Diszkrét matematika
3. előadás

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

Sapientia EMTE,
Matematika-Informatika Tanszék
Marosvásárhely, Románia

2024, őszi félév

Miről volt szó az elmúlt előadáson?

- Python:
  - operátorok,
  - vezérlő szerkezetek (`if`, `for`, `while`)
  - a `range` függvény, az `in` operátor,
  - logikai kifejezések
- Algoritmusok:
  - a faktoriális függvény - iteratív, rekurzív változatok,
  - tesztelés: egy szám négyzetszám-e,
  - osztók száma,
  - nullások száma a faktoriális végén,

Miről lesz szó?

- a `tuple`, a `list`, az `str` adattípusok, szeletelések
- adatbevitel, `print` - a kimenet formázása, kiíratás szövegállományba
- hibakezelés, függvények paraméterátadása
- természetes számok, egész számok
- gyorshatványozás
- algoritmusok futási ideje

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

1. 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
```

Adatbevitel

- `input` - adatbeviteli függvény, visszatérési értékének típusa `str`, ezért a beviteli értéket van amikor át kell alakítani,
- az `muveletek` függvény által meghatározott értékek lekérhetők 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 ("osszeg: ", e1)
    print ("kulonbseg: ", e2)
    print ("szorzat: ", e3)
    print ("osztasi egeszresz: ", e4)
    print ("osztasi maradek: ", e5)
    print ("valos osztas: ", 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, mikor, 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
```

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 = [10, 9.5, 9, 10, 8.75, 10]
>>> L1[::-1] #a lista fordított sorrendben
    [10, 8.75, 10, 9, 9.5, 10]
```

Műveletek karakterláncokkal

```
>>> 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'
```

Műveletek karakterláncokkal

2. feladat

Írjunk egy Python függvényt, amely kiválogatja a bemeneti listaelemek közül azokat amelyek kezdőszelete egyenlő a `https` karakterlánccal.

```
L = ['https://www.ms.sapientia.ro/',
     'https://www.geeksforgeeks.org/',
     'http://www.mag.ro/',
     'https://stackoverflow.com/'
     ]

def valogatElemek(L):
    ujL = []
    for elem in L:
        if teszt(elem): ujL += [elem]
    return ujL

def teszt(x):
    temp = x[:5]
    if temp == 'https': return 1
    return 0
```

Moduláris hatványozás

3. feladat

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

Megjegyzések:

- a *mod* művelet az osztási maradék meghatározását jelenti, a % operátor matematikai jelöléséről van szó,
- az `n` számot a matematikában modulusnak hívják,
- moduláris hatványozás: a `pow` függvény meghívható három paraméterrel, ahol a harmadik paraméter a modulus értéket jelöli,

```
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
```

A kiíratás formázása

- a kiíratás formázását a `print`-ben a % operátorral végezzük:
  - két paramétert kell megadni,
  - baloldali paramétere egy `str` típusú érték, amelyben meghatározzuk, hogy a jobboldali paraméter milyen formátumú legyen,
  - a jobboldali paraméter lehet egy érték, vagy egy tuple,
  - a formázás hasonló a C/C++-hoz,
- a Python a % alkalamzásakor nem fogja összekavarni, hogy mikor kell maradékot számoljon, mikor kell a kiíratást formázza.

```
>>> '%7i' % 78
    '     78'

>>> import math
>>> '%7.2f' % math.pi
    '   3.14'

>>> x, y = 340, 120
>>> '%12s%12s' % (x, y)
    '         340         120'
```

Kiíratás szövegállományba

4. feladat

Írjunk egy Python-függvényt, amely adott n, a érték esetében kiírja egy szövegállományba $a^i \pmod n$ értékét minden $i = 1, 2, \ldots, n-1$ értékre.

```
def osztasiM1(n, a = 2):
    all = open('hatvany.txt', 'w')
    print('%3s%4s%12s%3s%3i' % ('a', 'i', '(a^i)', 'mod', n), file = all)
    for i in range(1, n):
        print('%3i%4i%10i' % (a, i, pow(a, i, n)), file = all)
    all.close()

def osztasiM2(n, a = 2):
    with open('hatvany.txt', 'w') as all:
        print('%3s%4s%12s%3s%3i' % ('a', 'i', '(a^i)', 'mod', n), file = all)
        for i in range(1, n):
            print('%3i%4i%10i' % (a, i, pow(a, i, n)), file = all)

>>> osztasiM1(11)
>>> osztasiM2(11, 3)
```

Kiíratás szövegállományba

5. feladat

Írjunk egy Python-függvényt, amely adott n érték esetében kiírja egy szövegállományba $a^i \pmod n$ *értékét minden* $i = 1,2,\ldots,n-1$ *és minden* $a = 1,2,\ldots,n-1$ *értékre.*

```
def osztasiM3(n):
    with open('hatvanyok.txt', 'w') as all:
        for i in range(1, n):
            print('%4i' % i, file = all, end = ' ')
            for a in range(1, n):
                print('%10i' % pow(a, i, n), file = all, end = ' ')
            print(file = all)

>>> osztasiM3(17)
```

Python hibakezelés

Ha az adabevitel során nem megfelelő értéket olvasunk be egy adott változóba, akkor futási hiba adódik:

```
>>> beolvasFloat(4)
    elem = szoveg
    ...
    ValueError: could not convert string to float: 'szoveg'
```

Az ilyen típusú hibák elkerülése végett a megfelelő műveletsorokat egy `try...except` utasításblokk közé kell írni:

```
def beolvasFloat(n):
    L = []
    for i in range(n):
        try:
            L += [float(input("elem = "))]
        except:
            print("hibas bemenet")
            return
    return L
```

- az `try` részbe azokat a műveletsorokat kell írni, amelyek futási hibát okozhatnak,
- az `except` részbe pedig a hibaüzenetet, illetve a függvényből való kilépést kell megadni.

Python függvények paraméterátadása

```
def foF1():
    db = 0
    segedF1(db)
    print (db)

def segedF1(x):
    x += 10
```

```
>>> foF1()
    0
```

A `foF1` függvényben nem látjuk, hogy a `segedF1` függvényben megváltoztattuk az `X` paraméter értékét.

```
def foF2():
    db = 0
    db = segedF2(db)
    print (db)

def segedF2(x):
    x += 10
    return x
```

```
>>> foF2()
    10
```

A `foF2` függvényben látjuk, hogy a `segedF2` függvényben megváltoztattuk az `X` paraméter értékét.

Számok

- számtartományok: természetes számok, egész számok, racionális számok, irracionális számok, valós számok, komplex számok,
- alapműveletek számokkal: összegzés, különbség, szorzás, osztás, hatványozás, logaritmálás,
- természetes számok, egész számok: a diszkrét matematika gerincét alkotják,

Természetes számok

- halmazjelölés: $\mathbb{N} = \{0, 1, 2, 3, \dots\}$,
- tulajdonságok:
  - az összeadás, szorzás kommutatív: $a, b \in \mathbb{N}, a + b = b + a, a \cdot b = b \cdot a$
  - az összeadás, szorzás asszociatív: $a, b, c \in \mathbb{N}, (a + b) + c = a + (b + c), (a \cdot b) \cdot c = a \cdot (b \cdot c)$
  - az összeadás a szorzásra nézve disztributív: bármely $a, b, c \in \mathbb{N}, (a + b) \cdot c = a \cdot c + b \cdot c$
  - a természetes számok halmaza zárt az összeadásra, szorzásra nézve: bármely két természetes szám összeadható, szorozható az eredmény szintén természetes szám lesz, ez nem igaz a különbség és osztás műveletekre,
  - a 0 az összeadásra nézve semleges, az 1 a szorzásra nézve semleges elem,
  - jól rendezettség: a természetes számok minden nem üres részhalmazának van egy legkisebb eleme.

A hatványozás algoritmusa

6. feladat

Írjunk egy Python függvényt, amely meghatározza x^n értékét, de ne használjunk beépített függvényeket.

- többféle megoldás létezik,
- a legkézenfekvőbb algoritmus az, ha `n`-szer összeszorozzuk az `x`-et, ezt a gondolatmenetet követi a `myPowLin` függvény:

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

- az algoritmus futási ideje azonban nem jó, nagy számokra lassú,
- lemérhetjük a futási időt különböző bemenetekre: ha a hatványkitevő 500000 akkor már több másodpercig is eltarthat a számolási idő,
- megszámolhatjuk hány alapműveletet (szorzást) végez az algoritmus: ha a hatványkitevő 100, akkor 100 szorzást végez,
- az algoritmus futási ideje lineáris.

A gyorshatványozás algoritmusa

- x^n értékét meghatározhatjuk a gyorshatványozás algoritmusával: ha a hatványkitevő 100, akkor 9 darab szorzást végez az algoritmus,
- az algoritmus futási ideje logaritmikus

```
def myPow(x, n):
    res = 1
    while n != 0:
        if n % 2 == 1:
            res = res * x
        n = n // 2
        x = x * x
    return res
```

```
>>> myPow(2, 100)
    1267650600228229401496703205376L
```

```
def myPow1(x, n):
    res = 1
    while True:
        if n % 2 == 1:
            res = res * x
        n = n // 2
        if n == 0: break
        x = x * x
    return res
```

A gyorshatványozás algoritmusa

3^{100} meghatározása:

```
def myPow(x, n):
    res = 1
    while n != 0:
        if n % 2 == 1: res = res * x
        n = n // 2
        x = x * x
    return res
```

x	n	res
		1
3	100	1
$3^2 = 9$	50	1
$3^4 = 81$	25	81
$3^8 = 6561$	12	81
$3^{16} = 43046721$	6	81
$3^{32} = 1853020188851841$	3	150094635296999121
$3^{64} = 3433683820292512484657849089281$	1	515377520732011331036461129765621272702107522001
$3^{128} = 117\ldots961$	0	

Az eredmény: $3^{100} = 3^4 \cdot 3^{32} \cdot 3^{64} = 515377520732011331036461129765621272702107522001$.

Futási idők

7. feladat

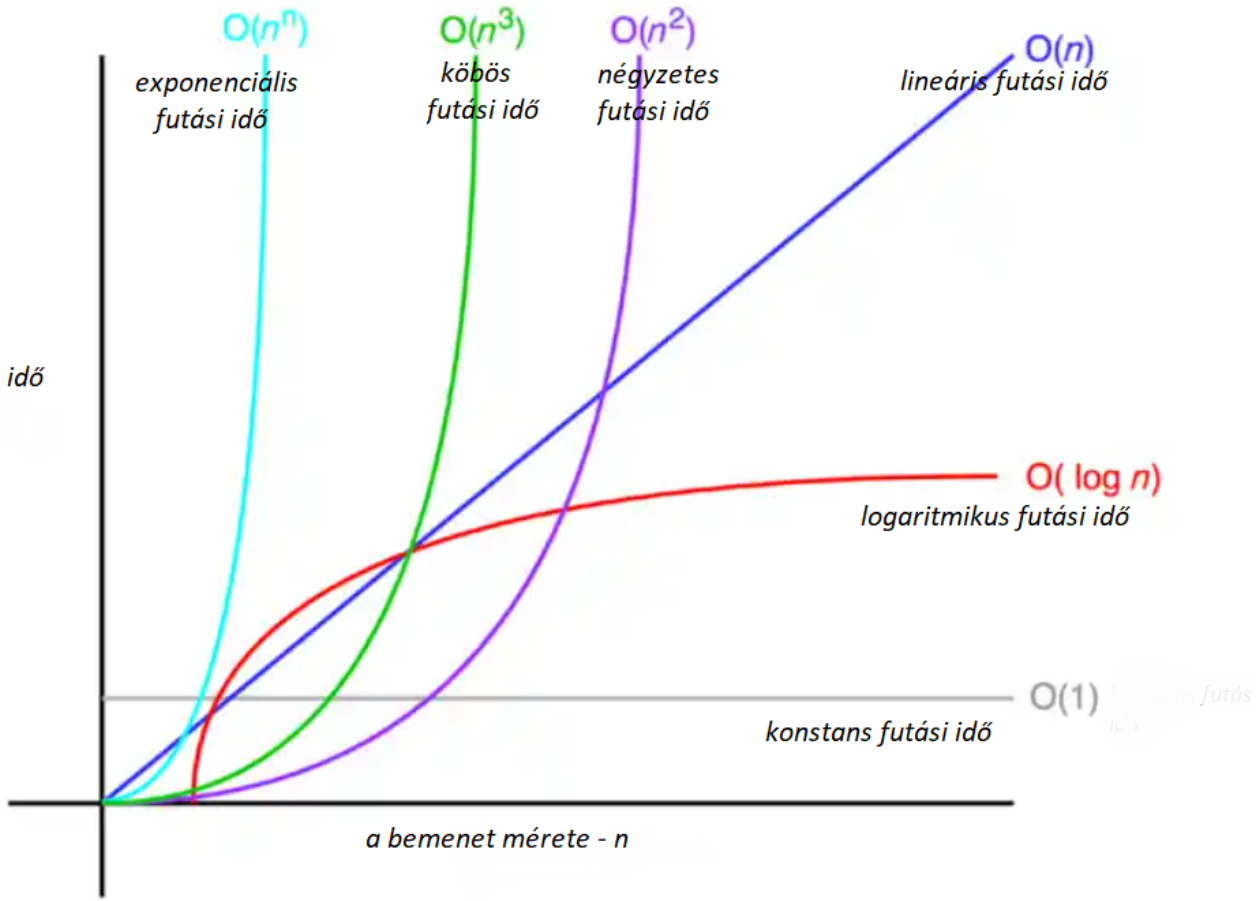
Írjunk Python függvényt, amely meghatározza különböző nagyságrendű bemenetek esetén a `myPowLin` és `myPow` függvények futási idejét.

```
from time import time

def mainTimeLog(x):
    for n in range(1, 10):
        st = time()
        myPow(x, 10**n)
        fs = time()
        print("kitevo: 10^",n, ", futasi ido:", fs - st)

def mainTimeLin(x):
    for n in range(1, 7):
        st = time()
        myPowLin(x, 10**n)
        fs = time()
        print("kitevo: 10^",n, ", futasi ido:", fs - st)
```

Futási idők


O(n^n)
exponenciális futási idő
O(n^3)
köbös futási idő
O(n^2)
négyzetes futási idő
O(n)
lineáris futási idő
idő
O(log n)
logaritmikus futási idő
O(1)
konstans futási idő
a bemenet mérete - n