Diszkrét matematika
4. előadás

MÁRTON Gyöngyvér
mgyongyi@ms.sapientia.ro

Sapientia Egyetem,
Matematika-Informatika Tanszék
Marosvásárhely, Románia

2023, őszi félév

Miről volt szó az elmúlt előadáson?

- Python: hibakezelés, függvények paraméterátadása,
- természetes számok, egész számok,
- algoritmusok futási ideje,
- maradékos osztás,
- Algoritmusok:
  - minimum keresés,
  - gyorshatványozás - iteratív, rekurzív változatok, szorzások száma
  - a faktoriális függvény - iteratív, rekurzív változatok,
  - legnagyobb közös osztó - iteratív, rekurzív változatok,
  - a kiterjesztett euklideszi algoritmus,
  - diofantoszi egyenletek.

Miről lesz szó?

- Python:
  - a `strip`, `split`, `zip` függvények,
  - szövegállományok,
  - a `Fraction` típus, a `Decimal` típus,
- racionális számok, lánctörtek, irracionális számok,
- "híresebb" irracionális számok: $\sqrt{n}$,
- algoritmusok:
  - adott kritérium szerinti kiválogatás,
  - racionális számok irreducibilis alakja,
  - racionális számok sorba rendezése - két módszer,
  - racionális számok lánctört jegyei.

A strip, split függvények

a strip függvény levágja az adott String típusú adat elejéről és végéről a paraméterként megadott karaktert:

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

a split függvény az adott String típusú adatot feldarabolja a megadott karakter mentén, rész-Stringeket hozva létre:

```
>>> 'Diszkret Matek'.split()
    ['Diszkret', 'Matek']
>>> 'Diszkret\tMatek\tInformatika'.split('\t')
    ['Diszkret', 'Matek', 'Informatika']
>>> 'Diszkret Matek'.split('t')
    ['Diszkre', ' Ma', 'ek']
>>> 'Diszkret Matek'.split('te')
    ['Diszkret Ma', 'k']
```

A zip függvények és a * operátor

```
>>> hL =  ['januar','februar','marcius','aprilis']
>>> homL = [-3, -1, 3, 5]
>>> for h1, h2 in zip(hL, homL): # osszecipzarozzuk a ket listat
        print(h1, h2)

    januar -3
    februar -1
    marcius 3
    aprilis 5

>>> L = [('januar', -3), ('februar', -1), ('marcius', 3), ('aprilis', 5)]
>>> h1, h2 = zip(*L) # visszaallatjuk a cipzarozas elotti allapotot
>>> h1
    ('januar', 'februar', 'marcius', 'aprilis')
>>> h2
    (-3, -1, 3, 5)
```

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

Adott kritérium szerinti kiválogatás

1. 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

2. 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

3. 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

4. feladat

A `homerseklet.txt` a hónapok neveit, illetve a hónapokhoz tartozó 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
```

- a beolvas során egy, tuple típusú listát hozunk létre, ahol a tuple első eleme a hónapnevet, második eleme pedig a hőmérsékleti értéket jelöli
- 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 `str`, szükség esetén ezt át kell alakítani `Int`, `Float` stb. típusúvá

Szövegállományok

- az adatok feldolgozásához használhatjuk a `homerseklet1`, `homerseklet2`, illetve `homerseklet3` már megírt függvényeket, csak vigyázni kell a paraméterezésre,
- a `homerseklet1` függvény használata előtt, például a tuple típusú listát, amelyet a `beolvas` függvény hoz létre, fel kell osztani két részlistára:

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

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

Racionális számok

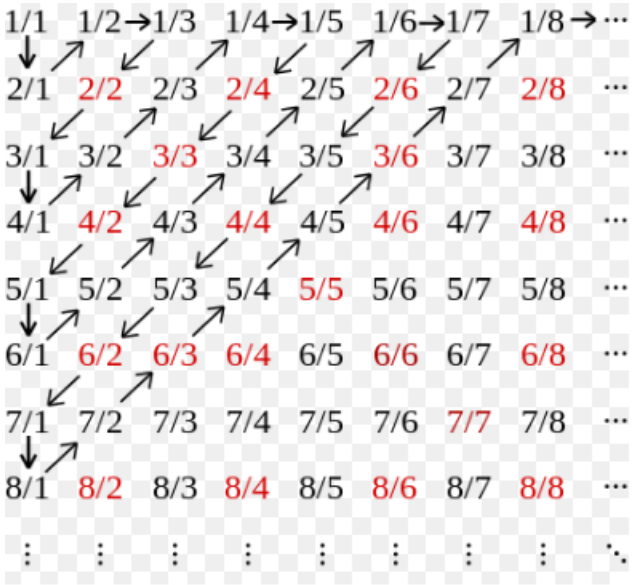
A racionális számok halmazát: $\mathbb{Q}$-val jelöljük. Bármelyik eleme felírható: $\left\{\frac{a}{b} : a, b \in \mathbb{Z}, b \neq 0\right\}$ alakba. Tekinthetünk úgy is rájuk, mint egész számok rendezett párjaira. Tulajdonságok:

- 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

A racionális számok halmaza megszámlálható:

- a racionális számok halmaza felsorolható, azaz létezik egy számsorozat, amelyet a racionális számok alkotnak : $r_1, r_2, \ldots, r_n, \ldots,$
- bármelyik racionális szám felírható p/q alakba
- a racionális számok generálásának egyik módszere, ha elindulunk a következő mátrix bal-felső sarkában található elemtől, majd a nyilakat követjük, pirossal azokat a számok jelennek meg, amelyek nem irreducibilis alakban vannak, korábban pedig már ki lettek generálva.



Racionális számok sorba rendezése

5. feladat

Írjunk egy Python függvényt, amely az előző oldalon megadott bejárási sorrend szerint bejárja a mátrixot és kiírja az első $\frac{n \cdot (n+1)}{2}$ *racionális számot.*

Az `auxRacionalis` függvény a `k`-ik átló kiíratását valósítja meg és aszerint, hogy páratlan vagy páros sorszámú átló kiíratásánál tart, a számláló értékeit növekvő, vagy csökkenő sorrendbe generálja ki.

```
def auxRacionalis(k):
    for j in range(1, k+1):
        if k % 2 == 1: print((j, k+1-j), end = ' ')
        else: print((k+1-j, j), end = ' ')
    print()

>>> auxRacionalis(4)
    (4, 1) (3, 2) (2, 3) (1, 4)
>>> auxRacionalis(5)
    (1, 5) (2, 4) (3, 3) (4, 2) (5, 1)
```

Racionális számok sorba rendezése

A `racionalis1` függvény `n`-szer fogja meghívni az `auxRacionalis` függvényt ahhoz, hogy a kívánt számosorzatot kiírja:

```
def racionalis1(n):
    for k in range(1, n+1):
        auxRacionalis(k)
```

```
>>> racionalis (10)
    (1, 1)
    (2, 1) (1, 2)
    (1, 3) (2, 2) (3, 1)
    (4, 1) (3, 2) (2, 3) (1, 4)
    (1, 5) (2, 4) (3, 3) (4, 2) (5, 1)
    ...
```

A feladat megadható egymásba ágyazott for ciklussal. Az `if` feltétel is elhagyható, mert a számok sorrendje az egyes sorokon belül nem számít:

```
def racionalis2(n):
    for k in range(1, n+1):
        for j in range(1, k+1):
            print((j, k+1-j), end = ' ')
        print()
```

Ha egy számpárból képezhető racionális szám alakja nem irreducibilis, akkor azt jelenti, hogy már egyszer ki volt generálva. Ennek a feltételnek a bevezetése házi feladat.

Racionális számok sorba rendezése

6. feladat

Írjunk egy Python függvényt, amely az előző oldalakon ismertetett módszerrel meghatározza egy listába az első n pozitív, racionális számot.

```
def racionalisL(n):
    L = []
    for k in range(1, n+1):
        for j in range(1, k+1):
            L += [(j, k+1-j)]
            n = n - 1
            if n == 0: return L
    return L

>>> racionalisL(7)
    [(1, 1), (1, 2), (2, 1), (1, 3), (2, 2), (3, 1), (1, 4)]
```

Racionális számok sorba rendezése

7. feladat

Írjunk egy Python függvényt, amely meghatározza egy listába az első n pozitív, racionális számot, alkalmazva a következő algoritmust: az első racionális szám $\frac{1}{1}$, az $\frac{x}{y}$ után következő racionális szám: $2 \cdot \left\lfloor \frac{x}{y} \right\rfloor + 1 - \frac{x}{y}$ reciproka, ahol $\lfloor\ \rfloor$ alsó egészrészt jelent.

A kódsorban az $\left\lfloor \frac{x}{y} \right\rfloor$ értéket // művelettel határozzuk meg, azaz osztási egészrészt fogunk számolni. A különbség pedig tört számok, azaz *tuple elemek* közötti különbséget jelent.

pl: $\frac{5}{2} \to 2 \cdot \left\lfloor \frac{5}{2} \right\rfloor + 1 - \frac{5}{2} = 2 \cdot 2 + 1 - \frac{5}{2} = 5 - \frac{5}{2} = \frac{10-5}{2} = \frac{5}{2} \to \frac{2}{5}$

pl: $\frac{5}{3} \to 2 \cdot \left\lfloor \frac{5}{3} \right\rfloor + 1 - \frac{5}{3} = 2 \cdot 1 + 1 - \frac{5}{3} = \frac{9-5}{3} = \frac{4}{3} \to \frac{3}{4}$

Ezzel a módszerrel a következő törteket kapjuk:
$\frac{1}{1}, \frac{1}{2}, \frac{2}{1}, \frac{1}{3}, \frac{3}{2}, \frac{2}{3}, \frac{3}{1}, \frac{1}{4}, \frac{4}{3}, \frac{3}{5}, \frac{5}{2}, \frac{2}{5}, \frac{5}{3}, \frac{3}{4}, \frac{4}{1}$, *stb.*

Racionális számok sorba rendezése

- a racionális számokat most is tuple típusú adatként fogjuk kezelni,
- `nextRac` meghatározza az $\frac{x}{y}$ után következő racionális számot,
- `racionalis3` a feladat főfüggvénye:

```
def racionalis3(n):
    if n < 1: return []
    L = [(1, 1)]
    x, y = 1, 1
    while n > 1:
        x, y = nextRac(x, y)
        L += [(x, y)]
        n = n - 1
    return L
```

```
def nextRac(x, y):
    nrx = (2 * (x // y) + 1) * y - x
    nry = y
    g = lnkoF(nrx, nry)
    return (nry // g, nrx // g)
```

az alkalmazott képlet:

$$2 \cdot \left\lfloor \frac{x}{y} \right\rfloor + 1 - \frac{x}{y} = \frac{\left(2 \cdot \left\lfloor \frac{x}{y} \right\rfloor + 1\right) \cdot y - x}{y}$$

```
>>> racionalis3(10)
    [(1,1), (1,2), (2,1), (1,3), (3,2), (2,3), (3,1), (1,4), (4,3), (3,5)]
```

A fractions modul

- A következőkben a feladat megoldásához a racionális számokat `Fraction` típusként fogjuk kezelni.
- A Python `Fraction` típusának a használatához szükséges importálni a `fractions` modult.
- Példák `Fraction` használatára:

```
from fractions import Fraction
>>> rac1 = Fraction(2,3)
>>> rac2 = Fraction(4,5)

>>> rac1 + rac2
    Fraction(22, 15)

>>> rac1 * rac2
    Fraction(8, 15)

>>> print('szamlalo: ', rac1.numerator)
    szamlalo:  2
>>> print('nevezo: ', rac1.denominator)
    nevezo:  3
```

A fractions modul

Az $\frac{x}{y}$ utáni racionális szám meghatározása így a kövekező lesz:

```
from fractions import Fraction
def nextRacFrac(r):
    x, y = r.numerator, r.denominator
    temp = 2 * (x // y) + 1
    res = Fraction(temp, 1) - Fraction(x, y)
    return Fraction(res.denominator,  res.numerator)

>>> racNr = Fraction(4,3)
>>> nextRacFrac(racNr)
    Fraction(3, 5)
```

A racionális számokat tartalmazó lista kigenerálásához hívjuk meg a `nextRacFrac` függvényt a `racionalis3` függvényben.

Lánctörtek (Continued fraction)

A lánctört egy *emeletes* tört, amely kétféle alakban is megadható, ahol a két alak átalakítható egymásba:

$$a_0 + \cfrac{b_1}{a_1 + \cfrac{b_2}{a_2 + \cfrac{b_3}{a_3 + \cfrac{b_4}{\ddots}}}} \qquad d_0 + \cfrac{1}{d_1 + \cfrac{1}{d_2 + \cfrac{1}{d_3 + \cfrac{1}{\ddots}}}}$$

A második alakot egyszerű lánctörtnek, a $[d_0, d_1, d_2, d_3, \dots]$ számsorozatot, pedig a lánctört jegyeinek hívjuk.

Ha a $[d_0, d_1, d_2, d_3, \dots]$ számsorozat véges számú elemet tartalmaz, akkor véges lánctörtről beszélünk.

A racionális számok mindegyike felírható egyszerű, véges lánctört alakba.

Lánctörtek, példa

Ha meg akarjuk határozni az $\frac{x}{y}$ racionális szám lánctörtjét, akkor meghatározzuk az `x` és `y` osztási egészrészét (`//`), illetve osztási maradékát (`%`), felülírjuk az `x` értékét `y`-al és az `y` értékét az osztási maradékkal, majd ismételjük a műveletsort, amíg az `x` és `y` osztási maradéka nem lesz 0.

Az algoritmus a legnagyobb közös osztó algoritmusának gondolatmenetét követi, amelynek során eltároljuk az osztási egészrészeket, ezek fogják képezni a lánctörtjegyeket.

Példa, $\frac{61}{47}$ lánctört jegyeinek a meghatározása:

x	y	$x//y$	$x\%y$
61	47	1	14
47	14	3	5
14	5	2	4
5	4	1	1
4	1	4	0

Lánctörtek, példa

$$\frac{61}{47} = 1 + \cfrac{1}{3 + \cfrac{1}{2 + \cfrac{1}{1 + \cfrac{1}{4}}}}$$

$$\frac{61}{47} = 1 + \cfrac{1}{3 + \cfrac{1}{2 + \cfrac{1}{1 + \cfrac{1}{3 + \cfrac{1}{1}}}}}$$

$\frac{61}{47}$ lánctört jegyei: $[1, 3, 2, 1, 4]$, vagy

$\frac{61}{47}$ lánctört jegyei: $[1, 3, 2, 1, 3, 1]$

$\frac{61}{47}$ tizedes alakja: $1.(29787234042553191489361702127659574468085106 38)$

Ha a lánctört utolsó jegye nem 1, akkor ez az x érték helyettesíthető két további értékkel: $x - 1, 1$. A két alak ekvivalens.

Lánctörtek, példa

Alakítsuk át $\frac{41}{11}$-t lánctörtté, hat. meg a lánctört jegyeket, és a tizedes alakot:

$$\frac{41}{11} = 3 + \cfrac{1}{1 + \cfrac{1}{2 + \cfrac{1}{1 + \frac{1}{2}}}}$$

$$\frac{41}{11} = 3 + \cfrac{1}{1 + \cfrac{1}{2 + \cfrac{1}{1 + \cfrac{1}{1}}}}$$

$\frac{41}{11}$ lánctört jegyei: $[3, 1, 2, 1, 2]$, vagy

$\frac{41}{11}$ lánctört jegyei: $[3, 1, 2, 1, 1, 1]$

$\frac{41}{11}$ tizedes alakja: $3.(72)$

Racionális számok lánctörtjegyei

8. feladat

Határozzuk meg az $\frac{x}{y}$ racionális szám tizedes alakját és lánctört jegyeit.

```
def lanct(x, y):
    tAlak = x / y
    L = []
    while True:
        temp = x // y
        L += [temp]
        r = x % y
        if r == 0: break
        x = y
        y = r
    return (tAlak, L)

>>> lanct(41, 11)
    (3.727272727272727, [3, 1, 2, 1, 2])

>>> lanct(89, 55)
    (1.6181818181818182, [1, 1, 1, 1, 1, 1, 1, 1, 2])
```

Racionális számok lánctörtjegyei

Átírjuk a maradékos osztást:

```
def lanct1(x, y):
    tAlak = x / y
    L = []
    while True:
        temp = x // y
        L += [temp]
        r = x - temp * y
        if r == 0: break
        x = y
        y = r
    return (tAlak, L)

>>> lanct(61, 47)
    (1.297872340425532, [1, 3, 2, 1, 4])
```

Racionális számok lánctörtjegyei

Ellenőrző műveleteket vezetünk be:

```
def lanct2(x, y):
    if not float(x).is_integer() or not float(y).is_integer():
        print ('a bemenet nem egesz szam')
        return
    L = []
    while True:
        temp = x // y
        L += [temp]
        r = x - temp * y
        if r == 0: break
        x = y
        y = r
    return L

>>> lanct2(41, 11.2)
    a bemenet nem egesz szam

>>> lanct2(41, 11.0)
    [3, 1, 2, 1, 2]
```

Irracionális számok

- halmazjelölés: $\mathbb{Q}^*$,
- azok a számok, melyek nem írhatóak fel két egész szám hányadosaként, azaz a végtelen, nem szakaszos tizedes törtek,
- "híresebb" irracionális számok:

$$\begin{aligned} \sqrt{2} &= 1.4142\ldots, \\ \pi &= 3.1415\ldots, \\ e &= 2.7182\ldots, \\ \phi &= \frac{1+\sqrt{5}}{2} = 1.6118\ldots, \text{az aranyarány.} \end{aligned}$$

- a számítástechnika az irracionális számok közelített értékét tudja kezelni
- az irracionális számok végtelen lánctörtek
- lánctörtek segítségével könnyedén meglehet határozni a közelített érték tízedesjegyeit

$\sqrt{n}$ értéke

9. feladat

Határozzuk meg $\sqrt{n}$ értékét lánctörtek segítségével.

A kiinduló képlet a következő, ahol a értéke egy akármilyen szám lehet:

$$\sqrt{n} = a + \frac{n - a^2}{a + \sqrt{n}}$$

ha $a = 1$, akkor: $\sqrt{n} = 1 + \cfrac{n-1}{2 + \cfrac{n-1}{2 + \cfrac{n-1}{2 + \cfrac{n-1}{2 + \dots}}}}$

$\sqrt{2}$ értéke

10. feladat

Határozzuk meg $\sqrt{2}$ értékét lánctörtek segítségével.

$$\sqrt{2} = 1 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \ldots}}}$$

```
def my_sqrt2_():
    temp = 0
    for x in range(0, 500):
        temp = 1 / (2 + temp)
    return 1 + temp

>>> my_sqrt2_()
    1.4142135623730951
>>> 2 ** 0.5
    1.4142135623730951
```

A decimal modul

Próbáljuk ki az alábbi műveleteket:

```
>>> 0.1 + 0.1 - 0.2
    0.0

>>> 0.1 + 0.1 + 0.1
    0.30000000000000004

>>> 0.1 + 0.1 + 0.1 - 0.3
    5.551115123125783e-17

>>> from math  import log
>>> log(10**6, 10)
    5.999999999999999
```

A Python bevezeti a `Decimal` típust, amely pontosabb számolást tesz lehetővé, a nem egész számok körében.

A decimal modul

A `Decimal` típus a `decimal` modulban van definiálva, és a felhasználó által óhajtott pontossággal ábrázolja a lebegőpontos (valós) számokat:

```
>>> from decimal import Decimal
>>> Decimal('0.1') + Decimal('0.1') + Decimal('0.1') - Decimal('0.3')
    Decimal('0.0')

>>> (Decimal(10)**6).log10()
    Decimal('6')

>>> 1/3
    0.3333333333333333

>>> Decimal(1)/3
    Decimal('0.33333333333333333333333333')
```

A `getcontext.prec()` segítségével a tizedes jegyek számát adhatjuk meg.

```
>>> from decimal import getcontext
>>> getcontext().prec = 20
>>> Decimal(1)/3
    Decimal('0.33333333333333333333')
```

A decimal modul

11. feladat

Határozzuk meg egy szám számjegyeinek számát, és a bináris alakjában a számjegyek számát.

```
from decimal import Decimal, getcontext

def szamSz(x):
    x = Decimal(x)
    count = x.log10()
    return int(count) + 1

def binSzamSz(x):
    getcontext().prec = 100
    x, k = Decimal(x), Decimal(2)
    count = x.log10()/k.log10()
    return int(count) + 1
```

$\sqrt{2}$ értéke

Ha több tizedes jegyet szeretnénk, akkor a `Decimal` típussal és a `getcontext` metódussal kell dolgozzunk.

```
from decimal import Decimal, getcontext
def my_sqrt2(p):
    getcontext().prec = p
    temp = Decimal(0)
    for x in range(0, 500):
        temp = 1 / (2 + temp)
    return 1 + temp

>>> my_sqrt2(30)
    Decimal('1.41421356237309504880168872421')

>>> my_sqrt2(50)
    Decimal('1.41421356237309504880168872420969807856967187537769')
```