

Diszkrét matematika

1. előadás

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









Sapientia EMTE,
Matematika-Informatika Tanszék
Marosvásárhely, Románia

2023, őszi félév



Követelmények, osztályozás

- egy egyetemi félév 14 hétből áll, Diszkrét matematikából egy héten 2 óra előadás, 2 óra gyakorlat van,
- a vizsgajegy két felmérő átlagából áll, ahol a második jegy kötelező módon nagyobb kell legyen, mint 5,
- a felmérőkre a labortermekben laborórákon kerül sor, és egy feladatsorból fognak állni, amelyeket számítógépen kell leprogramozni Python programozási nyelvben,
- a felmérők pontos időpontja 2 héttel a felmérők előtt lesz közölve,
- a laborórákon való jelenlét kötelező, egy laboróráról való hiányzást bármelyik másik csoporttal be lehet pótolni, de csak a hiányzás utáni első három héten,
- a felmérőkön abban az esetben lehet részt venni, ha a diáknak nincs laborórákon hiányzása, ellenkező esetben csak a pótvizsgákon lehet résztvenni,
- a vizsgajegyet pótvizsgákon is meg lehet szerezni, erre a két pótszesszióban van lehetőség, amelyeket az egyetem szervez,
- a pótvizsgák is egy-egy feladatsorból fognak állni, amelyeket hasonlóan számítógépen kell leprogramozni Python programozási nyelvben,
- az első pótvizsgaidőszak a félévi vakáció után van,
- a második pótvizsgaidőszak az egyetemi év végén, ősszel van, amelyért az egyetem vizsgáztatási díjat számít fel.

-  Freud R., Gyarmati E., Számelmélet, Nemzeti Tankönyvkiadó, Budapest, 2000.
-  Cormen T.H., Leiserson C.E., Rivest R.L., Algoritmusok, Műszaki Könyvkiadó, Budapest, 2001.
-  Lovász L., Pelikán J., Vesztergombi K., Diszkrét matematika, Typotex, Budapest, 2006.
-  Rónyai L. Ivanyos G., Szabó R., Algoritmusok, Typotex, Budapest, 2004
-  Rosen K.H., Discrete Mathematics and its Applications, McGrawHill, New-York, 2012.
-  Magnus Lie Hetland M.L., Beginning Python: From Novice to Professional, 2nd edition, Apress, 2008.
-  <https://realpython.com/>
-  <http://nyelvek.inf.elte.hu/leirasok/Python/>

Áttekintő - diszkrét matematika

- számok, számtartományok
 - gyorsítványozás algoritmusai,
 - algoritmusok hatékonysága,
 - rekurzió,
 - kiválogatás, minimum érték meghatározása,
 - legnagyobb közös osztó, legkisebb közös többszörös,
 - a Farey sorozat elemei,
 - lánc törtek,
 - irracionális számok számjegyei,
 - a Fibonacci számsorozat,
 - polinom helyettesítési értéke,
 - fraktálok

Áttekintő - diszkrét matematika

- számrendszerek és kódolási technikák
 - természetes szám alapú számrendszerek,
 - számrendszerek közötti kapcsolat,
 - vegyes alapú számrendszerek,
 - bitműveletek,
 - az ASCII kódolás
 - az Unicode szabvány és az utf-8 kódolás
 - a base64 kódolás

Áttekintő - diszkrét matematika

- számelméleti alapfogalmak
 - prímszámok, prímtesztelő, prímfaktorizációs algoritmusok
 - kongruenciák, maradékosztályok, maradékrendszerek,
 - hatványok és generátor elemek,
 - az eukleidészi algoritmus és változatai
 - lineáris kongruenciák,
 - kongruencia rendszerek, a Kínai maradéktétel,
 - alkalmazások: az RSA klasszikus algoritmus, a Diffie-Hellmann kulcscsere,
 - diofantoszi egyenletek,

- **árvéletlen-szám generátorok**
 - lineáris kongruencián alapuló generátor,
 - a közép-négyzet módszer,
 - a Python árvéletlen számgenerátora
- **kombinatorika**
 - lexicografikus sorrend, permutációk, kombinációk, variációk

Bevezető

- diszkrét matematika → algoritmusok,
- minden algoritmust, programot Python programozási nyelvben írunk,
- ingyenesen letölthető verzió: **Python 3.10.7**,
<https://www.python.org/downloads/>
- a Python 1991-ben jelent meg, Guido van Rossum kezdte el fejleszteni,
- Python, tulajdonságok:
 - magasszintű adatszerkezetek: listák, ennesek (tuple), stringek, halmazok, stb.,
 - dinamikus típusrendszer: a típusinformációk kezelése futási időben történik
 - automatikus memóriakezelés,
 - objektum orientáltság,
 - rövid programok írása gyors és egyszerű
 - interpreter: a saját utasításait bemenő adatként kezeli, ezeket átalakítja a futtató gép utasításává, majd rögtön futtatja, ellentétben a fordító (kompilátor) típusú programozási nyelvekkel
 - a Python tulajdonképpen nem interpreter: byte kódot fordít és futtat

Python, elemi kódsorok

- a Python instalálása, majd indítása után megjelenik egy ablak egy kétsoros standard szöveggel (Python verzió szám, stb.) és a prompt: `>>>`
- a prompt után kifejezések, utasítások írhatóak, amelyeket a Python interpretere rögtön kiértékel:

Példák:

```
>>> print("Hello világ!")  
Hello világ!
```

```
>>> 251 + 965  
1216
```

```
>>> 2 ** 100  
1267650600228229401496703205376  
#A Python tetszolegesen nagy, egész számokkal is tud muveleteket  
#vegezni
```

```
>>> import math  
>>> math.sqrt(10)  
3.1622776601683795
```

Python lekérdezések

Példák:

```
>>> x = 251
>>> y = 965
>>> print(x + y)
1216
```

```
>>> x, y = 10, 4
>>> print(x / y, x // y , x % y)
2.5 2 2
```

```
>>> x, y = 17, 5
>>> print("valos osztas: ", x/y)
valos osztas: 3.4
>>> print("osztasi egeszresz: ", x // y)
osztasi egeszresz: 3
>>> print("osztasi maradek: ", x % y)
osztasi maradek: 2
```

Python változók/referenciák

- A programozási nyelvek egyik alapfogalma a **változó**:
 - a változók értékek tárolását teszik lehetővé, a Python azonban nem rendelkezik a hagyományos értelemben vett változókkal, helyette objektum-referenciákon keresztül végzi az adattárolást,
 - az objektum-referencia jelentése programozási nyelvektől függően változik, kezdő programozók számára a változó és referencia közötti különbségnek nincs nagy jelentősége, ezért az egyszerűség kedvéért a továbbiakban a változó megnevezést fogjuk használni,
 - a változó **értéke**: értékadás, értékmodosító utasításokkal határozzuk meg,
 - a változó **típusa**: a típus alapján dől el, hogy a milyen fajta értékeket kezel/tárol,
 - a változókkal végzett műveletsorok képezik a programírás alapjait,
 - Pythonban nincs explicit változódeklarálás, a változódeklarálás **automatikus**, pl. értékadás során.
- más alapfogalmak: **operátor**, **függvény**:
 - Az értékadó operátor az egyenlőség (=).
 - A **print** függvény kiértékeli a zárójelben megadott kifejezést, és az eredményt kiírja a standard kimenetre.

Python alaptípusok/osztályok

- Kezdő programozók számára az **osztályok és típusok** egy fogalmat jelölnek, jelzik hogy milyen fajta adattal dolgozunk. Egy adott érték típusa pont olyan fontos informatikai fogalom, mint az érték.
- alaptípusok: int, float, str, bool

```
>>> type(103373189)
<class 'int'>
```

```
>>> type(106.909)
<class 'float'>
```

```
>>> type(True)
<class 'bool'>
```

```
>>> type('helo vilag')
<class 'str'>
```

```
>>> type("helo vilag")
<class 'str'>
```

Karakterláncok (str) jelölésére egyformán használhatjuk a ' , illetve " szimbólumokat

Típusok közötti átalakítások

Egész számmá alakítunk:

```
>>> int(23.11)
```

```
23
```

```
>>> int(23.0)
```

```
23
```

```
>>> int(-23.900)
```

```
-23
```

```
>>> int(10/3)
```

```
3
```

```
>>> int('23 szo')
```

```
Traceback (most recent call last):
```

```
  File "<pyshell#25>", line 1, in <module>
```

```
    int('23 szo')
```

```
ValueError: invalid literal for int() with base 10: '23 szo'
```

Típusok közötti átalakítások

Valós számmá alakítunk:

```
>>> float(15)
15.0
>>> float('23.67')
23.67
```

Karakterlánc típusúvá alakítunk:

```
>>> str(12)
'12'
>>> str(12.67)
'12.67'
>>> str(True)
'True'
```

A programírás lépései

- az interpreterbe beírt kifejezések, utasítások elvesznek, ha kilépünk a Pythonból,
- IDLE (integrated development environment): a Python standard fejlesztői környezete, programszerkesztésre, mentésre, állomány megnyitásra, futtatásra ad lehetőséget,
- a kódszerkesztőt a File/New File menüpontból lehet elindítani
- a kódszerkesztő menüpontjai:
 - File/New File -új állomány létrehozása;
 - File/Save -állomány mentése,
 - Run/Run Module vagy Ctrl/Fn + F5 -a program futtatása a Python shell-ből
 - stb.
- Python program: scriptnek is mondják

Python vezérlő szerkezetek

- Az `if` elágazásutasítás definiálása:

```
if <kif> : <elágazástörzs>
(elif < kif > : <elágazástörzs>)
[else : <elágazástörzs>]
```

Jelentése: ha igaz a megfelelő `kif` kifejezés, akkor a megfelelő elágazástörzs hajtódik végre. Az `elif`, `else` ágak nem kötelezőek.

- mentsük el `eload1.py` néven azt az állományt, amelybe a következő kódsorokat tesszük:

```
x = 10
y = 13
if x > y: print (x,' nagyobb, mint ',y)
else: print (y, ' nagyobb vagy egyenlo, mint ', x)
```

- az eredmény:

```
13 nagyobb vagy egyenlo, mint 10
```


Python programok futtatása

Windows parancssorból(Command Prompt, PowerShell) való futtatás:

- be kell állítani a PATH környezetváltozót (environment variable), úgy hogy megadjuk a python.exe állomány elérési útvonalát (System/Advanced system settings)

- hogy megtudjuk hol van a python.exe, a Python shellbe írjuk be a következőket:

```
>>> import os
>>> import sys
>>> os.path.dirname(sys.executable)
```

- a beállítás után az op rendszer fel fogja ismerni a python parancsot, ez később is hasznos lesz amikor Python csomagot akarunk telepíteni
- a parancssorban a CD rendszerparancssal kiválasztjuk azt a mappát, ahol az eload1.py állomány van
- futtatás: C:\> python eload1.py
- futtatás: kétszer klikkelve az eload1.py-on

Python, megjegyzések

Megjegyzések, "kommentek" használata:

- egysoros megjegyzés: `#ez egy egysoros megjegyzés`
- többsoros megjegyzés:

```
"""ez egy  
többsoros megjegyzés"""
```

Python, vezérlőszerkezetek

A **for** ciklusutasítás definiálása:

```
for <elem> in <halmaz> :  
    <ciklustörzs>
```

- a **for** a **halmaz** minden **elem** elemére végrehajtja a **ciklustörzs**-ben megadott utasításokat.
- megfelelő tördeléssel (egy tabulátornyival helyet bennebb) kell jelezni, hogy melyek azok az utasítások amelyek a ciklustörzshöz tartoznak

Példák:

```
>>> for i in range(10):  
    print (i) #ugyeljunk a tordelesre!!  
    #ket ENTER-t kell nyomni
```

```
>>> range(0, 10)  
range(0, 10)
```

```
>>> list(range(0, 10))  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- a **range** függvény egy számsorozatot generál, a megadott határértékel/határértékekkel,
- a **list** átalakítja lista típusú értéké a generált számsorozatot

Python, lekérdezések

Írjuk be az eload1.py állományba a következő kódsorokat is majd futassuk az állományt, a korábban beírt kódsorokat kommenteljük ki:

```
for i in range(0, 10):  
    print (i, end = " ")  
print()
```

```
for i in range(10):  
    print (i, end = "")  
print()
```

```
for i in range(10):  
    print (2 ** i, end = " ")
```

Az állomány futtatásakor a megfelelő számsorozatok kerülnek kiíráásra:

0 1 2 3 4 5 6 7 8 9

0123456789

1 2 4 8 16 32 64 128 256 512

Python, vezérlőszerkezetek

- a `while` ciklusutasítás definiálása:

```
while <kifejezes>:  
    <ciklustörzs>
```

- a ciklustörzs addig kerül ismételtlen végrehajtásra, ameddig a kifejezes logikai értéke igaz
- a megfelelő tördeléssel kell most is jelezni, hogy melyek azok az utasítások amelyek a ciklustörzshöz tartoznak
- bármely kifejezes logikai értéke igaznak minősül, ha az nullától különbözik
- ha a kifejezes értéke indulásból hamis, akkor egyszer sem hajtódik végre a ciklustörzs

A következő kódsort is írhatjuk az `eload1.py` állományba. A műveletek elvégzése után meghatározásra kerül az első n természetes szám szorzata, azaz n faktoriális értéke. A matematikai definíció szerint: $n! = n \cdot (n-1) \cdot \dots \cdot 2 \cdot 1$, és $0! = 1$.

```
n, res = 10, 1  
while n >= 1:  
    res = res * n  
    n = n - 1  
print(res)
```

Python függvények

1. feladat

Határozzuk meg $n!$ értékét, azaz az összes n -nél kisebb pozitív szám szorzatát.

Könyvtárfüggvénnyel:

```
>>> from math import factorial
>>> factorial(10)
3628800
```

Iteratív függvénnyel:

for ciklusutasítással:

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

while ciklusutasítással:

```
def factorialWhile(n):
    res = 1
    while n >= 1:
        res = res * n
        n = n - 1
    return res
```

Python függvények

Rekurzív függvényekkel:

```
def factorialRek1(n):  
    if n == 0: return 1  
    return n * factorialRek1(n-1)
```

```
def factorialRek2(n):  
    return factorialAux(n, 1)
```

```
def factorialAux(n, res):  
    if n == 0: return res  
    return factorialAux(n - 1, res * n)
```

- a `factorialRek1` a következő sorrendbe szorozza össze a számokat: $1 \cdot 2, 1 \cdot 2 \cdot 3, 1 \cdot 2 \cdot 3 \cdot 4, \dots$, azaz akkor számol mikor jön vissza a rekurzióból
- a `factorialRek2` feltételezve, hogy $n = 10$, a következő sorrendbe szorozza össze a számokat: $1 \cdot 10, 1 \cdot 10 \cdot 9, 1 \cdot 10 \cdot 9 \cdot 8, \dots$, azaz akkor számol mikor megy be a rekurzióba