# Diszkrét matematika
# 8. előadás

MÁRTON Gyöngyvér
mgyongyi@ms.sapientia.ro

Sapientia Egyetem,
Matematika-Informatika Tanszék
Marosvásárhely, Románia

2024, őszi félév

# Miről volt szó az elmúlt előadáson?

- egyéb számrendszerek:
  - a faktoriális számrendszer
  - a Fibonacci számrendszer
- az `n`-ik Fibonacci szám, exponenciális, lineáris, logaritmikus futásidejű algoritmusok
- bitműveletek

# Miről lesz szó?

- a Hamming súly,
- az `ord, chr` függvények, bitműveletek,
- a `random` modul,
- bináris állományok,
- kódolási technikák: ASCII, UTF-8, Unicode,
- az `index, join` metódusok,
- az `str, bytes` típusok, átalakítások,
- a `to_bytes`, a `from_bytes` metódusok
- titkosítás (az XOR egy alkalmazása),

# Bitműveletek

**1. feladat**

*Hány 1-es bitet tartalmaz egy természetes szám kettes számrendszerbeli alakja?*

Más megfogalmazásban határozzuk meg a Hamming-súlyát egy adott számnak. Nem kell összetéveszteni a Hamming távolsággal!

```
def HammingW1(nr):
    db = 0
    while nr:
        db += (nr & 1)
        nr >>= 1
    return db
```

```
>>> HammingW1(1023)
    10
>>> HammingW1(1024)
    1
```

```
def HammingW2(nr):
    db = 0
    while nr:
        nr &= nr - 1
        db += 1
    return db
```

```
>>> HammingW2(1023)
    10
>>> HammingW2(1024)
    1
```

# Bitműveletek

kiíratjuk a részadatok értékét:

```
def HammingW1(nr):
    db = 0
    while nr:
        db += (nr & 1)
        print(format(nr, 'b'))
        print(format(nr & 1, 'b'))
        print()
        nr >>= 1
    return db
```

```
>>> HammingW1(59)
111011
1

11101
1

1110
0

111
1

11
1

1
1

5
```

# Bitműveletek

kiíratjuk a részadatok értékét:

```
def HammingW2(nr):
    db = 0
    while nr:
        print(format(nr, 'b'))
        print(format(nr-1, 'b'))
        print(format(nr & (nr-1), 'b'))
        print()
        nr &= nr - 1
        db += 1
    return db


>>> int('111110', 2)
    62
>>> int('111101', 2)
    61

>>> int('111100', 2)
    60
>>> int('111011', 2)
    59
```

*nr* alapján a $nr - 1$ meghatározása: a *nr* bináris alakjában, jobbról az utolsó 1-es bitet 0-ra állítjuk és az összes után következő bitet 1-re.

```
>>> HammingW2(63)
    111111
    111110
    111110

    111110
    111101
    111100

    111100
    111011
    111000

    111000
    110111
    110000

    110000
    101111
    100000

    100000
    011111
    000000

    6
```

# Python, az ord, chr függvények

A számítástechnikában használt karakterekhez, egy kódolási eljárás segítségével, számokat rendelnek. A Pythonban az `ord` megadja egy adott karakter kódját, a `chr`, pedig meghatározza a kódértékhez tartozó karaktert.

```
def fugv1(L):
    nL = []
    for elem in L:
        nL += [ord(elem)]
    return nL

def fugv2(L):
    nL = ""
    for elem in L:
        nL += chr(elem)
    return nL

>>> fugv1("matematika informatika tanszek")
    [109, 97, 116, 101, 109, 97, 116, 105, 107, 97, 32, 105, ...]
>>> fugv2([109, 97, 116, 101, 109, 97, 116, 105, 107, 97])
    'matematika'
```

# Python, a random könyvtárcsomag

**2. feladat**

*Generáljunk véletlenszerűen $n$ darab nagybetűt.*

```
def myRand(n, k1 = 65, k2 = 65 + 25):
    L = ''
    for i in range(0, n):
        temp = chr(random.randint(k1, k2))
        L += temp
    return L

>>> myRand(10)
    'AUQXEVJLRQ'
>>> myRand(10, 97, 97+25)
    'khobnyeqin'

>>> myRand(10, 50000, 53000)

>>> random.sample([chr(i) for i in range(65, 65 + 26)], 10)
>>> ''.join(random.sample([chr(i) for i in range(65, 65 + 26)], 26))

>>> random.sample(range(1, 100), 10)
```

# Bináris állományok

**3. feladat**

*Írjunk programot, amely megjeleníti egy állomány bájtjait hexa számrendszerben.*

```
def fileHexa():
    fnev = input('kerek egy allomanynevet:')
    try:
        inf = open(fnev, 'rb')
        bajtL = inf.read()
        inf.close()
    except:
        print('megnyitasi/olvasasi hiba!!')
        return
    out = open('hexImage.txt', 'wt')
    bajtR = ''
    for bajt in bajtL:
        bajtR += format(bajt, 'x') + ' '
    out.write(bajtR)
    out.close()
```

# Kódolási technikák

Többféle kódolási technika létezik, ahol a kódolás a feldolgozandó adat egy átalakítási módját jelenti. Kódolási módszerek, kódok:

- gépi kód: bináris formában tárolja az adatokat és utasításokat,
- Ascii kód, Unicode, UTF-8,
- a leíró nyelvek kódrendszere: HTML, LaTex,
- tömörítést megvalósító kódok: Huffman kód,
- hibajelző/hibajavító kódok: Hamming kód,
- base64 kódolás:
  - az SMTP (Simple Mail Transfer Protocol) átviteli protokollban használták először,
  - adatbázisok esetében ID-k tárolása,
  - weboldalak tárolása,
  - kriptográfia protokollok: kulcsok tárolása.

# Az ASCII kódolás

- 1968-ben írták le a standardot, különböző szimbólumokhoz rendel numerikus értéket
- kezdetben 128 szimbólumot kódolt, később 256 (extended ASCII)

**4. feladat**

*Írjunk Python függvényt, amely kiírja a képernyőre az ASCII szimbólumokat és a hozzájuk tartozó kódértékeket.*

```
def myASCIItable0():
    for i in range(128):
        print(chr(i), i)

def myASCIItable1(n = 0, m = 256):
    for i in range(n, m):
        print(chr(i), i)

>>> myASCIItable1(65, 91)

>>> myASCIItable1(97, 123)
```

# Az Unicode szabvány

- egységes karakterkódolás, karakterosztályozás, szabvány, fejlesztője az Unicode Consortium
- tartalmazza a világ különböző nyelveinek szimbólumait, műszaki szimbólumokat, stb.
- az Unicode karakterek implementálása többféleképpen is lehetséges, az egyik legismertebb az UTF (Unicode Transformation Formats)

```
def myUnicodeTable(n = 51700, m = 51710):
   for i in range(n, m):
      print('%3c%7i' % (chr(i), i), end = ''),
      print('%18s' % format(i, 'b'), end = '')
      print('%8s' % format(i, 'o'), end = '')
      print('%6s' % format(i, 'x'), end = '')
      print()

def myUnicodeTableF(n = 0, m = 55296):
    with open('myUnicode.txt', 'wt', encoding = 'utf-8') as outf:
        for i in range(n, m):
            outf.write('%5s%7i' % (chr(i), i))
            outf.write('%18s' % format(i, 'b'))
            outf.write('%8s' % format(i, 'o'))
            outf.write('%6s' % format(i, 'x'))
            outf.write('\n')
```

- a `myUnicode.txt` állományt notepad++-ban nyissuk meg

# Python, az index, a join metódusok

Az index megadja azt a sorszámot (a legelsőt), amelyen egy adott karakter előfordul a karakterláncban, ha nincs benne, futási hibával leáll:

```
>>> 'http://www.ms.sapientia.ro/'.index('/')
    5
```

A join a megadott karakterláncot elválasztó értékként felhasználva egymásután fűzi a listában vagy tuple-ben megadott elemeket:

```
>>> '#+'.join('hello')
    'h#+e#+l#+l#+o'
>>> ''.join(['A','B','C'])
    'ABC'
>>> ', '.join(('Istvan', 'Zsuzsi', 'Kati'))
    'Istvan, Zsuzsi, Kati'
```

# Python, az index, a join metódusok

```
def betu_kod():
    abece = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    #abece = [chr(i) for i in range(65, 65 + 26)]
    #abece = ''.join([chr(i) for i in range(65, 65 + 26)])
    print(abece)
   c = input('kerek egy nagybetut: ')
   try:
      betu = abece.index(c)
      return betu, ord(c)
   except:
      print('nem nagy betu', end = ' ')

>>> betu_kod()
    kerek egy nagybetut:  B
    (1, 66)
>>> betu_kod()
    kerek egy nagybetut:  &
    nem nagy betu
```

# UTF-8 kódolás

- az Unicode szabvány segítségével a világon használt összes szimbólumnak megfeleltethető egy egész szám, pontosabban ezt kódpontnak hívják, ez azonban nem mindig tárolható el egy bájton
- kérdés: milyen technikával tároljuk ezeket a kódpontokat? azaz mi legyen a bájtsorrend, hány bájtot használjunk?
- a leggyakrabban alkalmazott technika az UTF-8 kódolás

| bájt szám | bit szám | 1.kódpont | 2.kódpont | 1.bájt | 2. bájt | 3. bájt | 4. bájt |
|---|---|---|---|---|---|---|---|
| 1 | 7 | U+0000 | U+007F | 0bbbbbbb | | | |
| 2 | 11 | U+0080 | U+07FF | 110bbbbb | 10bbbbbb | | |
| 3 | 16 | U+0800 | U+FFFF | 1110bbbb | 10bbbbbb | 10bbbbbb | |
| 4 | 21 | U+10000 | U+10FFFF | 11110bbb | 10bbbbbb | 10bbbbbb | 10bbbbbb |

| karakter | kód | bin | 1.bájt | | | 2.bájt | | |
|---|---|---|---|---|---|---|---|---|
| | | | bin | hex | int | bin | hex | int |
| ű | 369 | 101 110001 | 110 00101 | 0xc5 | 197 | 10 110001 | 0xb1 | 177 |

# Bájtsorrend

- big-endian: a legnagyobb helyiértékű bájt a legkisebb címen van tárolva, azaz az MSB (most significant byte) van elől, a hálózatok a csomagok címeiben így használják
- little-endian: a legnagyobb helyiértekű bájt a legnagyobb címen van tárolva, azaz az MSB van hátul, az Intel alapú számítógépek bájt tárolási módja

3DF149D1 tárolása a 32-es címtől kezdve:

big-endian, 1 bájtos tárolás

| memória cím | 32 | 33 | 34 | 35 |
|---|---|---|---|---|
| érték | 3D | F1 | 49 | D1 |

big-endian, 2 bájtos tárolás

| memória cím | 32 | | 33 | |
|---|---|---|---|---|
| érték | F1 | 3D | D1 | 49 |

little-endian, 1 bájtos tárolás

| memória cím | 32 | 33 | 34 | 35 |
|---|---|---|---|---|
| érték | D1 | 49 | F1 | 3D |

little-endian, 2 bájtos tárolás

| memória cím | 32 | | 33 | |
|---|---|---|---|---|
| érték | D1 | 49 | F1 | 3D |

# Python3 `str, bytes`

- a Python3 különböző módon kezeli a bináris adatokat (`bytes`) és a karakterláncokat (`str`)
- Python3-ban minden `str` típusú karakter Unicode-ként van eltárolva, a bináris adatok pedig bájtok (`bytes`) szekvenciáját jelentik
- egy `str` típusú adat átalakítható `bytes` típussá alkalmazva az `encode, bytes` függvényeket; a csak ASCII kódokat tartalmazó stringek a `b` prefix segítségével is átalakíthatóak `bytes` típusú adattá
- egy bytes típusú adat a `decode` függvényt alkalmazva alakítható át `str` típusú adattá

# Python3 str, bytes

Példák:

```
>>> mStr = 'muszaki'
>>> type(mStr)
    <class 'str'>
>>> for m in mStr:
        print(ord(m), end = ' ')
    109 117 115 122 97 107 105
>>> for m in mStr:
        print(m, end = ' ')
    m u s z a k i

>>> mStr = b'muszaki'
    for m in mStr:
        print(m, end = ' ')
    109 117 115 122 97 107 105
```

# Python3, str, bytes

Példák:

```
>>> mStr = 'műszaki'
>>> type(mStr)
    <class 'bytes'>
>>> for m in mStr:
        print(ord(m), end = ' ')
    109 369 115 122 97 107 105

>>> mStr = b'műszaki'
    SyntaxError: bytes can only contain ASCII literal characters.

>>> mStr = 'műszaki'.encode()
>>> for m in mStr:
        print(m, end = ' ')
    109 197 177 115 122 97 107 105
```

# Python3, str, bytes

Példák:

```
>>> mStr = bytes('műszaki', 'ascii')
    ...UnicodeEncodeError: 'ascii' codec can't encode character
    ...

>>> mStr = bytes('műszaki', 'utf-8')
>>> mStr
    b'm\xc5\xb1szaki'
```

A bytes alkalmas arra is, hogy egy megfelelő szerkezetű listát bytes típusú adattá alakítsunk:

```
>>> bytes([109, 197, 177, 115, 122, 97, 107, 105])
    b'm\xc5\xb1szaki'

>>> bytes([109, 300, 177, 115, 122, 97, 107, 105])
    ...ValueError: bytes must be in range(0, 256)
```

# Python3, str, bytes

Példák, bytes típusról alakítunk str típusra:

```
>>> mStr = 'műszaki'.encode()
>>> mStr
    b'm\xc5\xb1szaki'
>>> mStr.decode()
    'műszaki'

>>> mStr = bytes([109,197,177,115,122,97,107,105])
>>> mStr.decode()
    'műszaki'

>>> mStr = bytes([109,197,190,115,122,97,107,105])
>>> mStr.decode()
    'mžszaki'

>>> mStr = bytes([109,197,200,115,122,97,107,105])
>>> mStr.decode()
    ...UnicodeDecodeError: 'utf-8' codec can't decode ...

>>> mStr = bytes([109,117,115,122,97,107,105])
>>> mStr.decode()
    'muszaki'
```

# Python, a to_bytes, a from_bytes metódusok

`to_bytes()`:

- egy egész számot alakít át 256-os számrendszerbe,
- tulajdonképpen bájtokká alakít,
- az eredmény egy bájt szekvencia, hexadecimális alakban, amelynek típusa `bytes`,
- meg kell adni a kimeneti bájtszekvencia hosszát és a bájtsorrendet, amelyre szintén a `big` és `little` megnevezéseket kell használni

```
>>> nr = 63587321362
>>> bitLen = nr.bit_length()
>>> bajtLen = bitLen // 8 + 1
>>> nr.to_bytes(bajtLen, byteorder = 'big')
    b'\x0e\xce\x19\x86\x12'
>>> nr.to_bytes(10, byteorder = 'big')
    b'\x00\x00\x00\x00\x00\x0e\xce\x19\x86\x12'

>>> nr.to_bytes(bajtLen, byteorder = 'little')
    b'\x12\x86\x19\xce\x0e'
```

# Python, az `to_bytes`, a `from_bytes` metódusok

- a `big` sorrend matematikailag azt jelenti, hogy az eredménylista első elme a legnagyobb hatványkitevőn levő tag együtthatója lesz
- a `little` sorrendnél az eredménylista első elme a legkisebb hatványkitevőn levő tag együtthatója lesz

```
>>> for elem in b'\x0e\xce\x19\x86\x12':
        print(int(elem), end = ' ')
    14 206 25 134 18

>>> for elem in b'\x12\x86\x19\xce\x0e':
        print(int(elem), end = ' ')
    18 134 25 206 14
```

$$63587321362 = 14 \cdot 256^4 + 206 \cdot 256^3 + 25 \cdot 256^2 + 134 \cdot 256^1 + 18 \cdot 256^0.$$

# Python, az to_bytes, a from_bytes metódusok

from_bytes():

- a visszaalakítást végzi,
- az int osztály egy metódusa, ezért a meghívás módja eltér a to_bytes meghívás módjától

```
>>> bajtB = b'\x0e\xce\x19\x86\x12'
>>> int.from_bytes(bajtB, byteorder = 'big')
    63587321362
>>> bajtL = b'\x12\x86\x19\xce\x0e'
>>> int.from_bytes(bajtL, byteorder = 'little')
    63587321362
```

Lista típust is megadhatunk bemenetként, de előtte szükséges a listát bytes típusra alakítani:

```
>>> bajtI = bytes([14, 206, 25, 134, 18])
>>> int.from_bytes(bajtI, byteorder = 'big')
    63587321362
```

# Az XOR egy alkalmazása: titkosítás

**5. feladat**

*Titkosítsunk egy karakterláncot(str típust): végezzünk XOR műveletet a karakterlánc karakterjei és egy kulcs-karakterlánc karakterjei között.*

Megj: Valós alkalmazások esetében is használják a XOR-t titkosításra, mert gyors műveletvégzést tesz lehetővé, de mivel a lenti kódsorban a kulcsot körkörösen alkalmazzuk, a titkosítás nem lesz biztonságos!!

```
def crypt(myStr, key):
    crStr = ''
    i = 0
    n = len(key)
    for elem in myStr:
        crStr += cryptE(elem, key[i])
        if i == n - 1: i = -1
        i += 1
    return crStr

def cryptE(kar, crKar):
    return chr(ord(kar) ^ ord(crKar))
```

# Az XOR egy alkalmazása: titkosítás

Ahhoz, hogy visszakapjuk az eredeti karakterláncot, a titkosításhoz és visszafejtéshez is, ugyanazt a kulcsot kell használni, mert:

```
>>> elem = 97
>>> kulcs = 120
>>> elem ^ kulcs
    25
>>> 25 ^ kulcs
    97
```

```
def mainCR_():
   szoveg = input('kerek egy karakterlancot: ')
   kulcs = input('kerek egy kulcsot: ')
   tSzoveg = crypt(szoveg, kulcs)
   print('a titkositott szoveg: ', tSzoveg)

   vSzoveg  = crypt(tSzoveg, kulcs)
   print('a visszafejtett szoveg: ', vSzoveg)
```

```
>>> mainCR_()
    kerek egy karakterlancot: matematika informatika tanszek
    kerek egy kulcsot: kulcs2020
    a titkositott szoveg: ...
```

# Az XOR egy alkalmazása: titkosítás

A `mainCR` függvény elvégzi a titkosítást és a visszafejtést is, ahol a titkosított szöveget hexában írjuk ki, mert amúgy is értelmetlen:

```
def mainCR():
    szoveg = input('kerek egy karakterlancot: ')
    kulcs = input('kerek egy kulcsot: ')
    tSzoveg = crypt(szoveg, kulcs)
    print('a titkositott szoveg: ')
    for c in tSzoveg:
        print(hex(ord(c)), end = ' ')
    print()

    vSzoveg  = crypt(tSzoveg, kulcs)
    print('a visszafejtett szoveg: ', vSzoveg)

>>> mainCR()
    kerek egy karakterlancot: matematika informatika tanszek
    kerek egy kulcsot: kulcs2020
    a titkositott szoveg: 0x6 0x14 0x18 0x6 0x1e 0x53 0x44 ...
    a visszafejtett szoveg:  matematika informatika tanszek
```

# Az XOR egy alkalmazása: titkosítás

**6. feladat**

*Titkosítsunk egy bytes szekvenciát XOR műveletet alkalmazva.*

```
def cryptBytes(bStr, bKey):
    C = bytes([])
    i, n = 0, len(bKey)
    for elem in bStr:
        C += bytes([elem ^ bKey[i]])
        if i == n - 1: i = -1
        i += 1
    return C

def mainCR_Bytes():
    szoveg = input('kerek egy karakterlancot: ')
    kulcs = input('kerek egy kulcsot: ')
    tSzoveg = cryptBytes(szoveg.encode(), kulcs.encode())
    print(tSzoveg.hex())
    vSzoveg = cryptBytes(tSzoveg, kulcs.encode())
    print('a visszafejtett szoveg: ', vSzoveg.decode())
```