# Diszkrét matematika
# 3. előadás

MÁRTON Gyöngyvér
https://ms.sapientia.ro/∼mgyongyi/
mgyongyi@ms.sapientia.ro

Sapientia EMTE,
Matematika-Informatika Tanszék
Marosvásárhely, Románia

2022, őszi félév

# Miről volt szó az elmúlt előadáson?

- Python: az `in` operátor, logikai kifejezések, függvények, a `tuple`, a `list`, az `str` adattípus, szeletelések, adatbevitel, print - a kimenet formázása,
- Algoritmusok:
  - tesztelés: egy szám négyzetszám-e,
  - osztók száma, osztók meghatározása,
  - a faktoriális függvény - iteratív, rekurzív változatok,
  - nullások száma a faktoriális végén,
  - moduláris hatványozás,
  - minimum keresés

# Miről lesz szó?

- Python: hibakezelés, függvények paraméterátadása, szövegállományok, a `strip, split, zip` függvények,
- természetes számok, egész számok, racionális számok,
- algoritmusok futási ideje,
- maradékos osztás, legnagyobb közös osztó,
- algoritmusok:
  - gyorshatványozás - iteratív, rekurzív változatok, szorzások száma
  - adott kritérium szerinti kiválogatás,

# Python hibakezelés

Ha az adabevitel során nem megfelelő értéket olvasunk be egy adott változóba, akkor futási hiba adódik:

```
>>> beolvasFloat(4)
    elem = szoveg
    ...
    ValueError: could not convert string to float: 'szoveg'
```

Az ilyen típusú hibák elkerülése végett a megfelelő műveletsorokat egy `try...except` utasításblokk közé kell írni:

```
def beolvasFloat(n):
    L = []
    for i in range(n):
        try:
            L += [float(input("elem = "))]
        except:
            print("hibas bemenet")
            return
    return L
```

- az `try` részbe azokat a műveletsorokat kell írni, amelyek futási hibát okozhatnak,
- az `except` részbe pedig a hibaüzenetet, illetve a függvényből való kilépést kell megadni.

# Python függvények paraméterátadása

```
def foF1():
    db = 0
    segedF1(db)
    print (db)

def segedF1(x):
    x += 10
```

```
>>> foF1()
    0
```

A `foF1` függvényben nem látjuk, hogy a `segedF1` függvényben megváltoztattuk az `X` paraméter értékét.

```
def foF2():
    db = 0
    db = segedF2(db)
    print (db)

def segedF2(x):
    x += 10
    return x
```

```
>>> foF2()
    10
```

A `foF2` függvényben látjuk, hogy a `segedF2` függvényben megváltoztattuk az `X` paraméter értékét.

# Számok

- számtartományok: természetes számok, egész számok, racionális számok, irracionális számok, valós számok, komplex számok,
- alapműveletek számokkal: összegzés, különbség, szorzás, osztás, hatványozás, logaritmálás,
- természetes számok, egész számok: a diszkrét matematika gerincét alkotják,

# Természetes számok

- halmazjelölés: $\mathbb{N} = \{0, 1, 2, 3, \dots\}$,
- tulajdonságok:
  - az összeadás, szorzás kommutatív: $a, b \in \mathbb{N}, a + b = b + a, a \cdot b = b \cdot a$
  - az összeadás, szorzás asszociatív: $a, b, c \in \mathbb{N}, (a + b) + c = a + (b + c), (a \cdot b) \cdot c = a \cdot (b \cdot c)$
  - az összeadás a szorzásra nézve disztributív: bármely $a, b, c \in \mathbb{N}, (a + b) \cdot c = a \cdot c + b \cdot c$
  - a természetes számok halmaza zárt az összeadásra, szorzásra nézve: bármely két természetes szám összeadható, szorozható az eredmény szintén természetes szám lesz, ez nem igaz a különbség és osztás műveletekre,
  - a 0 az összeadásra nézve semleges, az 1 a szorzásra nézve semleges elem,
  - jól rendezettség: a természetes számok minden nem üres részhalmazának van egy legkisebb eleme.

# A hatványozás algoritmusa

1. feladat

*Írjunk egy Python függvényt, amely meghatározza $x^n$ értékét.*

- többféle megoldás létezik,
- a legkézenfekvőbb algoritmus az, ha `n`-szer összeszorozzuk az `x`-et, ezt a gondolatmenetet követi a `myPowLin` függvény:

```
def myPowLin(x, n):
    res = 1
    for i in range(n):
        res *= x
    return res
```

- az algoritmus futási ideje azonban nem jó, nagy számokra lassú,
- lemérhetjük a futási időt különböző bemenetekre: ha a hatványkitevő 500000 akkor már több másodpercig is eltarthat a számolási idő,
- megszámolhatjuk hány alapműveletet (szorzást) végez az algoritmus: ha a hatványkitevő 100, akkor 100 szorzást végez,
- az algoritmus futási ideje lineáris.

# A gyorshatványozás algoritmusa

- $x^n$ értékét meghatározhatjuk a gyorshatványozás algoritmusával: ha a hatványkitevő 100, akkor 9 darab szorzást végez az algoritmus,
- az algoritmus futási ideje logaritmikus

```
def myPow(x, n):
    res = 1
    while n != 0:
        if n % 2 == 1:
            res = res * x
        n = n // 2
        x = x * x
    return res
```

```
>>> myPow(2, 100)
    1267650600228229401496703205376L
```

```
def myPow1(x, n):
    res = 1
    while True:
        if n % 2 == 1:
            res = res * x
        n = n // 2
        if n == 0: break
        x = x * x
    return res
```

# A gyorshatványozás algoritmusa

$3^{100}$ meghatározása:

```
def myPow(x, n):
    res = 1
    while n != 0:
        if n % 2 == 1: res = res * x
        n = n // 2
        x = x * x
    return res
```

| $x$ | $n$ | $res$ |
|---|---|---|
| | | 1 |
| 3 | 100 | 1 |
| $3^2 = 9$ | 50 | 1 |
| $3^4 = 81$ | 25 | 81 |
| $3^8 = 6561$ | 12 | 81 |
| $3^{16} = 43046721$ | 6 | 81 |
| $3^{32} = 1853020188851841$ | 3 | 150094635296999121 |
| $3^{64} = 3433683820292512484657849089281$ | 1 | 515377520732011331036461129765621272702107522001 |
| $3^{128} = 117\ldots961$ | 0 | |

Az eredmény: $3^{100} = 3^4 \cdot 3^{32} \cdot 3^{64} = 515377520732011331036461129765621272702107522001$.

# A gyorshatványozás algoritmusa - rekurzív változatok

A gyorshatványozás algoritmusának rekurzív változatai:

```
def myPowRek1(x, n):
    if n == 0: return 1
    if n % 2 == 1: return x * myPowRek1(x * x, n // 2)
    return myPowRek1 (x * x, n // 2)

def myPowRek2(x, n):
    if n == 0: return 1
    temp = myPowRek2(x, n // 2)
    if n % 2 == 1: return x * temp * temp
    else: return temp * temp
```

# Futási idők

**2. feladat**

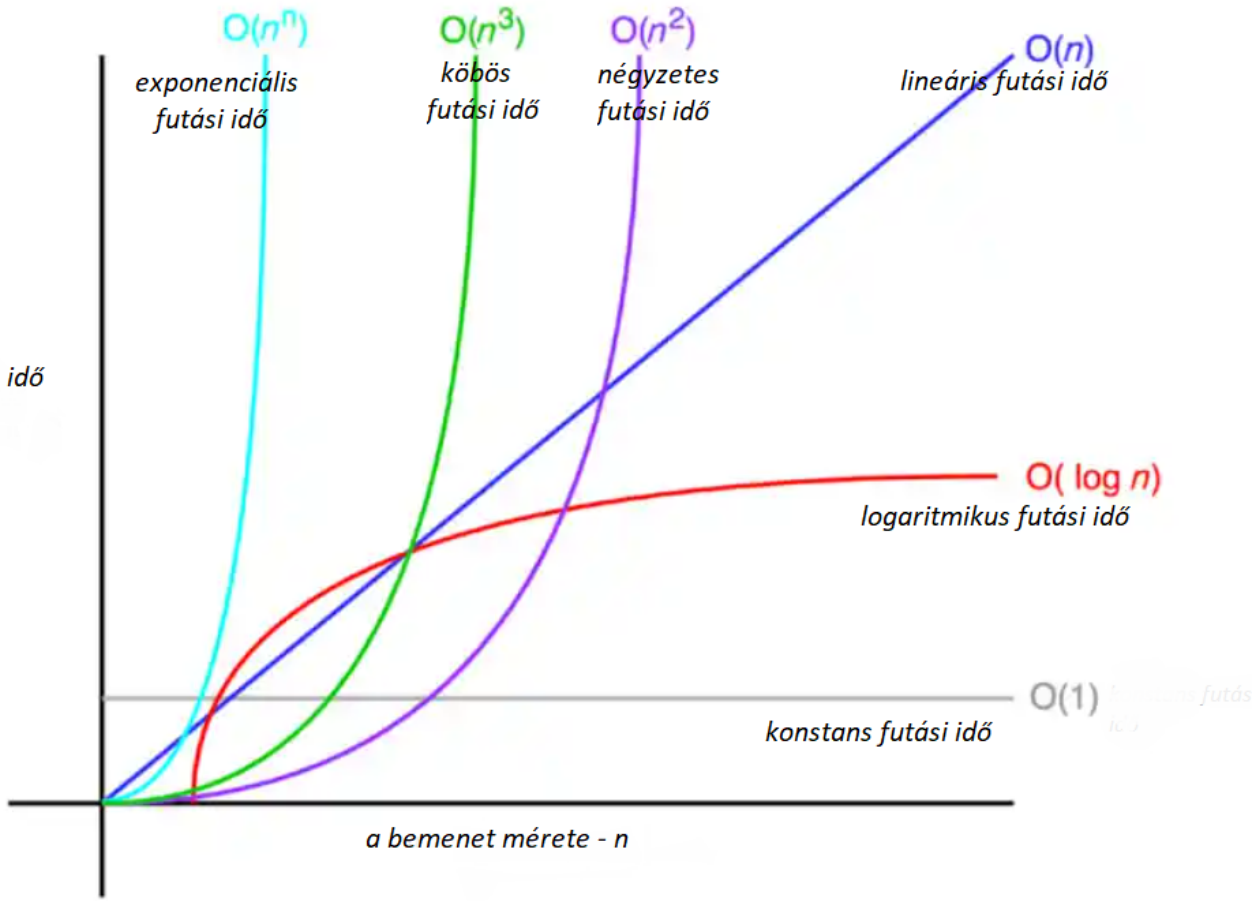
*Írjunk Python függvényt, amely meghatározza különböző nagyságrendű bemenetek esetén a `myPowLin` és `myPow` függvények futási idejét.*

```
from time import time

def mainTimeLog(x):
    for n in range(1, 10):
        st = time()
        myPow(x, 10**n)
        fs = time()
        print("kitevo: 10^",n, ", futasi ido:", fs - st)

def mainTimeLin(x):
    for n in range(1, 7):
        st = time()
        myPowLin(x, 10**n)
        fs = time()
        print("kitevo: 10^",n, ", futasi ido:", fs - st)
```

# Futási idők


O(n^n)
exponenciális futási idő
O(n^3)
köbös futási idő
O(n^2)
négyzetes futási idő
O(n)
lineáris futási idő
idő
O( log n)
logaritmikus futási idő
O(1)
konstans futási idő
a bemenet mérete - n

# A gyorshatványozás algoritmusa - szorzások száma

**3. feladat**

*Módosítsuk a `myPow1` függvényt, úgy hogy a műveletvégzések során elvégezett szorzások számát is meghatározza az algoritmus.*

```
def myPow2(x, n):
    res, db = 1, 0
    while True:
        if n % 2 == 1:
            res = res * x
            db += 1
        n = n // 2
        if n == 0: break
        x = x * x
        db += 1
    return db, res
```

```
def SzorSzama (x, n):
    db, res = myPow2(x, n)
    print ('Szorzasok szama: ', db)
    #print(res)
```

A `myPow2` függvény visszatérési értéke egy tuple típusú érték (számpár), ahol a kerek zárójeleket elhagytuk.

# A gyorshatványozás algoritmusa - szorzások száma

```
>>> SzorSzama(2, 1023)
    Szorzasok szama:  20

>>> SzorSzama(2, 1024)
    Szorzasok szama:  12

>>> SzorSzama(2, 1025)
    Szorzasok szama:  13
```

Miért van ekkora eltérés a szorzások számát illetően, a három különböző hatványkitevő esetében?

Meghatározzuk a kitevők kettes számrendszerbeli alakját, használjuk a beépített `bin` függvényt:

```
>>> bin(1023)
    '0b1111111111'
>>> bin(1024)
    '0b10000000000'
>>> bin(1025)
    '0b10000000001'
```

A szorzások száma egyenlő a hatványkitevő kettes számrendszerbeli alakjában szereplő 1-es bitek számával plusz a kettes számrendszerbeli ábrázoláshoz szükséges bitek számával!!

# Egész számok

- halmazjelölés: $\mathbb{Z} = \{\dots, -3, -2, -1, 0, 1, 2, 3, \dots\}$,
- tulajdonságok:
  - $\mathbb{N} \subset \mathbb{Z}$, és a két halmaz számossága ugyanaz,
  - két halmaz számossága, akkor egyezik meg, ha a két halmaz között létezik egy bijekció, pl: $f(x) = \begin{cases} 2x & \text{ha } x \geq 0, \\ -2x - 1 & \text{ha } x < 0, \end{cases}$
  - kommutatívitás, asszociativitás, disztributivitás,
  - az egész számok halmaza zárt az összeadásra, kivonásra, szorzásra nézve, de ez nem igaz az osztásra,
  - az összeadásra nézve minden elemnek van inverz eleme,
  - rendezettség: $a \leq b$, ha $b - a \in \mathbb{N}$.

# Adott kritérium szerinti kiválogatás

**4. feladat**

*A `hL` a hónapok neveit, a `homL` a hónapokhoz tartózó hőmérsékleti értékeket tárolják. Írjunk egy Python függvényt, amely meghatározza, azokat a hónapokat, amikor negatív volt a hőmérséklet.*

```
hL = ['januar','februar','marcius','aprilis','majus', 'junius',
        'julius','augusztus','szeptember','oktober','november','december']
homL = [-7,-5,-1,4,8,10,12,12,9,4,-1,-5]

def homerseklet1(honapL, homersekletL):
    resL = []
    for i in range(12):
        if homersekletL[i] < 0:
            resL += [honapL[i]]
    return resL
```

A függvény meghívása:

```
>>> homerseklet1(hL, homL)
    ['januar', 'februar', 'marcius', 'november', 'december']
```

# Adott kritérium szerinti kiválogatás

**5. feladat**

*A `hL` a hónapok neveit, a `homL` a hónapokhoz tartózó hőmérsékleti értékeket tárolják. Írjunk egy Python függvényt, amely meghatározza, azokat az őszi hónapokat, amikor negatív volt a hőmérséklet.*

```
def homerseklet2(honapL, homersekletL):
    osziL = honapL[8:11]
    resL = []
    for i in range(12):
        if homersekletL[i] < 0 and honapL[i] in osziL:
            resL += [honapL[i]]
    return resL
```

A függvény meghívása:

```
>>> homerseklet2(hL, homL)
    ['november']
```

# Adott kritérium szerinti kiválogatás

6. feladat

*A `tLista` értékpárokat tartalmaz, amelyek hónapok neveit, illetve hónapokhoz tartózó hőmérsékleti értékeket jelölnek. Írjunk egy Python függvényt, amely meghatározza, azokat a hónapokat, amikor negatív volt a hőmérséklet.*

```
tLista = [('januar', -7),('februar', -5),('marcius', -1),('aprilis', 4),
 ('majus',8), ('junius', 10), ('julius',12),('augusztus',12),
 ('szeptember',9), ('oktober', 4), ('november',-1,), ('december',-5)]

def homerseklet3(L):
    resL = []
    for elem in L:
        honap, homerseklet = elem
        if homerseklet < 0:
            resL += [honap]
    return resL

>>> homerseklet3(tLista)
    ['januar', 'februar', 'marcius', 'november', 'december']
```

# Szövegállományok

7. feladat

*A `homerseklet.txt` a hónapok neveit, illetve a hónapokhoz tartózó hőmérsékleti értéket tárolja. Írjunk egy Python függvényt, amely beolvassa az adatokat az állományból, és meghatározza, hogy mely hónapokban volt negatív a hőmérséklet.*

A `homerseklet.txt` tartalma legyen a következő, ahol minden hónapnév külön sorban van megadva, úgy hogy a hónapok, és a hónaphoz tartozó hőmérsékleti értékek között egy *szóköz* van:

```
januar -7
februar -5
marcius -1
aprilis 4
majus 8
junius 10
julius 12
augusztus 12
szeptember 9
oktober 4
november -1
december -5
```

# Szövegállományok, az adatok beolvasása

```
def beolvas():
    inf = open('homerseklet.txt', 'rt')
    L = []
    while True:
        temp = inf.readline()
        if not temp: break
        temp = temp.strip('\n')
        honap, homerseklet = temp.split(' ')
        L += [(honap, int(homerseklet))]
    inf.close()
    return L
```

- az állományban levő adatok feldolgozásához szükséges az állományt először megnyitni: `open`, majd a végén bezárni: `close`
- az adatok beolvasása soronként történik: `readline`, ahol a beolvasott érték típusa mindig `String`, szükség esetén ezt át kell alakítani `Int`, `Float` stb. típusúvá
- a `strip` függvény levágja az adott `String` típusú adat elejéről és végéről a paraméterként megadott karaktert
- a `split` függvény az adott `String` típusú adatot feldarabolja a megadott karakter mentén, rész-Stringeket hozva létre.

# Szövegállományok

- a beolvasás során egy, tuple típusú listát hozunk létre, ahol a tuple első eleme a hónapnevet, második eleme pedig a hőmérsékleti értéket jelöli
- az adatok feldolgozásához használhatjuk a `homerseklet1`, `homerseklet2`, illetve `homerseklet3` már megírt függvényeket, csak vigyázni kell a paraméterezésre
- a `homerseklet2` függvény használata előtt, például a tuple típusú listát, amelyet a `beolvas` függvény hoz létre, fel kell osztani két részlistára, ezt végzi a `zip` beépített függvény

```
def mainF1():
    L = beolvas()
    hL, homL = zip(*L)
    resL = homerseklet1(hL, homL)
    return resL

def mainF2():
    L = beolvas()
    resL = homerseklet3(L)
    return resL
```

Pythonban a * operátor használata sokrétű, bővebben itt lehet utána olvasni: *link*

# A zip, strip, split függvények

```
>>> hL = ['januar','februar','marcius','aprilis']
>>> homL = [-3, -1, 3, 5]
>>> list(zip(hL, homL))
[('januar', -3), ('februar', -1), ('marcius', 3), ('aprilis', 5)]

>>> L = [('januar', -3), ('februar', -1), ('marcius', 3), ('aprilis', 5)]
>>> list(zip(*L))
[('januar', 'februar', 'marcius', 'aprilis'), (-3, -1, 3, 5)]
```

Példa a strip és split használatára:

```
>>> '!Sapientia Egyetem!!'.strip('!')
'Sapientia Egyetem'
>>> 'Diszkret Matek\t\t'.strip('\t')
'Diszkret Matek'

>>> 'Diszkret Matek'.split()
['Diszkret', 'Matek']
>>> 'Diszkret\tMatek\tInformatika'.split('\t')
['Diszkret', 'Matek', 'Informatika']
```

# Racionális számok

A racionális számok halmazát: $\mathbb{Q}$-val jelöljük. Bármelyik eleme felírható: $\left\{ \frac{a}{b} : a, b \in \mathbb{Z}, b \neq 0 \right\}$ alakba. Tekinthetünk úgy is rájuk, mint egész számok rendezett párjaira. Tulajdonságok:

- az összeadás, szorzás kommutatív, asszociatív műveletek,
- az összeadás a szorzásra nézve disztributív művelet,
- a racionális számok halmaza zárt az összeadásra, kivonásra, szorzásra, osztásra nézve,
- az összeadásra, szorzásra nézve minden elemnek lesz inverz eleme,
- a racionális számok halmaza megszámlálható, azaz létezik egy számsorozat amely a racionális számokból áll,
- a racionális számok sűrűn rendezett halmazt alkotnak: bármely két racionális szám között van egy harmadik,
- minden racionális szám felírható tizedes alakba, azaz véges, vagy végtelen szakaszos tizedes jegyek segítségével.

# Racionális számok

Legyen $a, b \in \mathbb{Z}$, ahol $b \neq 0$. $a$ osztja $b$-t, ha létezik olyan $c$ egész szám, amelyre fennáll: $b = a \cdot c$, jelölése: $a \mid b$. Azt az esetet, amikor $a$ nem osztja $b$-t $a \nmid b$-vel jelöljük.

1. tétel (A maradékos osztás tétele)

*Legyen $a$ egy egész szám, $c$ pedig egy pozitív egész szám. Ekkor egyértelműen létezik két olyan $q$ és $r$ egész szám, amelyekre igaz $a = c \cdot q + r$, $0 \leq r < c$.*

- Az $a$ és $b$ egész számok legnagyobb közös osztója az a legnagyobb pozitív egész szám, amely osztja az $a$-t és $b$-t is. Jelölése: $lnko(a, b)$, $(a, b)$, vagy $\gcd(a, b)$.
- Az $a$ és $b$ egész számok legkisebb közös többszöröse az a legkisebb pozitív egész szám amely osztható $a$-val és $b$-vel is. Jelölése: $lkkt(a, b)$, vagy $[a, b]$.
- Az $a$ és $b$ egész számok szorzata egyenlő az $a$ és $b$ legnagyobb közös osztójának és legkisebb közös többszörösének a szorzatával: $(a, b) \cdot [a, b] = a \cdot b$.
- A racionális számok végtelen sok alakban felírhatóak, az egyik az irreducibilis alak, ami azt jelenti, hogy a számláló és nevező legnagyobb közös osztója 1.

# Racionális számok

- A legnagyobb közös osztó meghatározására több algoritmus is ismert, az egyik az Eukleidészi algoritmus. Python-ban a `math` modulban található `gcd` függvény használható két szám legnagyobb közös osztójának a meghatározására:

```
>>> from math import gcd
>>> gcd(60, 45)
    15
>>> gcd(1789, 100)
    1
>>> gcd(-63, 45)
    9
```

- Azt mondjuk, hogy $a$ és $b$ relatív prímek, ha a legnagyobb közös osztójuk 1. A fenti példában 1789 és 100 relatív prímek, míg 60 és 45, illetve -63 és 45 nem.

2. tétel

*Legyenek $a, b, q, r$ egész számok, $a = b \cdot q + r$ és legyen $(a, b) = d$, azaz $a$ és $b$ legnagyobb közös osztója $d$. Ekkor igaz az, hogy: $d = (a, b) = (b, a - b \cdot q)$.*