Diszkrét matematika
4. előadás

MÁRTON Gyöngyvér
mgyongyi@ms.sapientia.ro

Sapientia Egyetem,
Matematika-Informatika Tanszék
Marosvásárhely, Románia

2022, őszi félév

Miről volt szó az elmúlt előadáson?

- 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,

Miről lesz szó?

- Python: `Fraction` típus, a `random`, a `decimal` a `numpy` és a `matplotlib` modulok,
- racionális számok, irracionális számok, lánctörtek,
- "híresebb" irracionális számok: $\sqrt{2}, \pi, e, \phi$,
- algoritmusok:
  - legnagyobb közös osztó (Eukleidész) algoritmusa - iteratív, rekurzív változatok,
  - racionális számok irreducibilis alakja,
  - racionális számok sorozatba rendezése - két módszer,
  - a racionális számok lánctört jegyei,
  - irracionális számok tizedes jegyei

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.

1. 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: $d = (a, b) = (b, a - b \cdot q)$.

Eukleidész algoritmusa

Eukleidész algoritmusa a maradékos osztás és 3. előadáson vett 1.tétel alapján adható meg: Legyenek a, b pozitív egész számok, ahol $a \geq b$ és legyen $r_0 = a$, $r_1 = b$:

$$
\begin{array}{rcll}
r_{\mathbf{0}} & = & r_{\mathbf{1}} \cdot q_{\mathbf{1}} + r_{\mathbf{2}} & 0 \leq r_{\mathbf{2}} < r_{\mathbf{1}}, \\
r_{\mathbf{1}} & = & r_{\mathbf{2}} \cdot q_{\mathbf{2}} + r_{\mathbf{3}} & 0 \leq r_{\mathbf{3}} < r_{\mathbf{2}}, \\
 & . & & \\
 & . & & \\
 & . & & \\
r_{n-\mathbf{2}} & = & r_n - 1 \cdot q_{n-\mathbf{1}} + r_n & 0 \leq r_n < r_{n-\mathbf{1}}, \\
r_{n-\mathbf{1}} & = & r_n \cdot q_n. &
\end{array}
$$

$$
\begin{array}{rcllcl}
(32, 76) & = & (76, 32 - 0 \cdot 76) & = & (76, 32) & \\
(76, 32) & = & (32, 76 - 2 \cdot 32) & = & (32, 12) & \\
(32, 12) & = & (12, 32 - 2 \cdot 12) & = & (12, 8) & \\
(12, 8) & = & (8, 12 - 1 \cdot 8) & = & (8, 4) & \\
(8, 4) & = & (4, 8 - 2 \cdot 4) & = & (4, 0) = 4 &
\end{array}
$$

A legnagyobb közös osztó egyenlő lesz a fenti számítási sorozat során meghatározott utolsó nem nullás maradékkal:

$$(a, b) = (r_{\mathbf{0}}, r_{\mathbf{1}}) = (r_{\mathbf{1}}, r_{\mathbf{2}}) = \ldots = (r_{n-\mathbf{2}}, r_{n-\mathbf{1}}) = (r_{n-\mathbf{1}}, r_n) = (r_n, r_{\mathbf{0}}) = r_n.$$

Eukleidész algoritmusa

1. feladat

Írjunk egy Python függvényt, amely Eukleidész módszerével meghatározza két egész szám legnagyobb közös osztóját.

```
def lnkoF_(a, b):
    while b != 0:
        r = a % b
        a = b
        b = r
    return a

>>> lnkoF(-32, -76)
    4
>>> lnkoF(32.7, 76)
    'hibás bemenet'
```

```
def lnkoF(a, b):
    if not isinstance(a, int) or not isinstance(b, int):
        return 'hibás bemenet'
    a = abs(a)
    b = abs(b)
    while True:
        r = a % b
        if r == 0: break
        a = b
        b = r
    return b
```

Eukleidész algoritmusa - rekurzív változat

2. feladat

Írjunk egy Python függvényt, amely Eukleidész módszerével, rekurzívan határozza meg két egész szám legnagyobb közös osztóját.

```
def lnkoR(a, b):
    temp = a % b
    if temp == 0: return b
    return lnkoR(b, temp)
```

3. feladat

Írjunk egy Python függvényt, amely meghatározza egy adott racionális szám irreducibilis alakját.

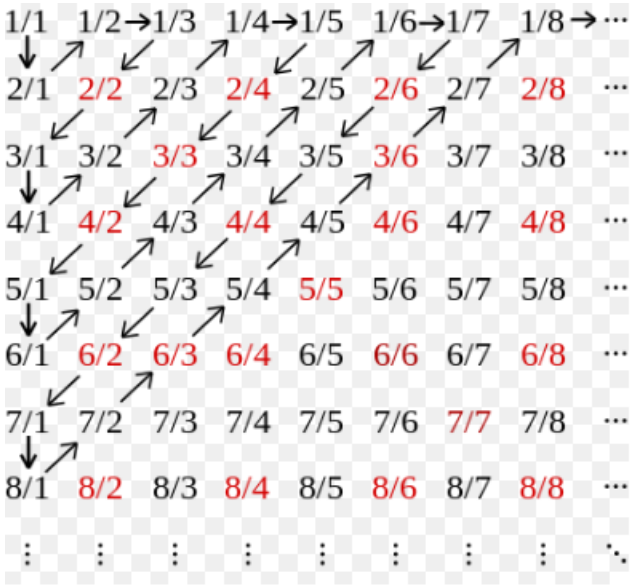
Az $\frac{a}{b}$ racionális számot az `(a, b)` értékpárral fogjuk jelölni, azaz `tuple` típusú adatként kezeljük. Egy tört irreducibilis alakját úgy határozzuk meg, hogy kiszámoljuk a számláló és nevező legnagyobb közös osztóját, majd végig osztunk ezzel az értékkel.

```
def iAlak(a, b):
    temp = lnkoF(a, b)
    return (a // temp, b // temp)
```

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

4. 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

5. 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

6. 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

7. 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

8. 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

9. 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
```

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

>>> 1/3
    0.3333333333333333

>>> Decimal(1/3)
    Decimal('0.333333333333333314829616256247390992939472198486328125')

>>> Decimal('0.3')
    Decimal('0.3')

>>> Decimal(0.1) + Decimal(0.1) + Decimal(0.1) - Decimal(0.3)
    Decimal('2.77555756156515654042363166 8E-17')

>>> Decimal('0.1') + Decimal('0.1') + Decimal('0.1') - Decimal('0.3')
    Decimal('0.0')
```

A decimal modul

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

```
>>> from decimal import getcontext

>>> x = Decimal(1)
>>> y = Decimal(3)

>>> getcontext().prec = 10
>>> x / y
    Decimal('0.3333333333')

>>> getcontext().prec = 25
>>> x / y
    Decimal('0.3333333333333333333333333')
```

Az `int`, a `float`, a `string` típusokból egyaránt létre lehet hozni Decimal típusú értéket.

$\sqrt{2}$ 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')
```

A π szám

- a kör kerületének és átmérőjének hányadosa az eukleidészi geometriában
- más definíciók is léteznek, melyek kihagyják a kört
- irracionális és transzcendens szám (nincs olyan egész együtthatós polinom amelynek gyöke lenne)

A π értékének meghatározására több lánctört-képlet is ismert:

$$\pi = \cfrac{4}{1 + \cfrac{1^2}{3 + \cfrac{2^2}{5 + \cfrac{3^2}{7 + \ldots}}}}$$

$$\pi = 3 + \cfrac{1}{6 + \cfrac{3^2}{6 + \cfrac{5^2}{6 + \cfrac{7^2}{6 + \ldots}}}}$$

A π szám

10. feladat

Határozzuk meg π értékét lánctörtek segítségével.

```
def my_pi(p):
    getcontext().prec = p
    temp = Decimal(0)
    for x in range (1000, 0, -1):
        a = x * x
        b = 2 * x + 1
        temp = a / (b + temp)
    return 4 / (1 + temp)

>>> my_pi(100)
    Decimal('3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706 7')
```

Az e szám

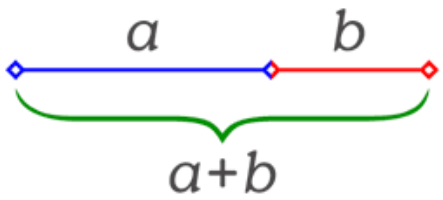
- irracionális és transzcendens szám
- többféleképpen lehet értelmezni:

$$e = \lim_{n \to \infty} \left(1 + \frac{1}{n}\right)^n.$$

$$e = 2 + \cfrac{1}{1 + \cfrac{1}{2 + \cfrac{2}{3 + \cfrac{3}{4 + \ldots}}}}$$

A φ szám, aranymetszés, aranyarány (Golden Ratio)

- két mennyiség, $a, b, a > b$ az aranymetszés szerint aránylik egymáshoz, ha fennáll:



$$\varphi \stackrel{def}{=} \frac{a}{b} = \frac{a+b}{a}$$

- a φ meghatározása érdekében felírhatjuk: $\frac{a+b}{a} = 1 + \frac{1}{\frac{a}{b}}$, azaz fennáll:

$$\varphi = 1 + \frac{1}{\varphi} \Leftrightarrow \varphi^2 = \varphi + 1$$

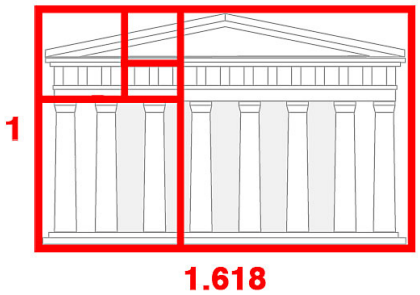
- megoldva a fenti egyenletet kapjuk, hogy

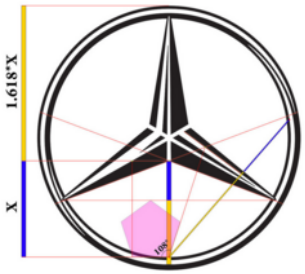
$$\varphi = \frac{1+\sqrt{5}}{2} = 1.61803\ldots \text{ és } \hat{\varphi} = \frac{1-\sqrt{5}}{2} = -0.61803\ldots$$

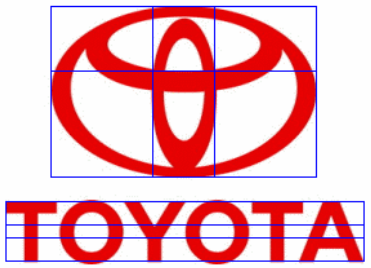
- a φ irracionális szám

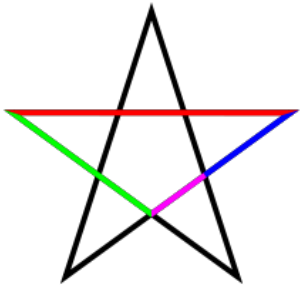
A φ szám, aranymetszés, aranyarány (Golden Ratio)

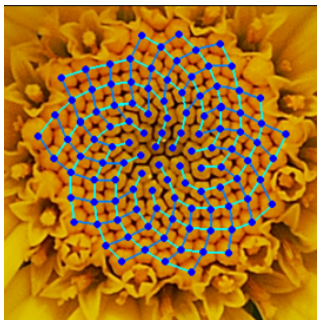
- építészet: Parthenon homlokzatának arányértékei:
- logok: Toyota, Mercedesz, stb.
- Pentagramma (szabályos ötszög): $\frac{piros}{zold} = \frac{zold}{kek} = \frac{kek}{lila} = \varphi$
- természet: napraforgó spiráljai











A φ szám, aranymetszés, aranyarány (Golden Ratio)

Kiindulva a $\varphi = 1 + \frac{1}{\varphi}$ összefüggésből, a φ értékét felírhatjuk a következőképpen:

$$\varphi = 1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{\ddots}}}}$$

A φ szám, aranymetszés, aranyarány (Golden Ratio)

- két egymás utáni Fibonacci szám arányaként is felírhatjuk a φ értékét
- egy Fibonacci szám a Fibonacci számsorozat egy eleme
- a Fibonacci számsorozat: $0, 1, 1, 2, 3, 5, 8, 13...$, kiindulva a 0, 1 kezdőértékekből, a következő elemet az előző két elem összegéből kapjuk

Az `n` és `m` közötti, két egymásutáni Fibonacci számból képzett arányok meghatározása:

```
def fib_phi(n, m):
    f1 = 0
    f2 = 1
    for i in range(2, m):
        f = f1 + f2
        if i >= n:
            print("{0:4d} {1:5d} {2:5d} {3:10,.6f}".format(i, f, f2, f/f2))
        f1 = f2
        f2 = f

>>> fib_phi(10, 14)
  10    55    34   1.617647
  11    89    55   1.618182
  12   144    89   1.617978
  13   233   144   1.618056
```