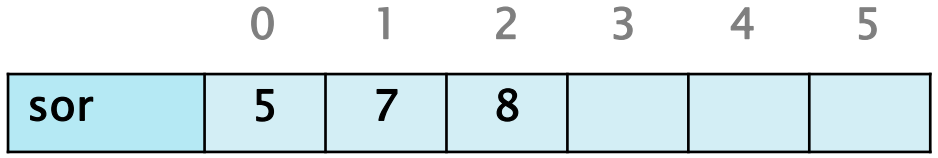
```
# LÉTREHOZÁS:

MAX_MERET = 6
sor=np.empty(MAX_MERET,dtype='int')
elemszam=0
```

Feladat: Írj függvényt, mely beteszi a paraméterben kapott elemet a sorba, **amennyien az lehetséges**.

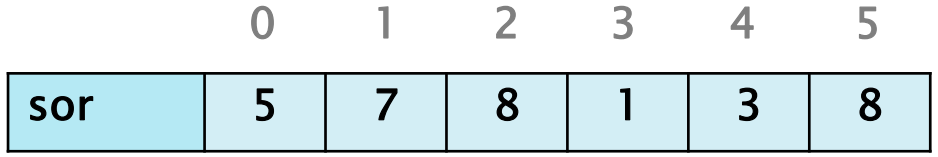
```
def put(x):
    global elemszam
    if not full():  # if elemszam < MAX_MERET:
       sor[elemszam] = x
       elemszam += 1
```



elemszam=0



elemszam=3



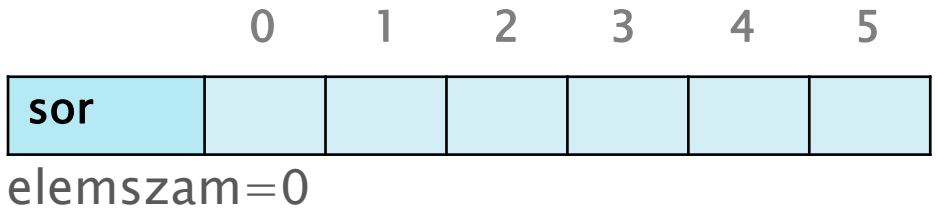
elemszam=6

Az access művelet

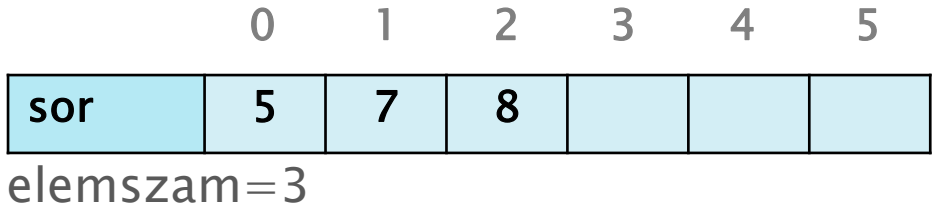
```
# LÉTREHOZÁS:

MAX_MERET = 6
sor=np.empty(MAX_MERET,dtype='int')
elemszam=0
```

Feladat: Írj függvényt, mely visszaadja a sor legelső elemét, amennyiben van. **Ha nincs eleme a sornak, akkor None értékkel térj vissza**.

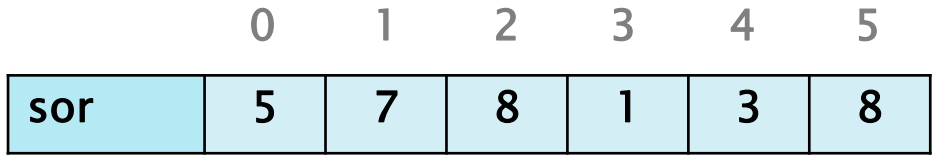
```
def access():
    if not empty():  # if elemszam > 0:
       return sor[0]
    return None
```



elemszam=0



elemszam=3



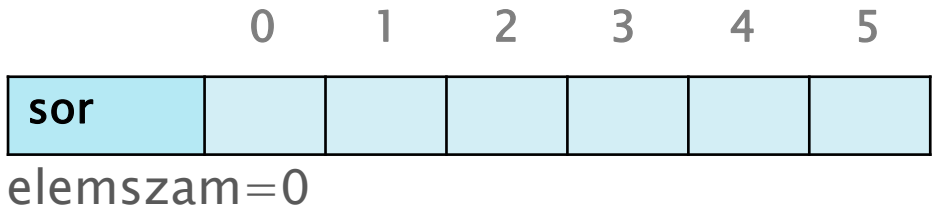
elemszam=6

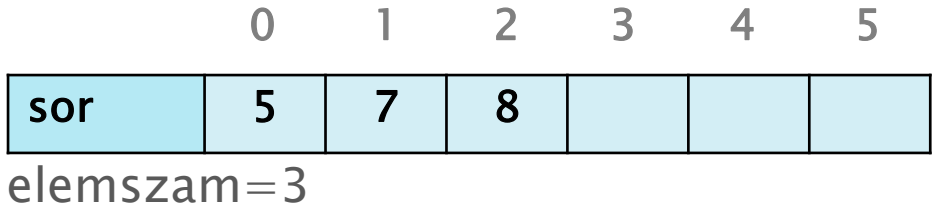
A get művelet

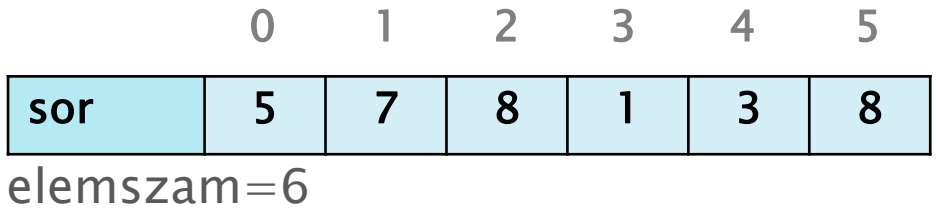
Feladat: Írj függvényt, mely kiveszi a sor legelső elemét (törli) és visszatér az értékkel. **Ha nincs eleme a sornak, akkor None értékkel térj vissza**.

```
def get():
    global elemszam
    if empty(): # if elemszam <= 0:
       return tmp

    tmp = sor[0]
    for i in range(1, elemszam):
        sor[i-1] = sor[i]
    elemszam-=1
    return tmp
```

```
# LÉTREHOZÁS:

MAX_MERET = 6
sor=np.empty(MAX_MERET,dtype='int')
elemszam=0
```







Gyakorlás

Feladat: Írj függvényt, mely visszatér a sor legelső elemének abszolút értékével. Ha az elem nem létezik, akkor -1 értéket adj vissza.
(A függvény ne törölje az elemet.)

```
def abs_ertek():
    if empty():
       return -1
    x = access()
    return x if x >=0 else -x
```

Az építőkockáink:

```
empty()
full()
access()
get()
put(x)
```

Az access és get kivételt dob, ha a sor üres.

A put függvény kivételt dob, ha a sor tele van.

Gyakorlás

Feladat: Írj függvényt, mely kivesz két elemet a sorból és visszaadja az összegüket.

Ha a sor üres, 0 értéket adj vissza.

Ha egy elem létezik, akkor azt tekintsd összegnek.

```
def ossz2():
    ossz = 0
    if not empty():
       ossz += get()
       if not empty():
           ossz += get()
    return ossz
```

Az építőkockáink:

```
empty()
full()
access()
get()
put(x)
```

Az access és get kivételt dob, ha a sor üres.

A put függvény kivételt dob, ha a sor tele van.

Gyakorlás

Feladat: Írj függvényt, mely a bepakol a sorba *n* darab 0-s értéket, vagy amennyi befér.

```
def pakol(n):
    for i in range(n):
        if not full():
            put(0)
        else:
            break
```

Az építőkockáink:

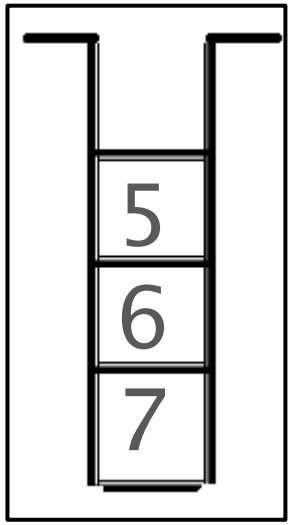
```
empty()
full()
access()
get()
put(x)
```

Az access és get kivételt dob, ha a sor üres.

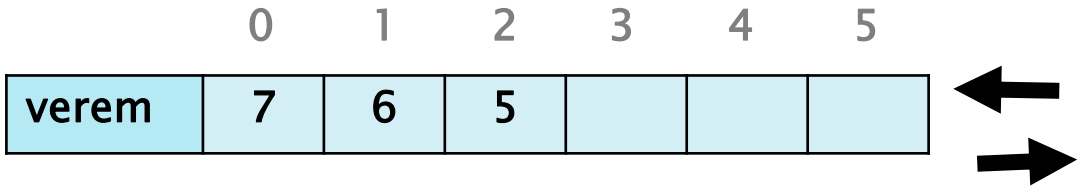
A put függvény kivételt dob, ha a sor tele van.

A verem

- Tulajdonságok
  - Homogén
  - Speciális szekvenciális adatszerkezet.
  - Dinamikus
  - **Megfordítja** az elemek sorrendjét: Last In First Out (**L**IFO)



- Reprezentáció: folytonos/szétszórt
- Folytonos:
  - Tömb
  - **A 0-ás indexen lesz a verem alja**
  - **A verem teteje változik → tárolni kell hol van a legfelső elem/az első szabad hely**



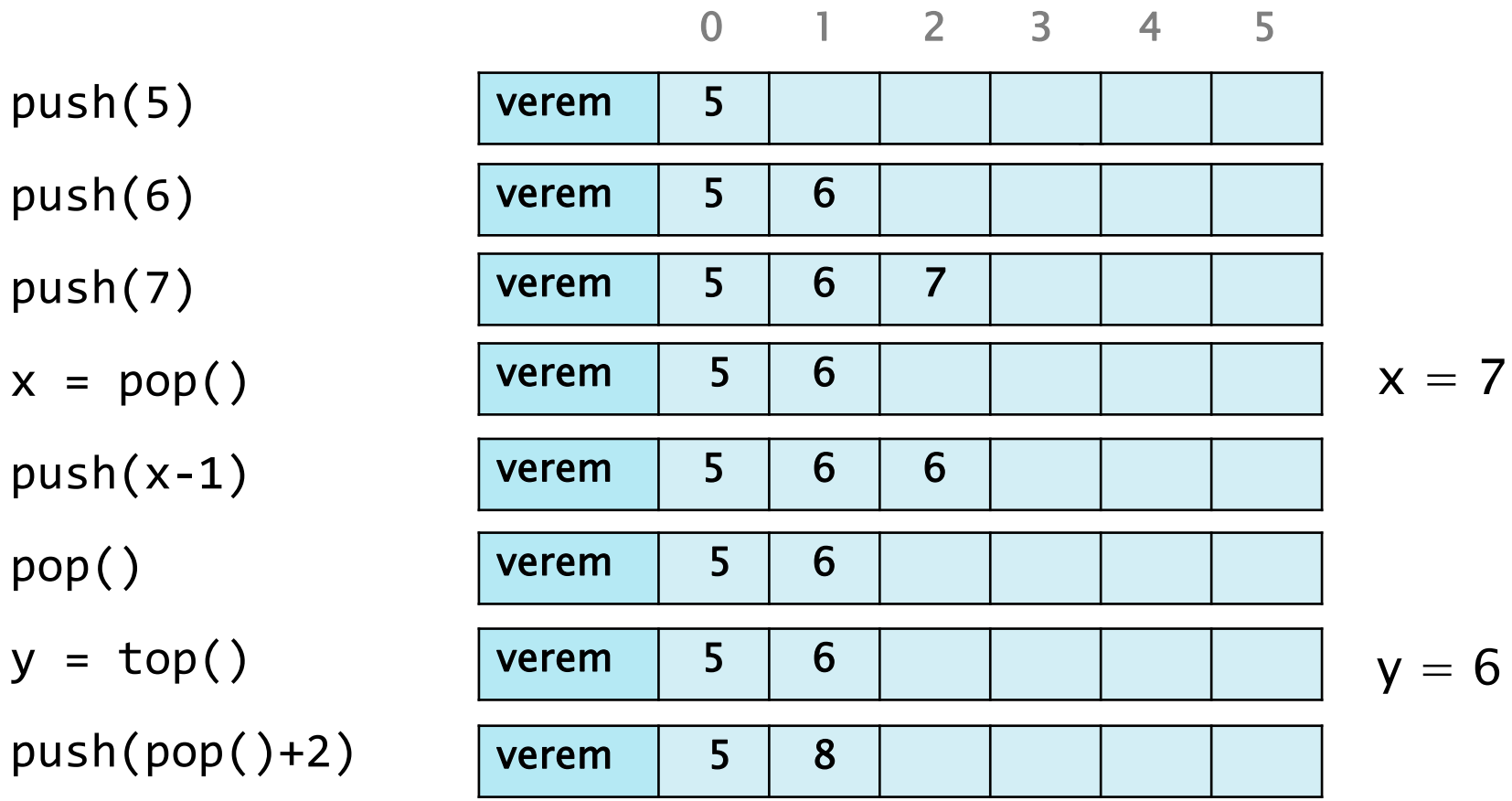
Műveletek áttekintése

- Létrehozás
- Elérés:
  - csak a verem tetején lévő elem érhető el: x = **top()**
- Módosítás:
  - Bővítés: a verem tetejére pakolja az x elemet: **push(x)**
  - Törlés: a verem tetején lévő elemet törli: x = **pop()**
  - Csere: -

- Bejárás: -
- Keresés: -
- Rendezés: -

Verem

- Mi történik a veremmel a műveletek hatására?



Létrehozás és segédfüggvények

```
MAX_MERET = 100
verem = np.empty(MAX_MERET, dtype = 'int')  # vagy akár 'object' (később)
elemszam = 0
```

Feladat: Írj függvényt, mely igaz értéket ad vissza, ha a verem üres, különben hamis értéket!

```
def empty():
    return elemszam <= 0        # v. elemszam == 0
```

Feladat: Írj függvényt, mely igaz értéket ad vissza, ha a verem tele van, különben hamis értéket!

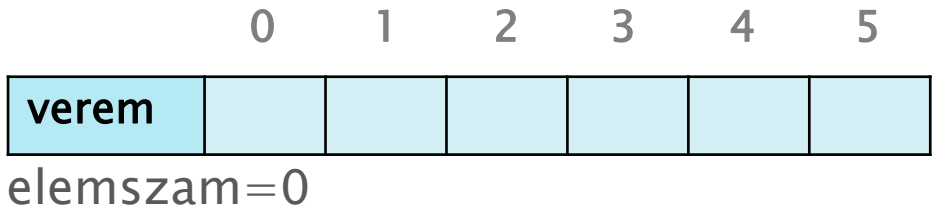
```
def full():
    return elemszam >= MAX_MERET    # v. elemszam == MAX_MERET
```

A push művelet

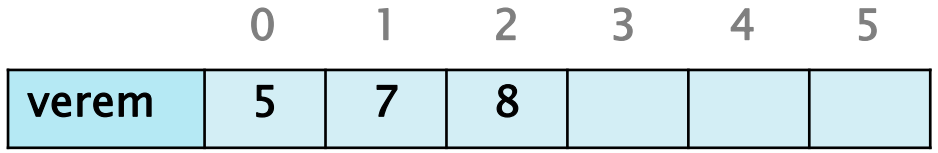
```
# LÉTREHOZÁS:

MAX_MERET = 6
verem=np.empty(MAX_MERET,dtype='int')
elemszam=0
```

Feladat: Írj függvényt, mely beteszi a paraméterben kapott elemet a verembe, **amennyien az lehetséges**.

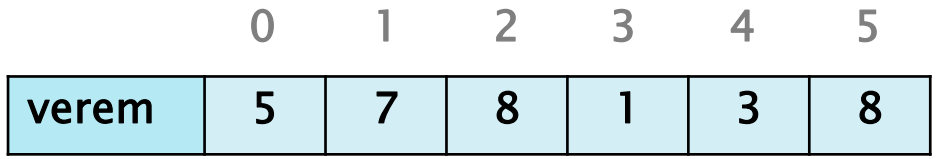
```
def push(x):
    global elemszam
    if not full():  # elemszam < MAX_MERET:
       verem[elemszam] = x
       elemszam += 1
```



elemszam=0



elemszam=3



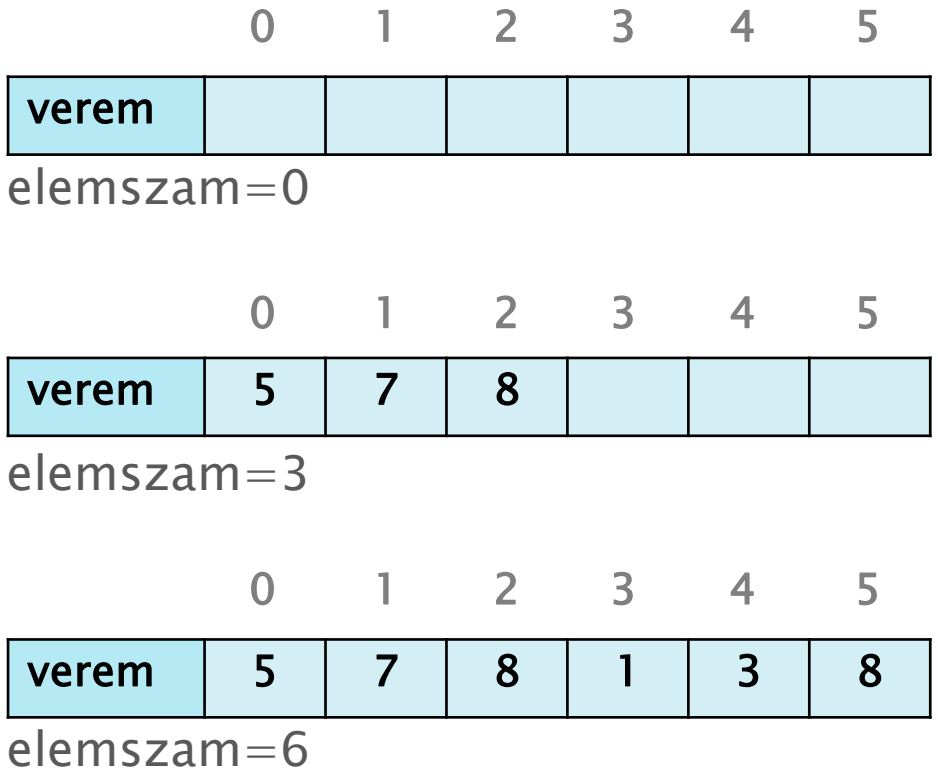
elemszam=6

A top művelet

```
# LÉTREHOZÁS:

MAX_MERET = 6
verem=np.empty(MAX_MERET,dtype='int')
elemszam=0
```

Feladat: Írj függvényt, mely visszaadja a verem tetején lévő értéket. Ha nincs elem a veremben, akkor None értékkel térjen vissza.

```
def top():
    if not empty():  # elemszam > 0:
       return verem[elemszam-1]
    return None
```

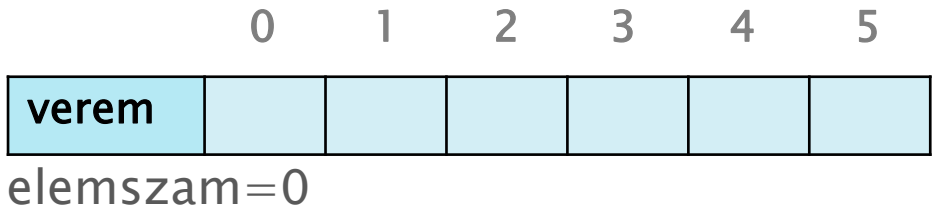


A pop művelet

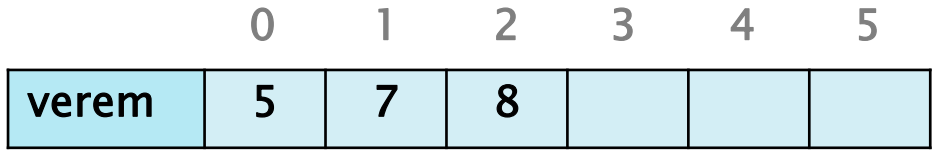
```
# LÉTREHOZÁS:

MAX_MERET = 6
verem=np.empty(MAX_MERET,dtype='int')
elemszam=0
```

Feladat: Írj függvényt, mely kiveszi a veremből (törli) a verem tetején lévő értéket és visszaadja azt. Ha nincs elem a veremben, akkor None értékkel térjen vissza.

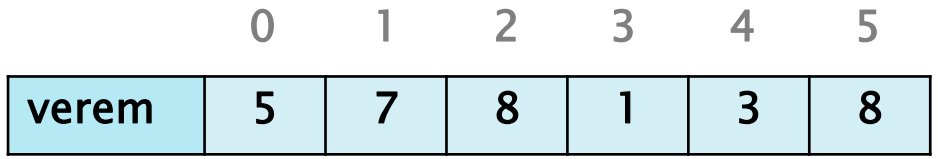
```
def pop():
    global elemszam
    if not empty():   # elemszam > 0:
       elemszam -= 1
       return verem[elemszam]
    return None
```



elemszam=0



elemszam=3



elemszam=6

Gyakorlás: verem – összetett elemek

Valahogy mindig az utoljára kiosztott feladat a legsürgősebb és utána töprenghetünk, hogy melyik feladathoz kellene visszatérnünk. Megoldás? Tároljuk a feladatainkat egy veremben. Az összetett elemek leírására az alábbi osztályt fogjuk használni:

```
class Feladat:
  def __init__(self, kitol, kinek, mit):
     self.kitol = kitol       # ki adta a feladatot, str
     self.kinek = kinek       # kinek kell megcsinálnia, str
     self.mit = mit           # a feladat leírása
```

Egy *Feladat* példány (objektum) létrehozása és használata

```
f = Feladat("Főnök", "Eszter", "szoftver frissítés")  # a self-nek nem adunk át paramétert

if f.kitol == "Főnök":
   f.kinek = "nem én"
print(f.kitol, f.kinek, f.mit)
```

Gyakorlás: verem

Írj függvényt, mely megjeleníti a képernyőn a legújabb feladatot.
Ha nincs feladat a veremben, akkor írja ki a "nincs" üzenetet.
A feladatot NE törölje ki a veremből.

```
def legfrissebb_feladat():
    if empty():
       print("nincs")
    else:
       print(top().mit)
```

```
class Feladat:
  def __init__(self, kitol, mit, mikorra):
     self.kitol = kitol # ki adta
     self.mit = mit
     self.mikorra = mikorra

Példa egy feladat létrehozására
 f = Feladat("Pali","Leltározás","04.10.")
```

Az építőkockáink:

```
empty()
full()
top()
pop()
push(feladat)

A top és pop kivételt
dob, ha a verem üres.

A push kivételt dob, ha a
verem megtelt.
```

Gyakorlás: verem

Ha új feladatot kaptunk a "Főnök"-től (verem tetején van), akkor vegyük ki a veremből és jelenítsük meg, hogy mit kell tenni.

```
def torol_ha_fonok():
    if not empty() and top().kitol == "Főnök":
       f = pop()
       print(f.mit)
```

```
class Feladat:
  def __init__(self, kitol, mit, mikorra):
     self.kitol = kitol # ki adta
     self.mit = mit
     self.mikorra = mikorra

Példa egy feladat létrehozására
 f = Feladat("Pali","Leltározás","04.10.")
```

Az építőkockáink:

```
empty()
full()
top()
pop()
push(feladat)
```

A top és pop kivételt dob, ha a verem üres.

A push kivételt dob, ha a verem megtelt.

Gyakorlás: verem

Egy egész tömbnyi (*n* elemu *t* tomb) feladat érkezett, melyek közül a "Főnök"-től érkezőeket kell áthelyezni a verembe (annyit, amennyi befér).

Használd ki a lehetőséged és a másolás során oszd ki "Feles Elek"-nek azon feladatokat, amelyeket neked kellene megcsinálni.

```
def pakol_ha_fonok(t, n):
    for i in range(n):
        if t[i].kitol == "Főnök":
           f = Feladat(t[i].kitol, t[i].kinek, t[i].mit)

           if t[i].kinek == <a saját neved>:
              f.kinek = "Feles Elek"

           if not full():
              push(f)
           else:
              break
```

```
class Feladat:
  def __init__(self, kitol, mit, mikorra):
     self.kitol = kitol # ki adta
     self.mit = mit
     self.mikorra = mikorra

Példa egy feladat létrehozására
 f = Feladat("Pali","Leltározás","04.10.")
```

Az építőkockáink:

```
empty()
full()
top()
pop()
push(feladat)

A top és pop kivételt
dob, ha a verem üres.

A push kivételt dob, ha a
verem megtelt.
```

Gyakorlás: sor – összetett elemek

Törpfalvában terjed a macskakór. Az ellenszert az interneten történő regisztráció sorrendjében osztják ki a törpöknek, ezért a szoftver egy sor adatszerkezetet használ az adatok memóriában történő tárolására.

A sor minden eleme Torp típusú:

```
class Torp: # Törp
  def __init__(self, n, b, e):
     self.nev = n          # a törp neve, str, Törpfalvában minden törpre egyedi
     self.betegsegei = b   # ismert betegségei, str
     self.eletkor = e      # a törp életkora, int
```

Egy *Torp* példány (objektum) létrehozása és használata

```
t = Torp("Törppapa", "allergia;elhízás", 546) #a self-et hagyd ki

if t.nev == "Törppapa":
    print(t.eletkor) #546
```

Gyakorlás: sor

A törpök rájöttek, hogy 50 év alatt teljesen veszélytelen a macskakór. Írj egy olyan függvényt, mely csak a legalább 50 éves törpöket teszi bele a sorba.
A függvény visszatérési értéke:
2, ha a törp az életkora miatt nem került be a sorba
1, ha a törp nem került be a sorba, mert a sor megtelt
0, ha a törp bekerült a sorba

```
def bepakol(nev, betegsegek, eletkor):
    if eletkor < 50:
       return 2
    if full():
       return 1

    put(Torp(nev, betegsegek, eletkor))
    return 0
```

```
class Torp:
  def __init__(self, n, b, e):
     self.nev = n
     self.betegsegei = b
     self.eletkor = e  # int

Példa egy törp létrehozására
t = Torp("Törppapa","demencia", 546)
```

Az építőkockáink:

```
empty()
full()
access()
get()
put(torp)

Az access és get kivételt
dob, ha a sor üres.

A put kivételt dob, ha a
sor megtelt.
```

Gyakorlás: sor

Írj egy függvényt, mely visszaadja, hogy milyen ellenszer adható be macskakór ellen a soron következő törpnek. A törpöt NE töröld a sorból. Ha a sor üres, akkor a függvény "nincs jelentkező" sztringet adjon vissza.

eletkor	betegség	ellenszer
<100	-	"macskabajusz"
>=100	"allergia"	"macskacsont"
	egyébként	"macskaszőr"

```
def hasznalhato_ellenszer():
    if empty():
       return "nincs jelentkező"
    t = access()
    if t.eletkor < 100:
       return "macskabajusz"
    if "allergia" in t.betegsegei: # str in művelete
       return "mcsakacsont"
    return "macskaszőr"
```

```
class Torp:
  def __init__(self, n, b, e):
     self.nev = n
     self.betegsegei = b
     self.eletkor = e  # int

Példa egy törp létrehozására
t = Torp("Törppapa","demencia", 546)
```

Az építőkockáink:

```
empty()
full()
access()
get()
put(torp)
```

Az access és get kivételt dob, ha a sor üres.

A put kivételt dob, ha a sor megtelt.

Gyakorlás: sor

Ellenszer érkezett 10 törp számára. Vedd ki az első 10 elemet a sorból (vagy amennyi van) és jelenítsd meg a törpök neveiket a képernyőn.

```
def elso10():
    for i in range(10):
        if not empty():
            t = get()
            print(t.nev)
        else: # mindig csak törlünk, ezért ha már üres az is marad
            break
```

```
class Torp:
  def __init__(self, n, b, e):
     self.nev = n
     self.betegsegei = b
     self.eletkor = e  # int

Példa egy törp létrehozására
t = Torp("Törppapa","demencia", 546)
```

Az építőkockáink:

```
empty()
full()
access()
get()
put(torp)
```

Az access és get kivételt dob, ha a sor üres.

A put kivételt dob, ha a sor megtelt.

Gyakorlás: sor

A sor elején álló törp nem fogadta el a számára ajánlott ellenszert. Vedd ki a sorból és tedd át a sor végére, hátha később lesz olyan ellenszer, melyet hajlandó elfogadni.
Megj.: van elem a sorban

```
def hatra_kerul():
    put(get())
```

```
class Torp:
  def __init__(self, n, b, e):
     self.nev = n
     self.betegsegei = b
     self.eletkor = e  # int

Példa egy törp létrehozására
t = Torp("Törppapa","demencia", 546)
```

Az építőkockáink:

```
empty()
full()
access()
get()
put(torp)

Az access és get kivételt
dob, ha a sor üres.

A put kivételt dob, ha a
sor megtelt.
```

Gyakorlás: sor

A sor elején álló törp hazudott az életkoráról, valójában csak 99 éves. Vedd ki a sorból, javítsd az életkorát és tedd át a sor végére.
Megj.: van elem a sorban

```
def hatra_kerul():
    t = get()
    t.eletkor = 99
    put(t)
```

```
class Torp:
  def __init__(self, n, b, e):
     self.nev = n
     self.betegsegei = b
     self.eletkor = e  # int

Példa egy törp létrehozására
t = Torp("Törppapa","demencia", 546)
```

Az építőkockáink:

```
empty()
full()
access()
get()
put(torp)

Az access és get kivételt
dob, ha a sor üres.

A put kivételt dob, ha a
sor megtelt.
```

Gyakorlás: sor

```
class Torp:
  def __init__(self, n, b, e):
     self.nev = n
     self.betegsegei = b
     self.eletkor = e  # int

Példa egy törp létrehozására
t = Torp("Törppapa","demencia;allergia",
546)
```

Az allergiás törpök kizárólag macskacsontot kaphatnak ellenszerből, de ebből sajnos hiány van. A többi törp kezelését azonban folytatni akarjuk.

Írj egy olyan *get2* függvényt, mely kiveszi a sorból és visszaadja **az első nem allergiás** törp objektumot. (**in** string operátor használható)

- Ha a nem allergiás törp előtt voltak allergiás törpök, akkor őket vedd ki és tedd át a sor végére.
- Ha a sor üres vagy nincs egy nem allergiás törp sem, akkor a függvény None értéket adjon vissza.

```
def get2():
    while not empty():
        t = get() # kiveszi
        if "allergia" in t.betegsegei:
            put(t)
        else:
            return t
    return None # Python default
```

Az építőkockáink:

```
empty()
full()
access()
get()
put(torp)
```

Az access és get kivételt dob, ha a sor üres.

A put kivételt dob, ha a sor megtelt.