# Bináris fa adatszerkezet gyakorlás

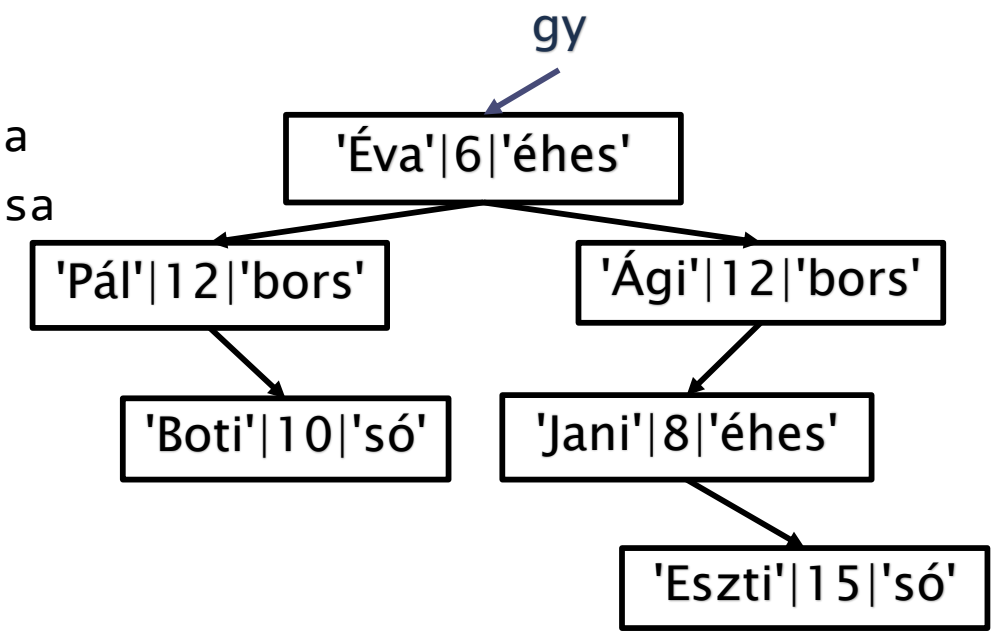
Dr. Szeghalmy Szilvia
Debreceni Egyetem, Informatikai Kar

# Binárisfa gyakorlás

- Egy gombócevő verseny résztvevőinek nevét (egyedi) és az általuk megevett gombócok számát és a csapatuk nevét egy bináris fában tároljuk.
- Írj függvényt, mely **megjeleníti**, hogy ki hány gombócot evett meg.

```
# LÉTREHOZÁS:
class Faelem:
  def __init__(self, n, c):
     self.nev = n
     self.gomboc = 0
     self.csapat = c
     self.bal = None
     self.jobb = None

gy = None
```

```
def eredmeny(gy):  gy a (rész)fa gyökere, Faelem típusú
    if not gy:       # üres fa esete
        return
    print(gy.nev, gy.gomboc)  # a gyökér feldolgozása
    eredmeny(gy.bal)  # a bal oldali részfa feldolgozása
    eredmeny(gy.jobb) # a jobb oldali részfa feldolgozása
```



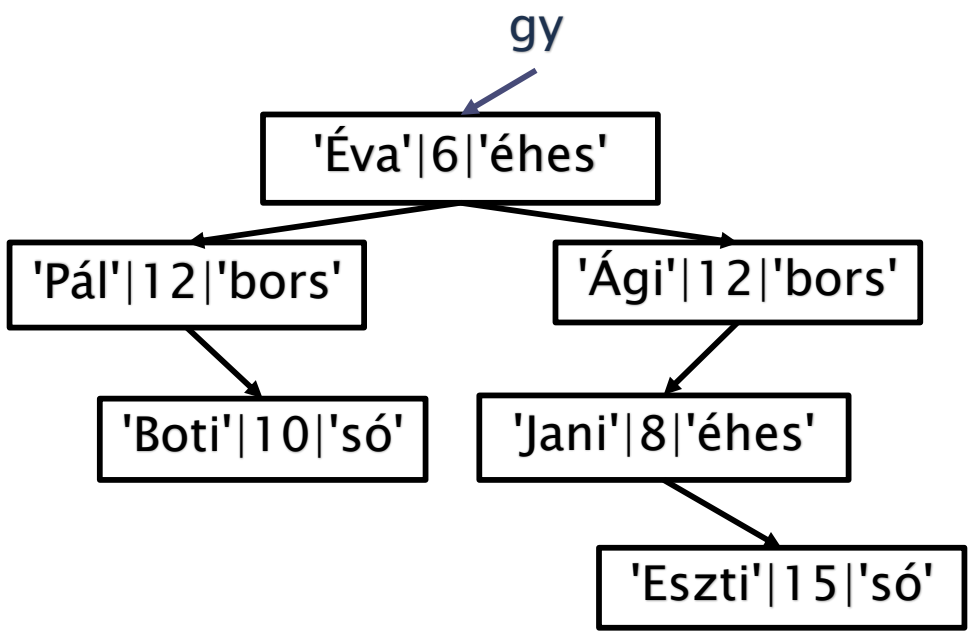
# Binárisfa gyakorlás

```
# LÉTREHOZÁS:
class Faelem:
  def __init__(self, n, c):
     self.nev = n
     self.gomboc = 0
     self.csapat = c
     self.bal = None
     self.jobb = None

gy = None
```

- A verseny közben folyamatosan frissíteni akarjuk az eredményeket:

  írj egy függvényt, mely **megnöveli** eggyel a gombócok számát a paraméterben átadott nevű versenyzőnél.

```
def evett_egyet(gy, nev): # gy:Faelem, nev:str
   if not gy:       # üres fa esete
       return
   if gy.nev == nev:  # a gyökér feldolgozása
      gy.gomboc += 1
      # a név egyedi, megtaláltuk => kiléphetünk
      return
   evett_egyet(gy.bal, nev)  # a bal oldali részfa,
   evett_egyet(gy.jobb, nev) # és a jobb oldali részfa
                                  # feldolgozása
```

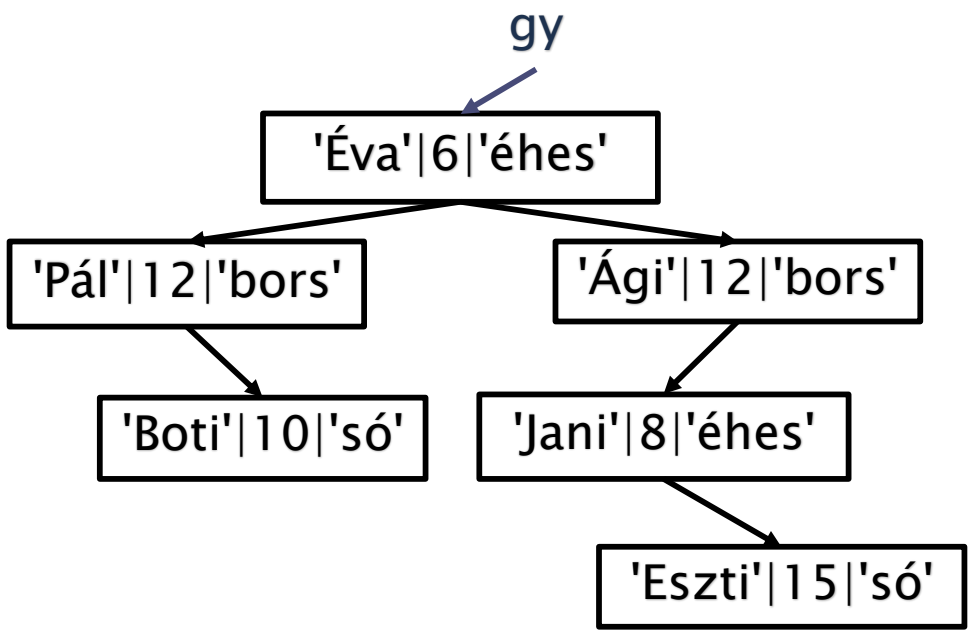


# Binárisfa gyakorlás

- Írj függvényt, mely **megjeleníti** egy paraméterben adott csapat tagjainak eredményeit (név, gombócok száma).

```
# LÉTREHOZÁS:
class Faelem:
  def __init__(self, n, c):
     self.nev = n
     self.gomboc = 0
     self.csapat = c
     self.bal = None
     self.jobb = None

gy = None
```

```
def csapattag_eredmeny(gy, csapat): # gy:Faelem, nev:str
    # üres (rész)fa esete
    if not gy:
        return

    # a gyökér feldolgozása
    if gy.csapat == csapat:
        print(gy.nev, gy.gomboc)

    # a bal és jobb oldali részfák feldolgozása
    csapattag_eredmeny(gy.bal, csapat)
    csapattag_eredmeny(gy.jobb, csapat)
```

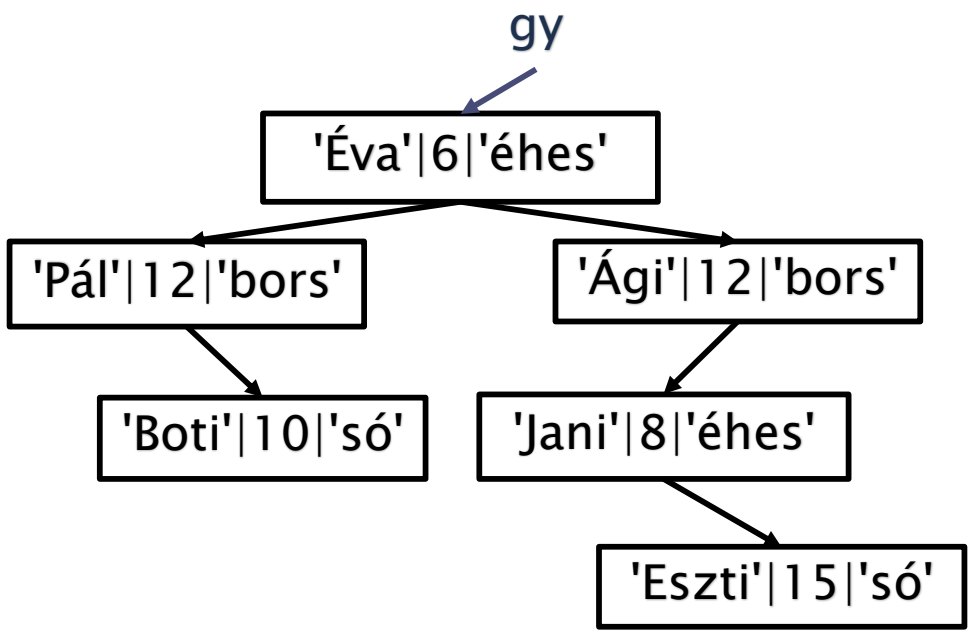


# Binárisfa gyakorlás

- Az 'só' csapat minden tagja megevett újabb két gombócot. Frissítsd az eredményeiket.

```
def kettot_evett(gy): # gy:Faelem, nev:str
    if not gy:
        return
    if gy.csapat == 'só':
        gy.gomboc += 2
    kettot_evett(gy.bal)
    kettot_evett(gy.jobb)
```

```
# LÉTREHOZÁS:
class Faelem:
  def __init__(self, n, c):
      self.nev = n
      self.gomboc = 0
      self.csapat = c
      self.bal = None
      self.jobb = None

gy = None
```

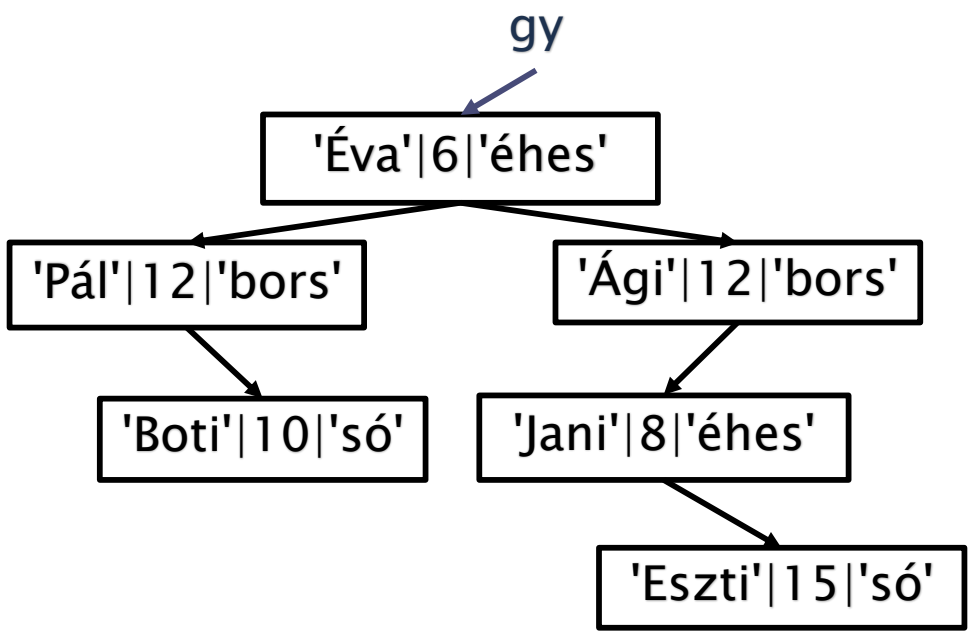


# Binárisfa gyakorlás

- Az "éhes" csapat csúnyán lemaradt a versenyben, ezért megvesztegették Eszti-t, hogy igazoljon át hozzájuk.
- A vesztegetés sikerrel járt, írj függvényt, mely Eszti csapatát 'éhes'-re cseréli.

```
# LÉTREHOZÁS:
class Faelem:
  def __init__(self, n, c):
     self.nev = n
     self.gomboc = 0
     self.csapat = c
     self.bal = None
     self.jobb = None

gy = None
```

```
def atigazol(gy): # gy:Faelem, nev:str
   if not gy:
       return
   if gy.nev == 'Eszti':
      gy.csapat = 'éhes'
      # a név egyedi, megtaláltuk => kiléphetünk
      return
   atigazol(gy.bal)
   atigazol(gy.jobb)
```



# Áttekintés

- Közös pontok az előző feladatokban
  - Az elemek bejárására építettünk
  - Nem volt visszatérési értéke a függvényeknek

- A megoldásaink sablonosak voltak:

```
def fv(gyökér és esetleges egyéb paraméterek):
    - üres (rész)fa kezelése
    - gyökérelem feldolgozása (a gyökeret mindig megkapunk paraméterben)
    - a függvényt meghívjuk a gyökér bal oldali gyerekére
    - a függvényt meghívjuk a gyökér jobb oldali gyerekére
```

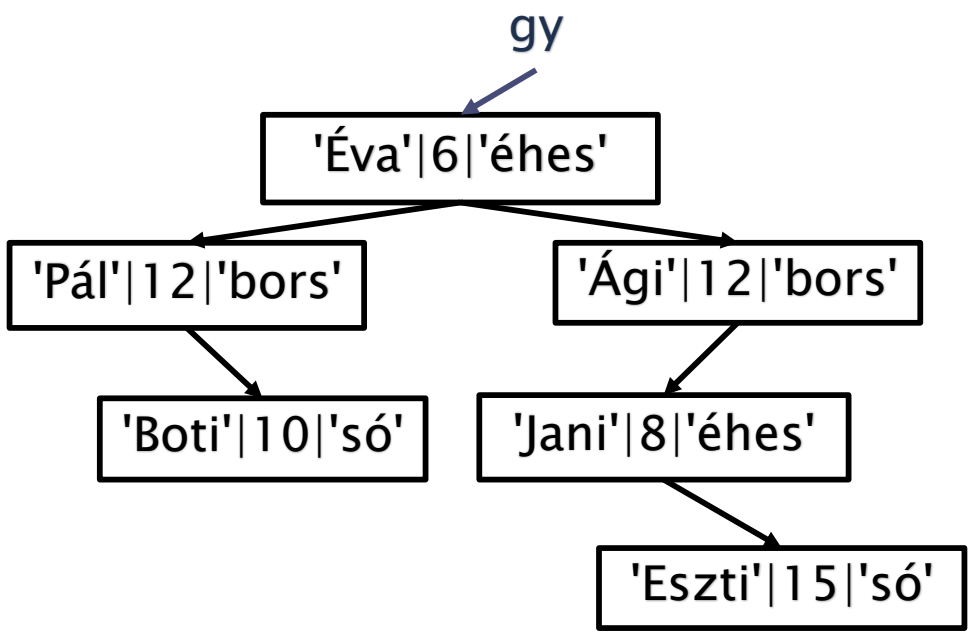
```
Megj.: A különböző részek végrehajtását néha feltételhez kötjük
    pl. ha egyetlen értéket kell módosítani és azt a részfa gyökerében megtaláltuk, akkor
    már nem szükséges meghívni a függvényt a bal és jobb oldali részfákra.
```

# Binárisfa gyakorlás

- Írj függvényt, mely **visszaadja**, hogy hány gombócot ettek meg összesen a résztvevők.

```
# LÉTREHOZÁS:
class Faelem:
  def __init__(self, n, c):
     self.nev = n
     self.gomboc = 0
     self.csapat = c
     self.bal = None
     self.jobb = None

gy = None
```

```
def ossz_gomboc(gy):  gy a (rész)fa gyökere, Faelem típusú
    if not gy:       # üres fa esete
        return 0

    # a gyökében lévő versenyző ennyi gombócot evett
    ossz_gy = gy.gomboc

    # a bal és jobb oldali részfákra kapott érték
    ossz_b = ossz_gomboc(gy.bal)
    ossz_j = ossz_gomboc(gy.jobb)

    return ossz_gy + ossz_b + ossz_j
```

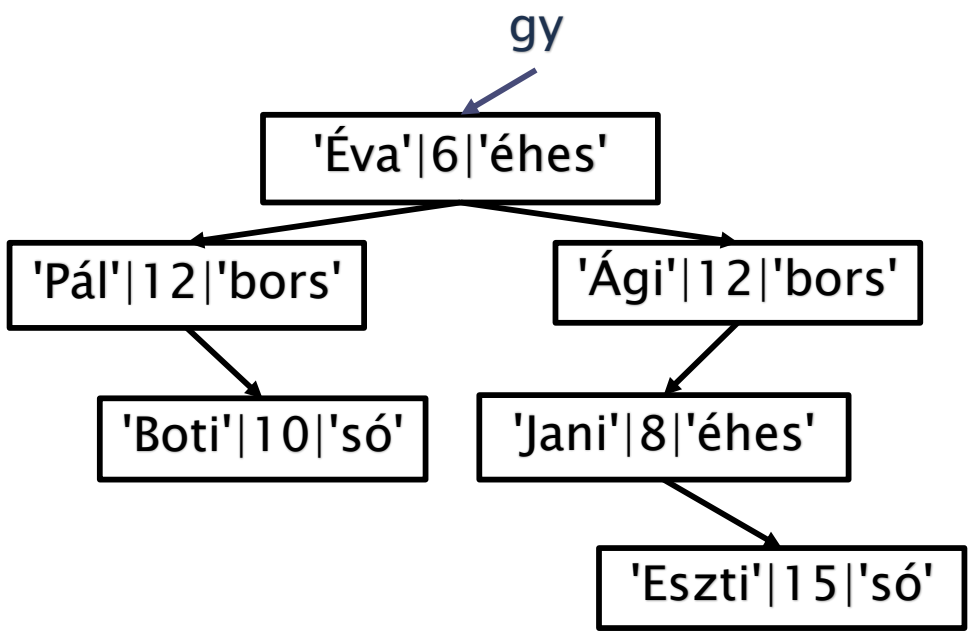


# Binárisfa gyakorlás

```
# LÉTREHOZÁS:
class Faelem:
  def __init__(self, n, c):
     self.nev = n
     self.gomboc = 0
     self.csapat = c
     self.bal = None
     self.jobb = None

gy = None
```

- Ugyanaz rövidebben:

```
def ossz_gomboc(gy):  gy a (rész)fa gyökere, Faelem típusú
    if not gy:
        return 0
    return gy.gomboc + ossz_gomboc(gy.bal) + ossz_gomboc(gy.jobb)
```

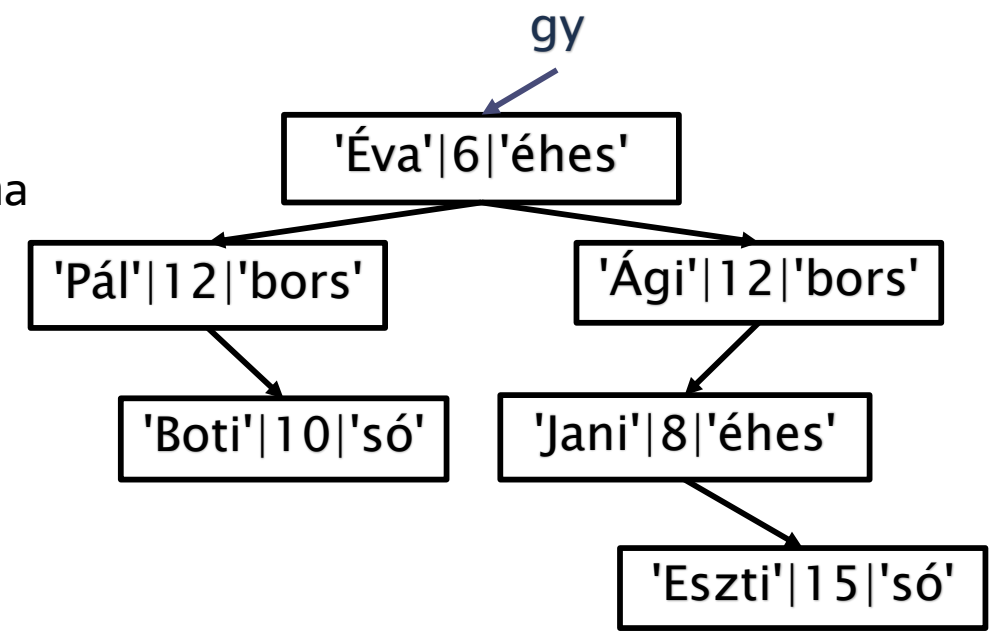


# Binárisfa gyakorlás

- Írj függvényt, mely **visszaadja**, hogy hány résztvevő van.
  /a fában lévő elemek száma/

```
# LÉTREHOZÁS:
class Faelem:
  def __init__(self, n, c):
     self.nev = n
     self.gomboc = 0
     self.csapat = c
     self.bal = None
     self.jobb = None

gy = None
```

```
def resztvevok(gy):  gy a (rész)fa gyökere, Faelem típusú
    if not gy:       # üres fa esete
        return 0

    # a gyökér elem tartalmaz 1 résztvevőt
    fo_gy = 1

    # a bal és jobb oldali részfákban a résztvevők száma
    fo_b = resztvevok(gy.bal)
    fo_j = resztvevok(gy.jobb)

    return fo_gy + fo_b + fo_j
```

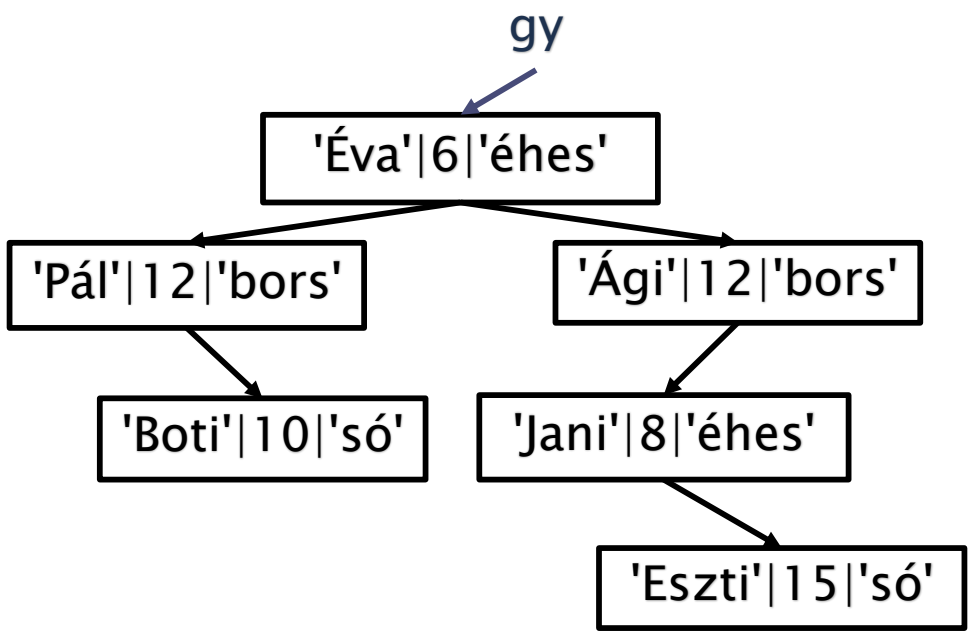


# Binárisfa gyakorlás

```
# LÉTREHOZÁS:
class Faelem:
  def __init__(self, n, c):
     self.nev = n
     self.gomboc = 0
     self.csapat = c
     self.bal = None
     self.jobb = None

gy = None
```

- Ugyanaz rövidebben:

```
def resztvevok(gy):  gy a (rész)fa gyökere, Faelem típusú
    if not gy:       # üres fa esete
        return 0

    return 1 + resztvevok(gy.bal)+resztvevok(gy.jobb)
```

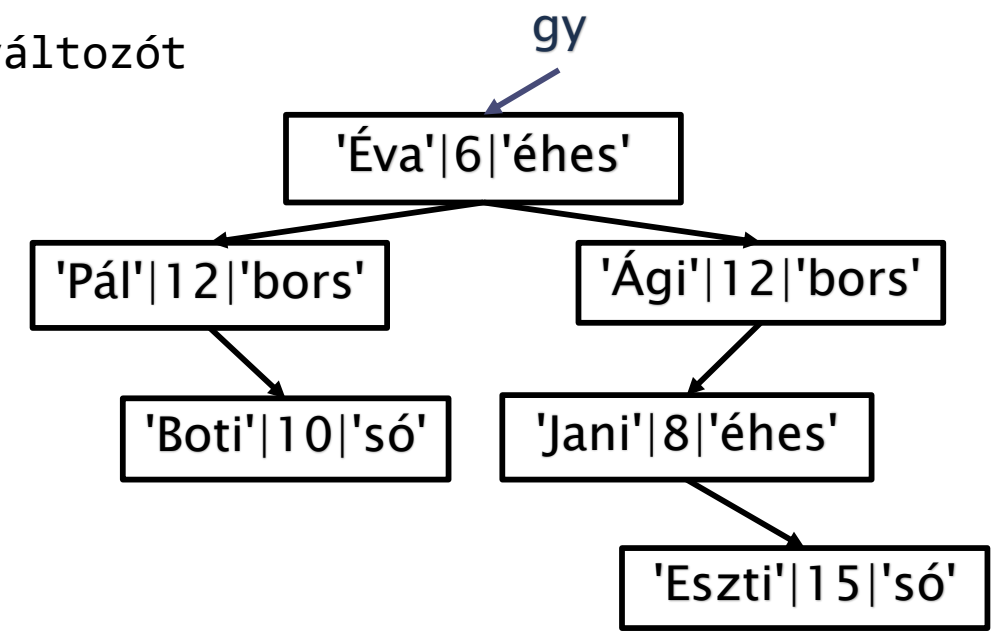


# Binárisfa gyakorlás

- Írj függvényt, mely **visszaadja**, hogy hány gombócot evett meg összesen a *csapat* paraméterben átadott nevű csapat.

```
# LÉTREHOZÁS:
class Faelem:
  def __init__(self, n, c):
     self.nev = n
     self.gomboc = 0
     self.csapat = c
     self.bal = None
     self.jobb = None

gy = None
```

```
def csapat_eredmeny(gy, csapat):  # gy: Faelem típusú
    if not gy:       # üres fa esete
        return 0
    # tegyük fel, hogy a gyökérben nem az a csapat van
    ossz_gy = 0
    # ha mégis a megfelelő csapat tagja, módosítjuk a változót
    if gy.csapat == csapat:
       ossz_gy = gy.gomboc
    # a bal és jobb oldali részfákra az eredmény
    ossz_b = csapat_eredmeny(gy.bal, csapat)
    ossz_j = csapat_eredmeny(gy.jobb, csapat)

    return ossz_gy + ossz_b + ossz_j
```

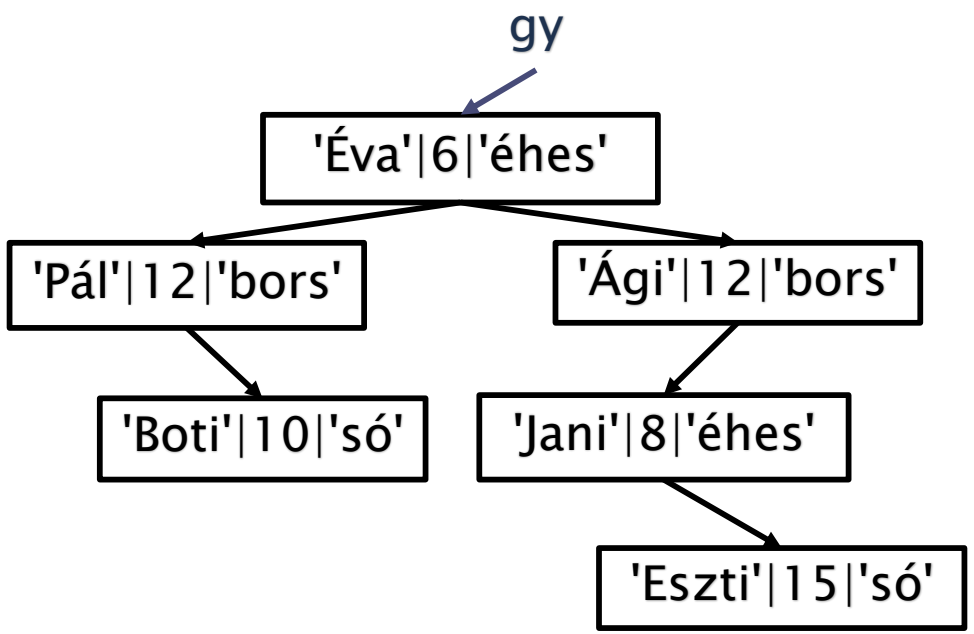


# Binárisfa gyakorlás

- Ugyanaz rövidebben

```
def csapat_eredmeny(gy, csapat):  # gy: Faelem típusú
    if not gy:       # üres fa esete
        return 0

    return csapat_eredmeny(gy.bal, csapat) +
           csapat_eredmeny(gy.jobb, csapat) +
           (gy.gomboc if gy.csapat == csapat else 0)
```

```
# LÉTREHOZÁS:
class Faelem:
  def __init__(self, n, c):
     self.nev = n
     self.gomboc = 0
     self.csapat = c
     self.bal = None
     self.jobb = None

gy = None
```

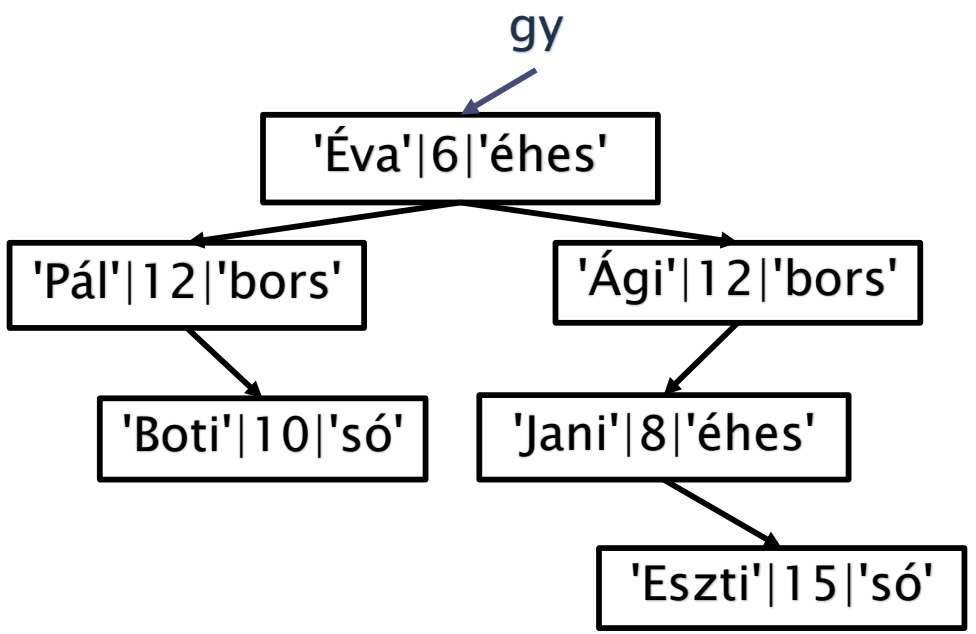


# Binárisfa gyakorlás

- Írj függvényt mely visszaadja, hogy Eszti hány gombócot evett meg.
- Ha Eszti nincs a fában, akkor -1 értéket adj vissza.

```
# LÉTREHOZÁS:
class Faelem:
  def __init__(self, n, c):
     self.nev = n
     self.gomboc = 0
     self.csapat = c
     self.bal = None
     self.jobb = None

gy = None
```

```
def megevett(gy):  # gy: Faelem típusú
    if not gy:      # üres fa esete
        return -1
    # megtaláltuk
    if gy.nev == 'Eszti':
        return gy.gomboc
    # keresés a bal oldali részfában
    ertek = megevett(gy.bal)
    if ertek == -1: # Eszti nem volt a baloldali fában
       ertek = megevett(gy.jobb) # keressük a jobbon
    return ertek
```

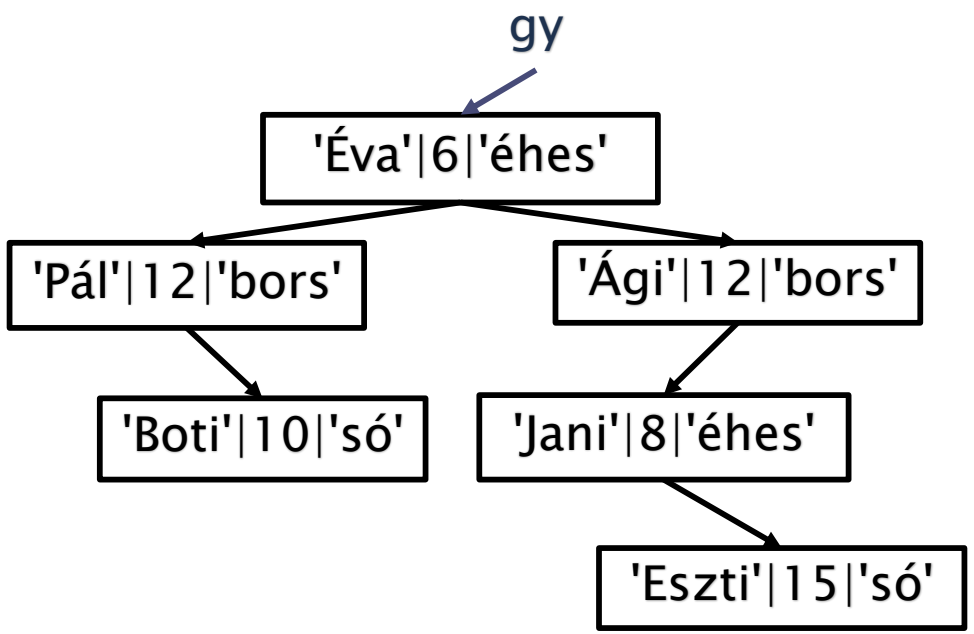


# Binárisfa gyakorlás

- Írj függvényt mely visszaadja, hogy hányan ettek 10-nél több gombócot.

```
def sokat_evett(gy):  # gy: Faelem típusú
    if not gy:       # üres fa esete
        return 0
    # vagy 0 vagy 1 lehet az éppen vizsgált csúcsban
    fo_gy = 0
    if gy.gomboc > 10:
        fo_gy = 1
    # a bal és jobb oldali részfákra az eredmény
    fo_b = sokat_evett(gy.bal)
    fo_j = sokat_evett(gy.jobb)
    return fo + fo_b+ fo_j
```

```
# LÉTREHOZÁS:
class Faelem:
  def __init__(self, n, c):
     self.nev = n
     self.gomboc = 0
     self.csapat = c
     self.bal = None
     self.jobb = None

gy = None
```

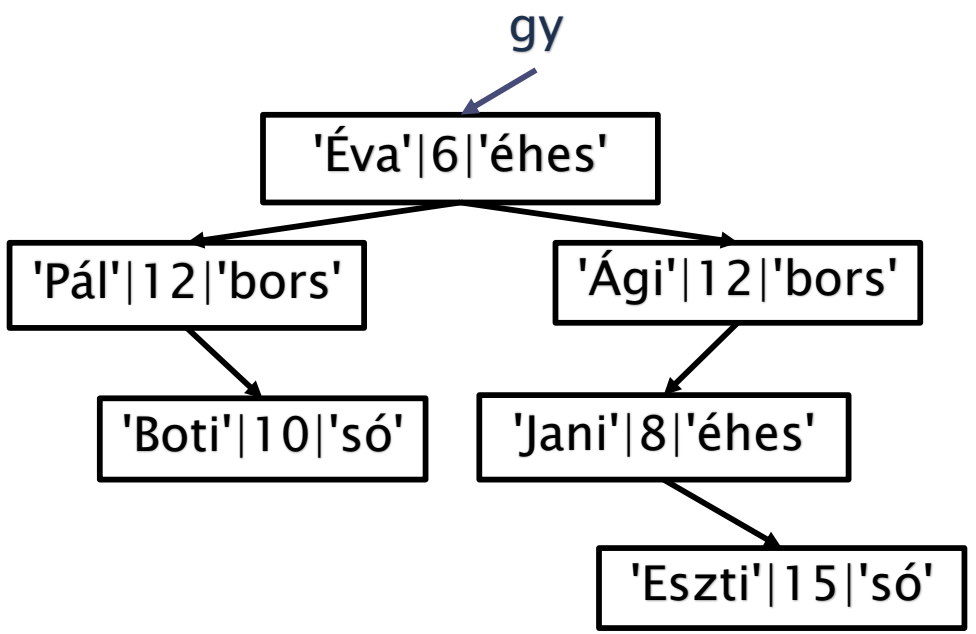


# Binárisfa gyakorlás

- Ugyanaz tömörebben

```
def sokat_evett(gy):  # gy: Faelem típusú
    if not gy:       # üres fa esete
        return 0
    return (1 if gy.gomboc>10 else 0) +
           sokat_evett(gy.bal) + sokat_evett(gy.jobb)
```

```
# LÉTREHOZÁS:
class Faelem:
  def __init__(self, n, c):
     self.nev = n
     self.gomboc = 0
     self.csapat = c
     self.bal = None
     self.jobb = None

gy = None
```

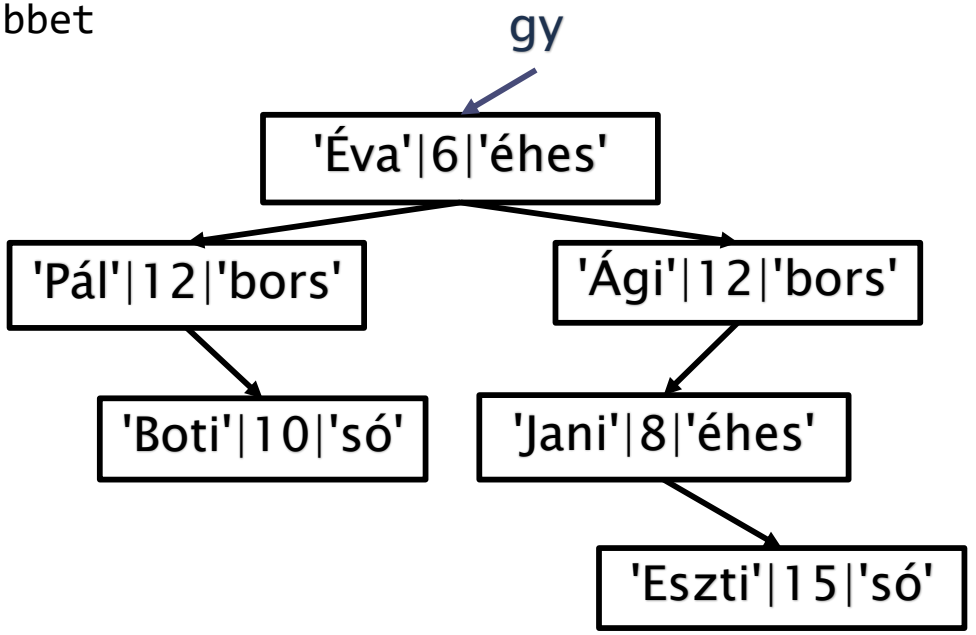


# Binárisfa gyakorlás

```
# LÉTREHOZÁS:
class Faelem:
  def __init__(self, n, c):
     self.nev = n
     self.gomboc = 0
     self.csapat = c
     self.bal = None
     self.jobb = None

gy = None
```

- Írj függvényt mely visszaadja, leggyengébb versenyző teljesítményét.
- Üres fa esetén 100-as értéket adj vissza.
  (Mivel összesen 100 gombócot főztünk a versenyre, ezért biztos, hogy ennél kevesebb lesz a minimum.)

```
def legkevesebb(gy):  # gy: Faelem típusú
    if not gy:      # üres fa esete
        return 100
    min = gy.gomboc     # tf. a gyökérben lévő ette a legkevesebbet
    minb = legkevesebb(gy.bal)
    minj = legkevesebb(gy.jobb)

    # ha mégsem, akkor módosítjuk a min-t
    if minb < min:
       min = minb
    if minj < min:
       min = minj
    return min
```



# Áttekintés

- Közös pontok az előző feladatokban
  - Az elemek bejárására építettünk
  - Volt visszatérési értéke a függvényeknek
  - A rekurzív hívásnál MINDIG felhasználtuk a kapott visszatérési értéket

- A megoldásaink "sablonja"

```
def fv(gyökér és esetleges egyéb paraméterek):
    - üres (rész)fa esetén visszaadjuk a megfelelő értéket
    - meghatározzuk a gyökérre vonatkozó értéket (mentjük az értéket)
    - a függvényt meghívjuk a gyökér bal oldali gyerekére (mentjük az értéket)
    - a függvényt meghívjuk a gyökér jobb oldali gyerekére (mentjük az értéket)
    - a három értéket felhasználva meghatározzuk és visszaadjuk az eredményt
```

# Megjegyzések

- Az értékek változóba mentése helyett gyakran egy kifejezésben összesíthetjük a gyökérre, bal- és jobb oldali részfára vonatkozó eredményt, így tömörebb a kód.

- Az "összesítés" alatt az érték felhasználását értjük, ami teljesen problémafüggő:
  lehet, hogy összegezzük, szorozzuk, a legnagyobbat, legkisebbet vesszük a három közül, stb.

- A bal és jobb oldali eredmények lekérdezését is feltételhez köthetjük.

  Lásd: keresés

  pl.: ha már megtaláltunk az értéket, a gyökérnél, akkor egyik részfában sem keresünk tovább

  ha megtaláltuk az elemet a bal oldali részfában, akkor nem keresünk a jobb oldalon

Természetesen vannak ennél jóval összetettebb problémák is, melyeket nem lehet ilyen sablonosan megoldani.