

Diszkrét matematika

2. előadás

MÁRTON Gyöngyvér
<https://ms.sapientia.ro/~mgyongyi/>
mgyongyi@ms.sapientia.ro

Sapientia EMTE,
Matematika-Informatika Tanszék
Marosvásárhely, Románia

2023, őszi félév



Miről volt szó az elmúlt előadáson?

- követelmények, osztályozás, könyvészet
- áttekintő
- Python:
 - telepítés, használat, a programírás lépései, változók, típusok,
 - típusok közötti átalakítások (`int()`, `float()`, `str()`, `list()`),
 - operátorok, vezérlő szerkezetek (`if`, `for`, `while`)
- a faktoriális függvény - iteratív, rekurzív változatok,

Miről lesz szó?

- 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,
 - nullások száma a faktoriális végén,

Python, az `in` operátor

Az `in` operátor segítségével eldönthető egy adott értékről, hogy hozzátartozik egy adathalmazhoz vagy sem:

```
>>> 4 in range(10)
```

```
True
```

```
>>> "A" in "sapientia"
```

```
False
```

Ha azt szeretnénk megtudni, hogy nem tartozik hozzá egy adott érték az adathalmazhoz, akkor a `not in` operátort kell használni:

```
>>> -1 not in range(10)
```

```
True
```

```
>>> "int" not in ["float", "bool", "int", "str"]
```

```
False
```

Python logikai kifejezések

Egy logikai kifejezés logikai értéke False, ha:

- egyenlő a False vagy a None konstanssal,
- egyenlő üres list, tuple, vagy str-el,
- a kifejezés numerikus és értéke egyenlő 0-val,
- minden más esetben True a kifejezés logikai értéke.

Logikai kifejezések **összehasonlító operátorok** segítségével adhatók meg:
`==`, `!=`, `>`, `>=`, `<`, `<=`, logikai értékük pedig a következő:

`a == b`, ha a **egyenlő** b-vel, akkor True

`a != b`, ha a **nem egyenlő** b-vel, akkor True

`a > b`, ha a **nagyobb**, mint b, akkor True

`a >= b`, ha a **nagyobb vagy egyenlő**, mint b, akkor True

`a < b`, ha a **kisebb**, mint b, akkor True

`a <= b`, ha a **kisebb vagy egyenlő**, mint b, akkor True

Python függvények

- a függvények a bemeneti értéken/értékalmazon végeznek műveleteket, adott esetben meghatározva egy kimeneti értéket/értékalmazt.
- Python függvénydefiniálás:

```
def <fvnév>( <paramlista> ) :  
    <fügtörzs>
```

- a fügvtörzs-ben **return** utasítások adhatók meg, amelyek lehetővé teszik, hogy a függvényben kilépési pontokat adjunk meg, illetve a **return** alkalmazásával határozható meg a függvény kimeneti értéke,
- a **return** alkalmazása maga után vonja, hogy az utána következő műveletsorok ne kerüljenek elvégzésre,
- a fügvtörzs-ben megadott **return** műveleteken keresztül **különböző típusú** kimeneti értékeket is meghatározhatuk.

Python függvények

1. feladat

Írjunk egy Python függvényt, amely megvizsgálja, hogy a függvény bemenete kisbetű-e vagy sem.

```
def tesztBetu1(x):  
    if 'a' <= x and x <= 'z':  
        print('kisbetu')  
    else: print('nem kisbetu')
```

```
>>> tesztBetu1('a')  
kisbetu
```

```
>>> tesztBetu2('/')  
False
```

```
def tesztBetu2(x):  
    if 'a' <= x <= 'z':  
        return True  
    return False
```

```
>>> 'a'.islower()  
True
```

```
>>> 'ab'.islower()
```

```
True
```

```
>>> 'aB'.islower()
```

```
False
```

Python függvények

2. feladat

Határozzuk meg hány nullás számjegy van $n!$ végén, anélkül, hogy meghatároznánk $n!$ értékét. Pl. hány nullás van $1000!$ végén.

- Megszámoljuk hogy az hány 5-el, 5^2 -el, 5^3 -al, stb. osztható szám van n -ig.
- Ha $n = 91$, akkor összesen $18 + 3 = 21$ nullás számjegy van $91!$ végén, mert
 - 5-el osztható számok száma: $91 // 5 = 18$
 - 5^2 -el osztható számok száma: $18 // 5 = 3$
 - 5^3 -al osztható számok száma: $3 // 5 = 0$

```
def factorialNullasok(n):  
    res = 0  
    while n > 0:  
        temp = n // 5  
        res += temp  
        n = temp  
    return res
```

```
>>> factorialNullasok(1000)  
249
```

```
def factorialNullasok_(n):  
    res = 0  
    while n > 0:  
        res += n // 5  
        n = n // 5  
    return res
```

```
>>> factorialNullasok_(1000)  
249
```


Python függvények

3. feladat

Teszteljük, hogy egy szám négyzetszám-e vagy sem.

```
def negyzetTeszt(x):  
    y = int(x ** 0.5)  
    if float(y * y) == x: return True  
    return False
```

4. feladat

Számoljuk meg, és írjuk ki a négyzetszámokat.

```
def negyzetTeszt2(x):  
    y = x ** 0.5  
    if y.is_integer(): return True
```

Python függvények

5. feladat

Írjunk egy Python függvényt, amely meghatározza egy pozitív természetes szám osztóinak számát.

```
def osztokSzama(n):  
    if n <= 0:  
        print("helytelen bemenet")  
        return  
    db = 0  
    i = 2  
    while i <= n // 2:  
        if n % i == 0: db += 1  
        i += 1  
    return db
```

Python függvények

A következő függvényben az osztók számának meghatározása mellett kiíratásra kerülnek az osztók is.

```
def osztok(n):
    if n <= 0:
        print("helytelen bemenet")
        return
    db = 0
    i = 2
    while i <= n // 2:
        if n % i == 0:
            db += 1
            print(i, end = ' ')
        i += 1
    print()
    return db
```

```
>>> osztok(60)
2 3 4 5 6 10 12 15 20 30
10
```

A tuple típus

- az alaptípusok (`int`, `float`, `bool`, `str`) mellett a `tuple` és `list` adatszerkezeteket fogjuk használni
- a `tuple` (rendezett ennes) típus:
 - elemek/adatok szekvenciája, amelyek értékét **nem lehet megváltoztatni/immutable**,
 - jelölésükre kerek zárójelet használunk, ami helyenként elhagyható: létrehozásakor az elemek közé elég ha vesszőt teszünk, ha azonban függvényparaméterként használjuk akkor kötelező a kerek zárójel,
 - kiíratáskor a Python kerek zárójelbe teszi a tuple típusú adatokat,
 - ha azt szeretnénk, hogy egy függvény egynél több értéket térítsen vissza, akkor nagyon előnyös a használatuk,
 - jelölhetjük az egész tuple-t, vagy hivatkozhatunk csak valamely elemére:

Példák:

```
>>> T = 2022, "samsung", 1500
>>> type(T)
<class 'tuple'>
>>> print(T)
(2022, 'samsung', 1500)
>>> T[1]
'samsung'
```

A tuple típus

Példák:

```
>>> T[1] = "huawei" #a tuple elemének értéke nem változtatható meg
```

```
...
```

```
TypeError: 'tuple' object does not support item assignment
```

```
>>> T = T + ("huawei",)
      (2022, 'samsung', 1500, 'huawei')
```

```
>>> len(T) #a tuple elemeinek száma
4
```

```
>>> t0, t1, t2, t3 = T
```

```
>>> t1
'samsung'
```

```
>>> for i in range(len(T)):
      print(T[i], end = " ")
```

```
>>> for elem in T:
      print(elem, end = " ")
```

A tuple típus

6. feladat

Írjunk egy Python függvényt, amely a két bemeneti paraméterén alkalmazza az alap aritmetikai műveleteket. A függvény visszatérési értéke egy 5 elemű tuple legyen.

```
def muveletek(x, y):  
    return x + y, x - y, x * y, x // y, x % y, x / y  
  
>>> muveletek(25, 3)  
(28, 22, 75, 8, 1, 8.333333333333334)  
  
>>> M = muveletek(25, 3)  
  
>>> for elem in M:  
    if isinstance(elem, float): print(elem)  
8.333333333333334  
  
>>> for elem in T:  
    if isinstance(elem, str): print(elem, end = " ")  
samsung huawei
```

A tuple típus

Az műveletek függvény által meghatározott értékek lekérhetőek egy másik függvényben is, a tuple elemeket ekkor nem kell zárójelbe tenni:

```
def muveletekMain():
    print ('x:', end = " ")
    x = int (input())
    print ('y:', end = " ")
    y = int (input())
    e1, e2, e3, e4, e5, e6 = muveletek(x, y)

    print ("összeg: ", e1)
    print ("különbség: ", e2)
    print ("szorzat: ", e3)
    print ("osztási egészrész: ", e4)
    print ("osztási maradék: ", e5)
    print ("valós osztás: ", e6)

>>> muveletekMain()
x: 91
...
```

A list típus

A lista (**list**) típus

- elemek/adatok szekvenciája, amelyek értékét **meg lehet változtatni/mutable**,
- jelölésére szögletes zárójelet használunk,
- hasonló a tuple adatszerkezethez, a programozó dönti el, melyiket előnyösebb használni,
- például azonos típusú adatok esetében list típust, különböző típusú adatok esetében tuple típust szoktak használni

Példák:

```
>>> L = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
>>> type(L)
<class 'list'>
```

```
>>> L[2] = -2    #a lista egy vagy több elemének értéke megváltoztatható
```

```
>>> print (L)
[0, 1, -2, 3, 4, 5, 6, 7, 8, 9]
```

```
>>> len(L)      #a lista elemeinek száma
10
```

```
>>> print ('a lista fordított sorrendben: ', L[::-1])
a lista fordított sorrendben: [9, 8, 7, 6, 5, 4, 3, -2, 1, 0]
```


A list típus

```
>>> L = L + [10]      #hozzafuzes, a vegere
>>> print(L)
[0, 1, -2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> L = [-10] + L    #hozzafuzes, az elejere
>>> print(L)
[-10, 0, 1, -2, 3, 4, 5, 6, 7, 8, 9, 10]

>>> L = ['egeszSzamok'] + L    #más típusú értékkel is bővíthető
>>> print(L)
['egeszSzamok', -10, 0, 1, -2, 3, 4, 5, 6, 7, 8, 9, 10]

>>> L[:3]           # a lista elemei, a harmadik elemig
['egeszSzamok', -10, 0,]
>>> L[3:]          # a lista elemei a harmadik elemtől kezdődően
[1, -2, 3, 4, 5, 6, 7, 8, 9, 10]

>>> L1 = 5 * ['helo']
>>> print(L1)
['helo', 'helo', 'helo', 'helo', 'helo']
```

A list típus

Teszteljük, hogy megadott listaelemek között, melyek a négyzetszámok:

```
def tesztNegyzetek1():
    for i in [12, 34, 121, 65, 256, 700]:
        if negyzetTeszt(i): print(i, end = ' ')
>>> tesztNegyzetek1()
121 256
```

A függvénynek a listát paraméterként adjuk meg:

```
def tesztNegyzetek2( L ):
    for i in L:
        if negyzetTeszt(i): print(i, end = ' ')
>>> tesztNegyzetek2([12, 625, 121, 65, 256, 700])
625 121 256
```

A négyzetszámokból egy új listát építünk, ez lesz a függvény kimeneti értéke:

```
def valogatNegyzetek(L):
    ujL = []
    for i in L:
        if negyzetTeszt(i): ujL += [i]
    return ujL
>>> valogatNegyzetek([12, 625, 121, 65, 256, 700])
[625, 121, 256]
```

Műveletek karakterláncokkal, a +, *, : operátorok

```
>>> str1, str2, str3 = "Diszkrét", " Matematika", " I. félév"  
>>> strT = str1 + str2 + str3  
>>> print (strT)  
    'Diszkrét Matematika I. félév'
```

```
>>> strT = 3 * 'Helo '  
>>> strT  
    'Helo Helo Helo '
```

Szeletelések str típusú adatokon:

```
>>> wStr = '<a href="http://www.ms.sapientia.ro">Sapientia EMTE</a>'
```

```
>>> wStr[8:]  
    '"http://www.ms.sapientia.ro">Sapientia EMTE</a>'
```

```
>>> wStr[9:35]  
    'http://www.ms.sapientia.ro'
```

```
>>> n = len(wStr)  
>>> wStr[:n-4]  
    '<a href="http://www.ms.sapientia.ro">Sapientia EMTE'
```

Python adatbevitel

input - adatbeviteli függvény, a beviteli értéket van amikor át kell alakítani

7. feladat

Írjunk egy Python-függvényt, amely meghatározza $a^i \pmod n$ értékét minden $i = 1, 2, \dots, n-1$ értékre. Az a és n értékeit a billentyűzetről olvassuk be.

Megjegyzések: A *mod* művelet az osztási maradék meghatározását jelenti, tulajdonképpen a % operátor matematikai jelöléséről van szó. Az n számot a matematikában modulusnak hívják. A `pow` függvény meghívható három paraméterrel, ahol a harmadik paraméter a modulus értéket jelöli. Ekkor **moduláris hatványozásnak** hívjuk a műveletet.

```
def osztasiM_():
    n = int(input('modulus: '))
    a = int(input('a: '))
    for i in range(1, n):
        print(pow(a, i, n))

>>> osztasiM_()
modulus: 11
a: 2
...

>>> pow(2, 9)
512
>>> (2 ** 9) % 11
6
>>> pow(2, 9, 11)
6
```

Python adatbevitel

A kiírás formázásához a print-ben használhatjuk % operátort, nem mint aritmetikai műveletet, hanem mint formázó szimbólumot. A Python nem fogja összekavarni a kétféle szerepkört. A % operátor két paraméteres, baloldali paramétere egy str típusú érték, amelyben meghatározzuk, hogy a jobboldali paraméter milyen formátumú legyen. A jobboldali paraméter lehet egy érték, vagy egy tuple.

```
def osztasiM():
    n = int(input('modulus: '))
    alap = int(input('alap: '))
    print('%4s%3s%12s%3s%3i' % ('alap', 'i', '(alap^i)', 'mod', n))
    for i in range(1, n):
        print('%3i%4i%10i' % (alap, i, pow(alap, i, n)))
```

```
>>> osztasiM()
modulus: 11
alap: 2
alap i      (alap^i)mod 11
 2   1         2
 2   2         4
  ...
```

```
>>> '%7i' % 78
'      78'
>>> import math
>>> '%7.2f' % math.pi
'   3.14'
>>> x, y = 340, 120
>>> '%12s%12s' % (x, y)
'          340          120'
```

Python adatbevitel

8. feladat

Írjunk egy Python függvényt, amely az Egyetem Napja alkalmából meghívót készít különböző diákok részére, ahol a diákok neveit a billentyűzetről olvassuk be.

```
def meghivo(n):
    diakok = []
    for i in range(n):
        print('nev: ', end= '')
        diakok += [input()]
    for d in diakok:
        s1 = 'Kedves ' + d + '! \n\n'
        s2 = 'Tisztelettel meghívjuk az Egyetem Napja \n'
        s3 = 'alkalmából tartott előadássorozatra! \n\n'
        print(s1 + s2 + s3)
```

```
>>> meghivo(3)
```

```
...
```

Az `input` által beolvasott értéket nem szüksége átalakítani, mert az algoritmus `str` típusú értékeket kell feldolgozzon.