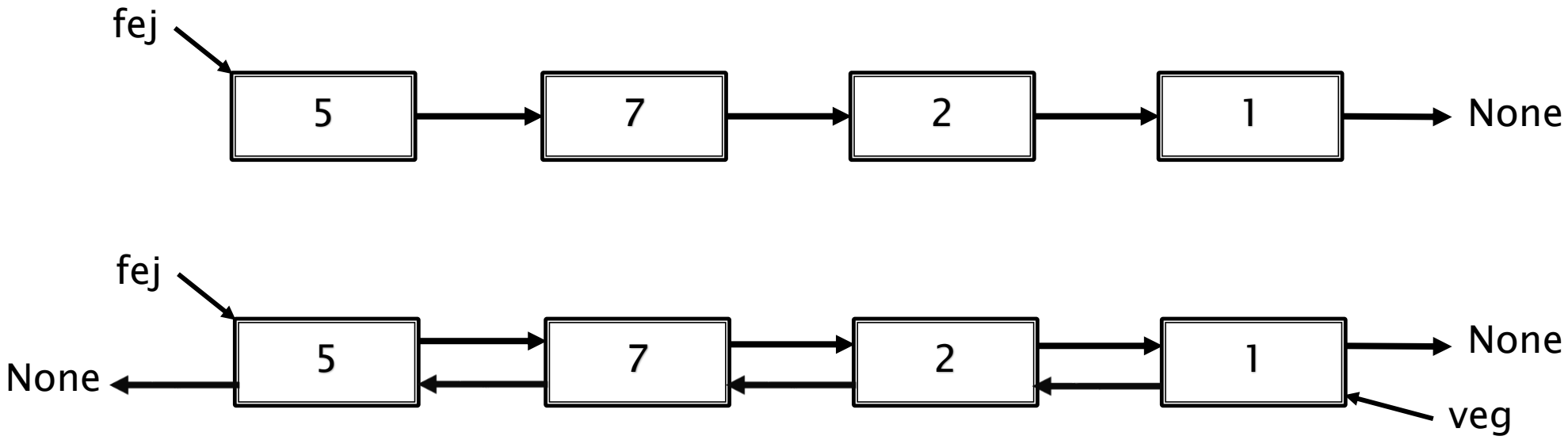
Kétirányban láncolt lista adatszerkezet

Dr. Szeghalmy Szilvia
Debreceni Egyetem, Informatikai Kar

A láncolt lista

- Tulajdonságok
  - Homogén
  - Szekvenciális
  - **Dinamikus**

- Reprezentáció: **szétszórt** / folytonos



Egyirányban láncolt lista: Létrehozás

- Összetett elem:
  - adatrész: problémafüggő, az adatrész is lehet összetett
  - a következő elem címe vagy annak hiányában None érték

- Létrehozás: üres listát hozunk létre és bővítjük

```
class Elem:
    def __init__(self, adat):
        self.adat = adat      # az adat rész típusa a problémától függ
        self.kov = None       # a következő objektum "címe"
# üres lista
fej = None
```

Kétirányban láncolt lista: Létrehozás

- Egy elem részei

```
class Elem:
    def __init__(self, adat):
        self.adat = adat    # Adattag: problémától függően
        self.elozo = None   # Az előző elem címe, ha nincs, akkor None
        self.kov = None     # Az utolsó elem címe, ha nincs, akkor None
```

- A lista első és utolsó elemének tárolására szolgáló saját típus:

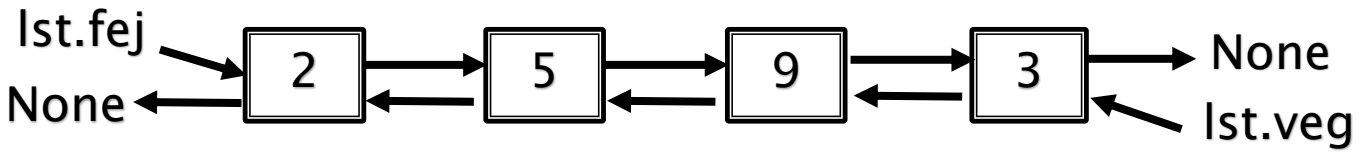
```
class Lista:
    def __init__(self):
        self.fej = None    # a lista legelső elemének címe
        self.veg = None    # a lista legutolsó elemének címe
```

- Egy üres lista létrehozása

```
lst = Lista()
```

Műveletek: elérés

- **Elérés**: szekvenciális, az első v. utolsó elemtől indulva a lánc segítségével.
- Pl.:

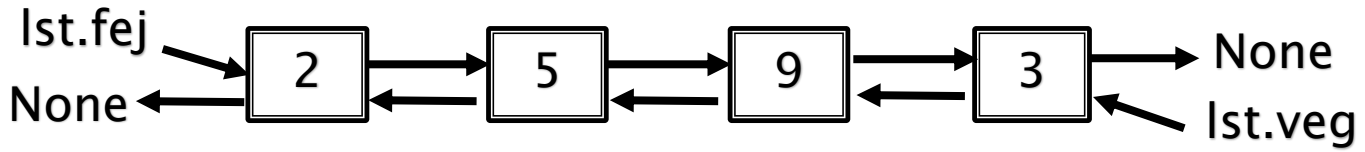


elem	elérés a fej felől	elérés a vég felől
2:	`lst.fej`	`lst.veg.elozo.elozo.elozo`
5:	`lst.fej.kov`	`lst.veg.elozo.elozo`
9:	`lst.fej.kov.kov`	`lst.veg.elozo`
3:	`lst.fej.kov.kov.kov`	`lst.veg`

Ha az adatrészre van szükség: lst.fej.**adat**, lst.fej.kov.**adat, …**

```
# LÉTREHOZÁS:

class Elem:
   def __init__(self,adat):
        self.adat = adat
        self.elozo = None
        self.kov = None

class Lista:
    def __init__(self):
        self.fej = None
        self.veg = None

lst = Lista()
```

Feladatok



Feladat1: Növeld meg a lista 3. elemének (9-es) értékét 1-gyel.

```
      lst.fej.kov.kov.adat += 1
#vagy
      lst.veg.elozo.adat += 1
```

Feladat1: Cseréld fel a lista a 4 elemű lista két középső elemének érétékét.

```
      tmp = lst.fej.kov.adat
      lst.fej.kov.adat = lst.veg.elozo.adat
      lst.veg.elozo.adat = tmp
```

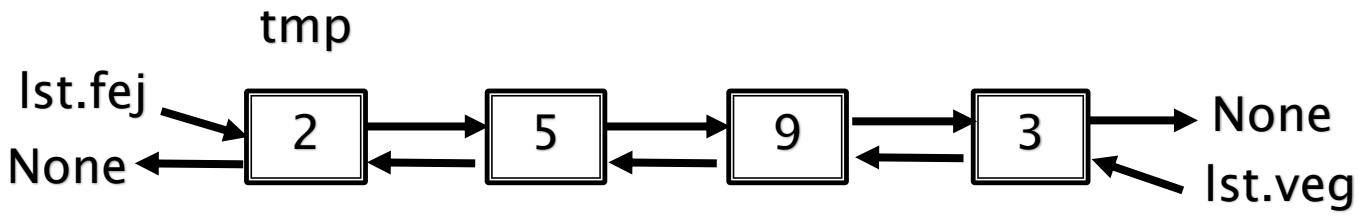
```
# LÉTREHOZÁS:

class Elem:
   def __init__(self,adat):
        self.adat = adat
        self.elozo = None
        self.kov = None

class Lista:
    def __init__(self):
        self.fej = None
        self.veg = None

lst = Lista()
```

Műveletek: bejárás 1

- **A fejtől indulva:**

Feladat: Írjuk ki a listában lévő adatokat előrefelé haladva.

```
def bejar_elore(lst): # Lista típusú
     tmp = lst.fej     # a lista első elemére ráállunk
     while tmp:        # amíg a cím érvényes (nem None)
         print(tmp.adat)   # aktuális elem feldolgozása
         tmp = tmp.kov     # átugrunk a következő elemre
```




```
# LÉTREHOZÁS:

class Elem:
   def __init__(self,adat):
        self.adat = adat
        self.elozo = None
        self.kov = None

class Lista:
    def __init__(self):
        self.fej = None
        self.veg = None

lst = Lista()
```

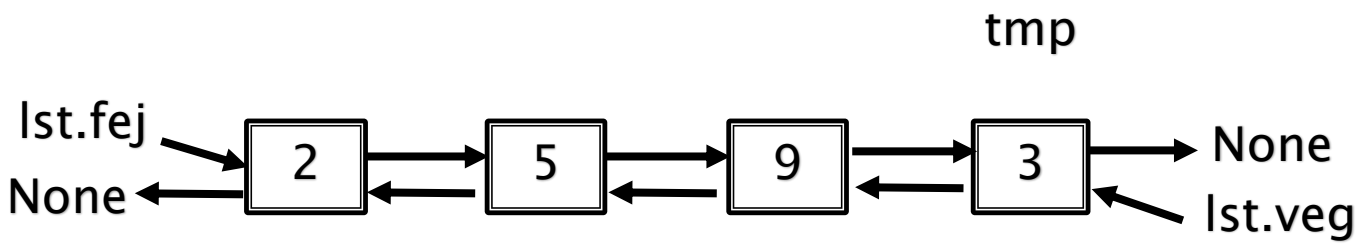
Műveletek: bejárás 2

- **A végtől indulva:**

Feladat: Írjuk ki a listában lévő adatokat hátrafelé haladva.

```
def bejar_hatra (lst):
      tmp = lst.veg
      while tmp:
          print(tmp.adat)
          tmp = tmp.elozo
```

```
# LÉTREHOZÁS:

class Elem:
   def __init__(self,adat):
        self.adat = adat
        self.elozo = None
        self.kov = None

class Lista:
    def __init__(self):
        self.fej = None
        self.veg = None

lst = Lista()
```



Műveletek: keresés (általános)

- Feladat: Írjunk függvényt, mely visszaadja a listában található első *x* értékű elemet.
- Ha nincs ilyen elem None értékkel térjünk vissza.

```
def keres(lst, x):  # lst típus Lista, az x típusa int
    tmp = lst.fej
    while tmp:   # while tmp is not None:
        if tmp.adat == x:
            return tmp
        tmp = tmp.kov
    return None
```

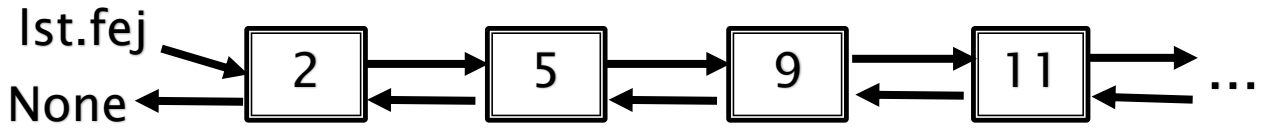
```
# LÉTREHOZÁS:

class Elem:
   def __init__(self,adat):
        self.adat = adat
        self.elozo = None
        self.kov = None

class Lista:
    def __init__(self):
        self.fej = None
        self.veg = None

lst = Lista()
```

Műveletek: keresés (rendezett)

- Feladat: Írjunk függvényt, mely visszaadja a listában található első *x* értékű elemet.
- Ha nincs ilyen elem None értékkel térjünk vissza.
- Tudjuk, hogy a lista rendezett (növekvő sorrend)

```
def keres(lst, x):  # lst típus Lista, az x típusa int
    tmp = lst.fej
    while tmp:
        if tmp.adat == x:
            return tmp
        if tmp.adat > x:  # kilép, ha már úgysem lehet benne
            return None
        tmp = tmp.kov
    return None
```

```
# LÉTREHOZÁS:

class Elem:
   def __init__(self,adat):
        self.adat = adat
        self.elozo = None
        self.kov = None

class Lista:
    def __init__(self):
        self.fej = None
        self.veg = None

lst = Lista()
```



Műveletek: csere

- A csere az **adatrészt** érinti, a lánc nem változik

- Írj függvényt, mely kicseréli a lista egyetlen mit értékű elemét a mire értékre. A megoldáshoz használd az előbb megírt *keres(lista, adat)* függvényt.

```
def felulir(lst, mit, mire):
    elem = keres(lst, mit)
    if elem:   # if elem is not None:
       elem.adat = mire
```

```
# LÉTREHOZÁS:

class Elem:
   def __init__(self,adat):
        self.adat = adat
        self.elozo = None
        self.kov = None

class Lista:
    def __init__(self):
        self.fej = None
        self.veg = None

lst = Lista()
```

Feladat

```
# LÉTREHOZÁS:

class Elem:
   def __init__(self,n, j):
        self.neptun = n
        self.jegy = j
        self.elozo = None
        self.kov = None

class Lista:
    def __init__(self):
        self.fej = None
        self.veg = None

lst = Lista()
```

- Tegyük fel a listában diákok érdemjegyét tároljuk. Írd felül az összes 1-es osztályzatot 2-esre.

```
def felulir(lst):
    tmp = lst.fej
    while tmp:
        if tmp.jegy == 1:
           tmp.jegy = 2
        tmp = tmp.kov
```

- Írd felül a saját osztályzatodat 5-ösre.
  A megoldáshoz használhatod a keres(lista, neptunkod) függvényt.

```
def felulir(lst):
    tmp = keres(lst, "XXXXXX")
    if tmp:
       tmp.jegy = 5  # nem gond ha jeles volt eleve, az is marad
```

Gyakorlat vs. e-learning

Az algoritmusok megadásánál feltételezzük, hogy az elemek rendben létrejönnek.

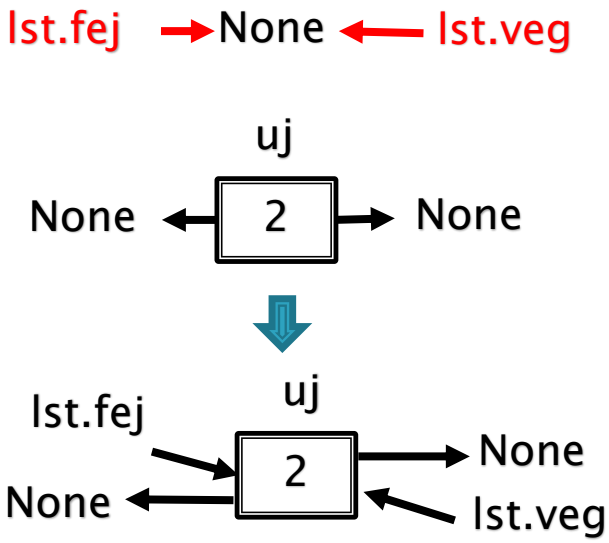
Az e-learningben találkozhatnak kivételeket kezelő részekkel.

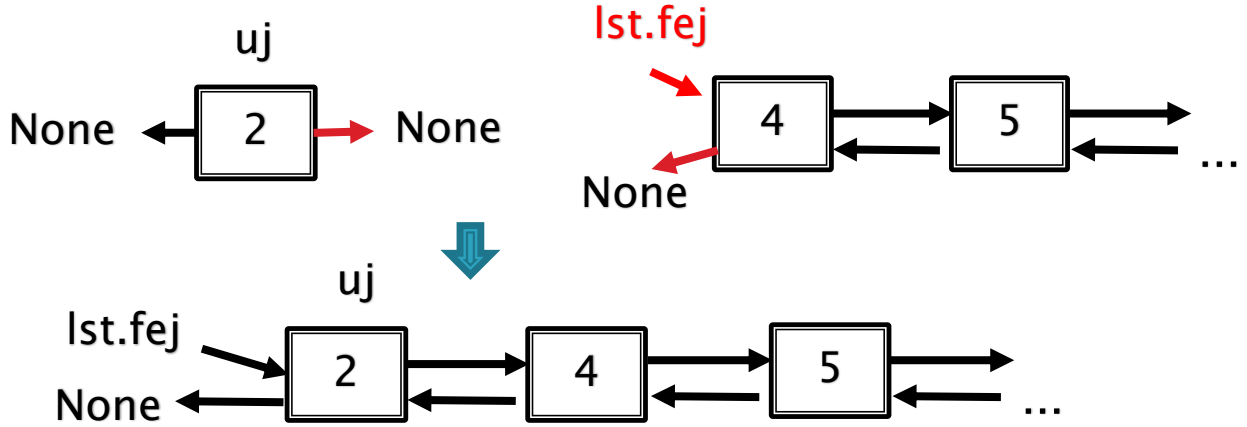
```
uj = None
try:
   uj = Elem(adat)
except MemoryError:
   …
```

Műveletek: beszúrás előre

- Esetek:
  - a lista üres: a fej és vég is változik
  - a lista nem üres: a **vég** nem változik

```
#lst:Lista; adat: int
def beszur_elore(lst, adat):
    uj = Elem(adat)
    # üres lista
    if lst.fej is None:
        lst.fej = lst.veg = uj

    # nem üres lista
    else:
        uj.kov = lst.fej
        lst.fej.elozo = uj
        lst.fej = uj
```



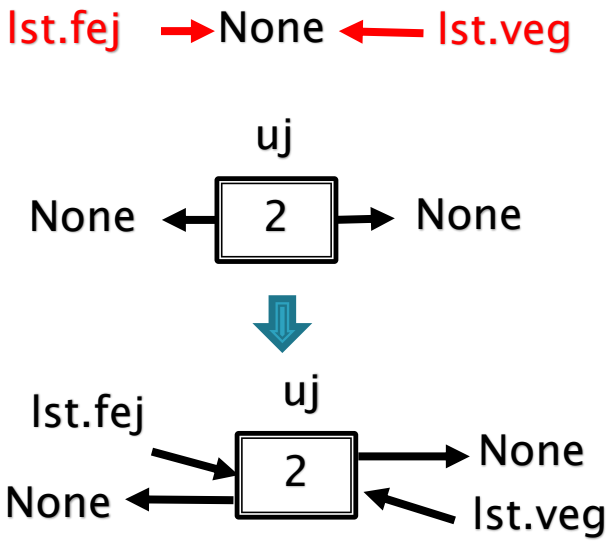


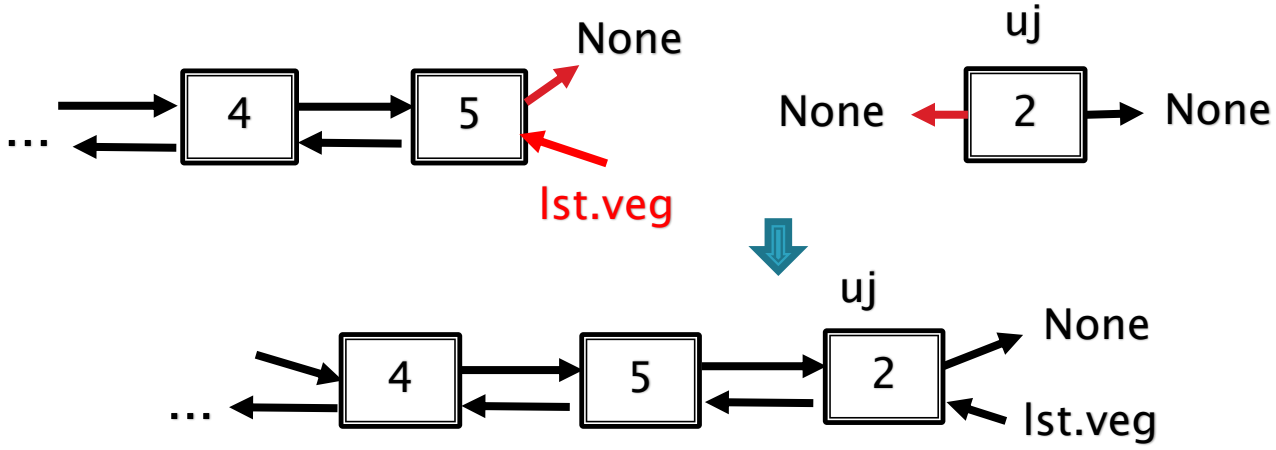

```
# LÉTREHOZÁS:

class Elem:
   def __init__(self,adat):
        self.adat = adat
        self.elozo = None
        self.kov = None

class Lista:
    def __init__(self):
        self.fej = None
        self.veg = None

lst = Lista()
```

Műveletek: beszúrás végére

- Esetek:
  - a lista üres: a fej és vég is változik
  - a lista nem üres: a **fej** nem változik

```
#lst:Lista; adat: int
def beszur_vegere(lst, adat):
    uj = Elem(adat)
    # üres lista
    if lst.fej is None:
        lst.fej = lst.veg = uj

    # nem üres lista
    else:
        uj.elozo = lst.veg
        lst.veg.kov = uj
        lst.veg = uj
```



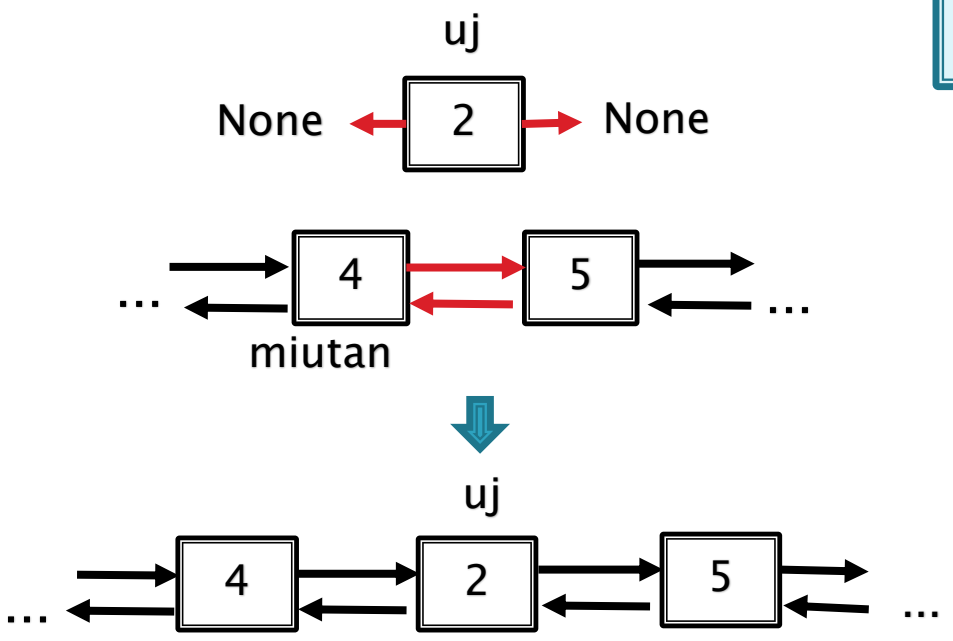



```
# LÉTREHOZÁS:

class Elem:
   def __init__(self,adat):
        self.adat = adat
        self.elozo = None
        self.kov = None

class Lista:
    def __init__(self):
        self.fej = None
        self.veg = None

lst = Lista()
```

Műveletek: beszúrás "középre"

- Az újonnan beszúrt elem a lista közbenső eleme lesz
- Legalább két elem van a listába
- Azt az elemet kapja meg a függvény, **ami után** beszúrunk

```
# miutan: Elem;   adat: int
def beszur_kozbenso(miutan, adat):
    uj = Elem(adat)
    uj.elozo = miutan
    uj.kov = miutan.kov
    miutan.kov = uj
    uj.kov.elozo = uj
```



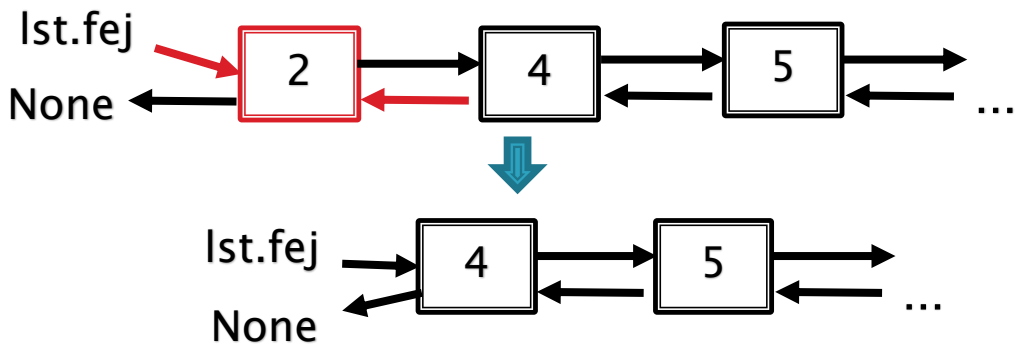

```
# LÉTREHOZÁS:

class Elem:
   def __init__(self,adat):
        self.adat = adat
        self.elozo = None
        self.kov = None

class Lista:
    def __init__(self):
        self.fej = None
        self.veg = None

lst = Lista()
```

Műveletek: fej törlése

- Esetek:
  - a lista üres -> semmi nem változik
  - a listába egy elem van -> a fej és vég is módosul
  - a listában több elem van -> a fej módosul, a vég nem

```
# LÉTREHOZÁS:

class Elem:
   def __init__(self,adat):
        self.adat = adat
        self.elozo = None
        self.kov = None

class Lista:
    def __init__(self):
        self.fej = None
        self.veg = None

lst = Lista()
```

```
# lst: Lista
def torol_fej(lst):
    if not lst.fej:  # nincs elem
        return
    if lst.fej == lst.veg:  # egy elem
        #del lst.fej  -- a felszabadítás helye, amely nyelvben szükséges
        lst.fej = lst.veg = None
    else:
        lst.fej = lst.fej.kov
        #del lst.fej.elozo
        lst.fej.elozo = None
```

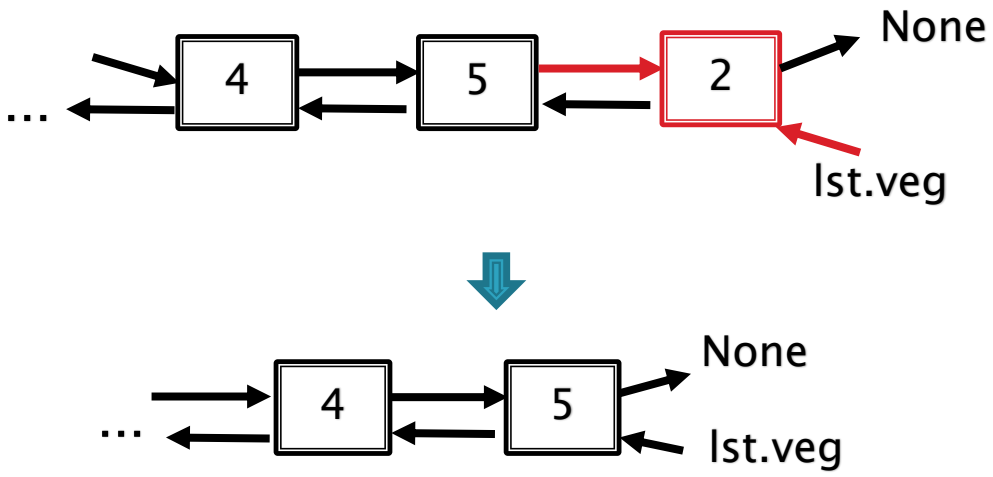


Műveletek: vég törlése

- Esetek:
  - a lista üres -> semmi nem változik
  - a listába egy elem van -> a fej és vég is módosul
  - **a listában több elem van -> a vég módosul, a fej nem**

```
# lst: Lista
def torol_veg(lst):
    if not lst.fej:  # nincs elem
        return
    if lst.fej == lst.veg:  # egy elem
        #del lst.fej
        lst.fej = lst.veg = None
    else:
        lst.veg = lst.veg.elozo
        #del lst.veg.kov
        lst.veg.kov = None
```

```
# LÉTREHOZÁS:

class Elem:
   def __init__(self,adat):
        self.adat = adat
        self.elozo = None
        self.kov = None

class Lista:
    def __init__(self):
        self.fej = None
        self.veg = None

lst = Lista()
```

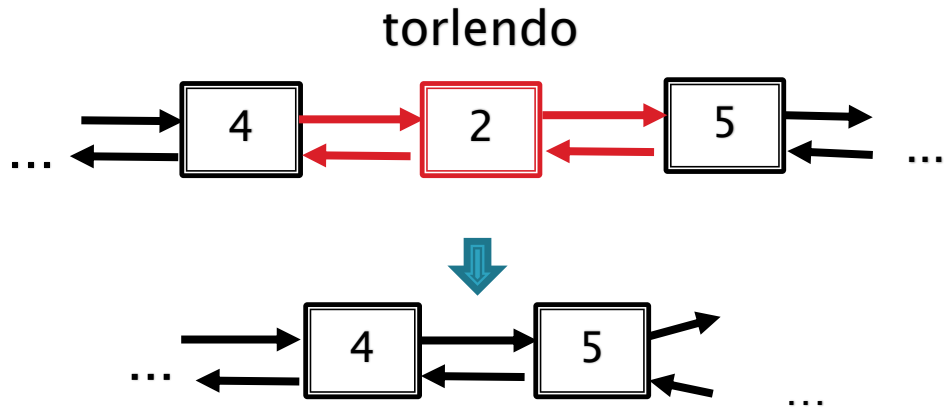


Műveletek: közbenső elem törlése

- A függvény a törlendő elemet kapja meg
- A törlendő elemnek van előző és következő eleme
  - => a listának legalább három eleme van, a fej és vég nem változik

```
# torlendo: Elem
def torol_kozbenso(torlendo):
    torlendo.elozo.kov = torlendo.kov
    torlendo.kov.elozo = torlendo.elozo
    # del torlendo – a felszabaditas helye, amely nyelvnél kell
    torlendo = None
```

```
# LÉTREHOZÁS:

class Elem:
   def __init__(self,adat):
        self.adat = adat
        self.elozo = None
        self.kov = None

class Lista:
    def __init__(self):
        self.fej = None
        self.veg = None

lst = Lista()
```



Műveletek: törlés (ált. eset)

- Töröljük ki a listából az *x* értékű elemet. (első előfordulás)
  - **nincs benne a törlendő a listában**
  - **a lista elejéről törlünk**
  - **a lista végéről törlünk**
  - **a lista közbenső elemét töröljük**

```
def torol( lst, x ):  # lst: Lista;  x - int
    torlendo = keres(lst, x)
    if not torlendo:
       return
    if torlendo == lst.fej:
       torol_fej(lst)
    elif torlendo == lst.veg:
       torol_veg(lst)
    else:
       torol_kozbenso(torlendo)
```

```
# LÉTREHOZÁS:

class Elem:
   def __init__(self,adat):
        self.adat = adat
        self.elozo = None
        self.kov = None

class Lista:
    def __init__(self):
        self.fej = None
        self.veg = None

lst = Lista()
```

Műveletek: felszabadítás

```
# LÉTREHOZÁS:

class Elem:
   def __init__(self,adat):
        self.adat = adat
        self.elozo = None
        self.kov = None

class Lista:
    def __init__(self):
        self.fej = None
        self.veg = None

lst = Lista()
```

- Az összes elem törlése

Általánosságban az algoritmus:

```
def felszabaditas( lst):  # lst: Lista;
    tmp = lst.fej
    while tmp:
        tmp2 = tmp.kov # elmentjük a lista következő elemének címét, mert
        del tmp  # a törlés után már nem érnénk el
        tmp = tmp2  # a következő elemre lépünk az elmentett cím segítéségével
    lst.fej = lst.veg = None
```

Megj.: Pythonban a list.fej = list.veg = None is megoldja a felszabadítást, mert a szemétgyűjtő minden olyan elemet automatikusan eltávolít a memóriából, mely elérhetetlenné válik. Észleli azt is, hogy a listaelemek csak egymásra hivatkoznak (ld. garbage collector, cycle collector).

Gyakorlás

Egy listában **különböző** városokhoz tároljuk a lakosok számát.

Jelenítsd meg azon városokat, melynek lakossága meghaladja a *hatar* paraméterben megadott értéket.

```
def sok_lakos( lst, hatar ):  # lst: Lista;  hatar - int
    tmp = lst.fej
    while tmp:
        if tmp.lakos > hatar:
            print(tmp.varos)
        tmp = tmp.kov
```

```
# LÉTREHOZÁS:

class Elem:
   def __init__(self,v,l):
        self.varos = v
        self.lakos = l
        self.elozo = None
        self.kov = None

class Lista:
    def __init__(self):
        self.fej = None
        self.veg = None

lst = Lista()
```

Gyakorlás

Írj függvényt, mely visszaadja, hogy hányan laknak Debrecenben.
Ha Debrecen nincs a listában, akkor -1 értéket adj vissza.

```
def debreceni_lakosok_szama(lst):  # lst: Lista;
    tmp = lst.fej
    while tmp:
        if tmp.varos == "Debrecen":
            return tmp.lakos
        tmp = tmp.kov
    return -1
```

```
# LÉTREHOZÁS:

class Elem:
   def __init__(self,v,l):
        self.varos = v
        self.lakos = l
        self.elozo = None
        self.kov = None

class Lista:
    def __init__(self):
        self.fej = None
        self.veg = None

lst = Lista()
```

Gyakorlás

Tegyük fel, hogy a megyét is tároljuk. Írj függvényt, mely megadja egy paraméterként kapott megye városainak átlagos lakosságát.

Amennyiben nincs a keresett megye a listában, akkor -1 értéket adj vissza.

```
def atlag_lakossag(lst, megye):  # lst: Lista; megye: str
    tmp = lst.fej
    s = 0
    db = 0
    while tmp:
        if tmp.megye == megye:
            s += tmp.lakos
            db += 1
        tmp = tmp.kov
    if db == 0:
        return -1
    return s / db
```

```
# LÉTREHOZÁS:
class Elem:
    def __init__(self,v,l,m):
        self.varos = v
        self.lakos = l
        self.megye = m
        self.elozo = None
        self.kov = None

class Lista:
    def __init__(self):
        self.fej = None
        self.veg = None

lst = Lista()
```

Gyakorlás

Írj függvényt, mely megadja, hogy mely városban laknak a legkevesebben. (nincs holtverseny)

Ha nincs elem a listában, akkor üressztringet adja vissza.

```
import math
def legkisebb_varos(lst):  # lst: Lista;
    tmp = lst.fej
    min = math.inf
    min_varos = ""

    while tmp:
        if tmp.lakos < min:
            min = tmp.lakos
            min_varos = tmp.varos
        tmp = tmp.kov
    return min_varos
```

```
# LÉTREHOZÁS:

class Elem:
   def __init__(self,v,l):
        self.varos = v
        self.lakos = l
        self.elozo = None
        self.kov = None

class Lista:
    def __init__(self):
        self.fej = None
        self.veg = None

lst = Lista()
```