PYTHON FELADATOK

CSONKA BENCE, SIMON ANDRÁS

1. Tudnivalók

- Hol van Python?
  - Bejelentkezés `tarski.math.bme.hu`-ra; `leibniz`-en egy terminálból pl. így:

    ```
    ssh -Y tarski
    ```

    aztán `spyder3` vagy egy terminálban `ipython3`.
  - Saját windows-os gépen: `http://wiki.math.bme.hu/view/AnacondaInstall` és aztán `spyder3`.
  - `https://colab.research.google.com/` vagy `https://cocalc.com`. Regisztrálni kell, de szép jupyter notebook-ot kapunk.
  - `https://sagecell.sagemath.org/`, válasszuk a Python-t. (Ez csak végszükségben javasolt megoldás.)
- Jegyzet: `http://math.bme.hu/~asimon/info2/python.pdf` .

2. Feladatok 0-ról indulóknak

Feladat 2.1. Írjon egy programot, amely Celsius fokokban írja ki az `f` változóban tárolt, Fahrenheitben megadott hőmérsékletet. Használja a $T_c = \frac{5}{9}(T_f - 32)$ képletet!

Feladat 2.2. Írjon olyan programot, amely kiírja az `n` változóban tárolt pozitív egész szám alatti összes természetes számot. Egy karakter hozzáadásával változtassa meg a prgramját úgy, hogy az `n`-nél nem nagyobb számokat írja ki.

Feladat 2.3. Írjon olyan programot, amely az `n` változóban tárolt pozitív egész számnál nem nagyobb számok összegét írja ki. Tegye ugyanezt úgy, hogy csak a páros számokat összegzi. (Ehhez ne használjon `if`-et.)

Feladat 2.4. Írjon olyan programot, amely kiírja az `n` változóban tárolt pozitív egész szám faktoriálisát.

Feladat 2.5. Írjon egy programot, amely az `n`-ben tárolt pozitív egész szám (tizes számrendszerbeli) számjegyeit fordított sorrendben írja ki. (Egész számok osztása: `//`)

Feladat 2.6. Írjon egy programot, amely kiírja az `n`-ben tárolt pozitív egész szám (tizes számrendszerbeli) számjegyeinek összegét.

Feladat 2.7. Írjon programot, amely kiírja a Fibonacci sorozat első `n` tagját. (Ez nem olyan egyszerű csak `while`-t használva. Segítség: jegyezze meg az utolsó két tagot.) Emlékeztető: $F_0 = 0$, $F_1 = 1$ és $F_{n+2} = F_n + F_{n+1}$.

Feladat 2.8. Oldja meg a 2.2-2.4 feladatokat `while` helyett `for`-ral.

Date: April 24, 2024.

Feladat 2.9. Írjon programot (`continue` használatával és anélkül), amely kiírja az `l` számlistabeli *pozitív* számok összegét. Használjon `for` ciklust.

Feladat 2.10. Írjon programot `for` ciklus és `break` használatával, amely kiírja az `n` változóbeli pozitív `int` első valódi osztóját, ha van ilyen, és semmit nem csinál egyébként.

Feladat 2.11. Írjon egy kétargumentumú (mindkettő szám) `compare()` függvényt, amely az írja ki, hogy `A számok egyenlőek.` ha a két argumentum egyenlő, `Az első szám nagyobb.`, ha ez a helyzet, egyébként meg azt, hogy `A második szám nagyobb.`.

Feladat 2.12. Írjon egy kétargumentumú (mindkettő szám) `divisible()` függvényt, amely `True`-t ad vissza ha az első argumentum osztja a másodikat, és `False`-t egyébként.

Feladat 2.13. Írjon (az `if` parancs, és az `if` kifejezés használatával is) egy egyargumentumú `absval()` függvényt, amely az argumentumaként adott szám abszolútértékét adja vissza. (Ezt teszi a beépített `abs()` függvény is.) Szükség van `else`-re az `if` parancsot használó változatban?

Feladat 2.14. Mit fognak kinyomtatni az alábbi programrészletek?

```
x = 10

def f():
  x = 5

f()
print(x)
```

```
x = 10

def f():
  x = 5
  return x


x = f()
print(x)
```

Feladat 2.15. Ha `s = 'abcdefghij'`, akkor mi a következő kifejezések értéke?

- `s[3]`
- `s[-3]`
- `s[3:6]`
- `s[3:-4]`
- `s[4::2]`
- `s[-4::2]`
- `s[3::2]`
- `s[5::-2]`
- `s[5:1:-2]`

Feladat 2.16. Legyen `numbers = list(range(5))`. Mi a különbség `numbers[1] = [True]` és `numbers[1:2] = [True]` között? Ekvivalens-e valamelyikük `numbers[1] = True`-val?

Feladat 2.17. Állítsa elő az első 10 páros szám listáját (a) listagenerálással (kétféleképpen), és (b) egy alkalmas `list`a szeleteként.

Feladat 2.18. Állítsa elő egymásba ágyazott listagenerálások segítségével a

```
[[[0, 2], [0, 3], [0, 4]], [[1, 2], [1, 3], [1, 4]]]
```

és a

```
[[[0, 2], [1, 2]], [[0, 3], [1, 3]], [[0, 4], [1, 4]]]
```

listát.

Feladat 2.19. Tegyük fel, hogy `a == 1`, `b == 2` és `c == 3`. Hogyan lehetne egyetlen utasítással elérni, hogy `a == 2`, `b == 3` és `c == 1` legyen?

Feladat 2.20. Állítsa elő az első 10 páros szám listáját egy ciklus és az `.append()` metódus segítségével.

Feladat 2.21. Írjon egy `solve2()` függvényt, amelyet egy másodfokú egyenlet három együtthatójával kell meghívni, és eredményül az egyenlet *valós* gyökeinek listáját adja vissza. Például

```
>>> solve2(1,1,-2), solve2(1,0,2)
([1.0, -2.0], [])
```

mert $x^2+x-2=(x-1)(x+2)$ és mert x^2+2-nek nincs valós gyöke. A négyzetgyökök kiszámításához használja a `math` modul `.sqrt()` függvényét!

Feladat 2.22. Írjon egy `sid(x, y)` (*s*afe *i*nteger *d*ivision) függvényt, amely `int` típusú `x`, `y`-ra a hányadosukat adja vissza `int` típusú értékként ha `y` osztja `x`-et, egyébként pedig az `x/y` `float`-ot. Például,

```
>>> [sid(x,2) for x in [1, 4, 18530201888518410]]
[0.5, 2, 9265100944259205]
```

Feladat 2.23. Írja meg az alábbi programrészletet `elif` használata nélkül!

```
>>> for i in [-5,5,15,25]:
...     if i<0:
...         print("negative")
...     elif i<=10:
...         print("small")
...     elif i<=20:
...         print("medium")
...     else:
...         print("big")
...
negative
small
medium
big
```

Feladat 2.24. Írjunk `break` és `else` használata nélkül egy `first_divisible(numbers,d)` függvényt, ami kiírja a `numbers` számlista első olyan tagját, amely osztható d-vel, vagy ha nincs ilyen, akkor ezt a tényt. Például:

```
>>> nums = [2, 5, 10, 14, 21, 35, 42, 51]
>>> first_divisible(nums, 7)
14 osztható 7-val/vel

>>> first_divisible(nums, 13)
Nincs a listában 13-val/vel osztható szám.
```

3. Vegyes nehézségű feladatok

Feladat 3.1. Írjon egy egyargumentumú függvényt, ami az $n_1, n_2, \dots, n_k$ számokat tartalmazó listával meghívva az $n_1^3 - 1, n_2^3 - 1, \dots, n_k^3 - 1$ számokat tartalmazó listát adja vissza. Csinálja meg ezt a feladatot listagenerálás segítségével és anélkül is.

Feladat 3.2. Írjon egy kétargumentumú `squares()` függvényt úgy, hogy `squares(m,n)` az m és n közti négyzetszámok listáját adja vissza. Csinálja meg ezt a feladatot listagenerálás segítségével és anélkül is.

Feladat 3.3. Általánosítsa az előző feladatbeli `squares()`-t tetszőleges pozitív egész kitevőkre! Vagyis a `squares()` valamelyik implementációjából kiindulva írjon egy olyan `powers()` függvényt, amely `powers(m,n,k)`-ra az m és n közti k. hatványok listáját adja vissza.

Feladat 3.4. Írjon egy kétargumentumú `index_of()` függvényt, amelynek argumentumai egy tetszőleges objektum és egy lista, és az objektum első listabeli előfordulásának indexét adja vissza, vagy `None`-t ha az objektum nem fordul elő a listában. (Emlékeztető: ha a függvény nem ad vissza semmit `return` segítségével, akkor `None`-t ad vissza.)

Feladat 3.5. Írjon egy kétargumentumú `indices_of()` függvényt, amelynek argumentumai egy tetszőleges objektum és egy lista, és az objektum listabeli előfordulásai indexeinek listáját adja vissza Például:

```
>>> indices_of(3,[4,1,3,2,3])
[2, 4]
>>> indices_of(0,[4,1,3,2,3])
[]
```

Feladat 3.6. Írjon egy `substitute()` függvényt, amelynek első argumentuma egy lista, a második és harmadik pedig tetszőleges objektumok, és ami olyan új listát ad vissza, amely pontosan ugyanazokat az elemeket tartalmazza ugyanabban a sorrendben, mint az eredeti, kivéve, hogy ahol az eredetiben a második argumentum szerepel, ott az eredményben a harmadik. Például:

```
>>> substitute([1,2,3,4,2],2,'a')
[1, 'a', 3, 4, 'a']
```

Feladat 3.7. Írja meg az előző feladatbeli `substitute()` egy olyan változatát, ami az első argumentumként adott listán végzi el a helyettesítést, és semmit sem ad vissza. Például:

```
>>> mylist = [1,2,3,4,2]
>>> substitute(mylist,2,'a')
>>> mylist
[1, 'a', 3, 4, 'a']
```

Feladat 3.8. Definiáljon egy `divisibles_by()` kétargumentumú függvényt: az első argumentum egy lista, a másik egy egész szám lesz. A függvény a lista azon tagjainak listáját adja vissza, melyek oszthatók a számmal.

Például:

```
>>> divisibles_by(list(range(30,50)),7)
[35, 42, 49]
```

Feladat 3.9. Definiáljon egy `divisors()` függvényt, amely egyetlen argumentuma valódi osztóinak listáját adja vissza.

Feladat 3.10. Definiálja az egyargumentumú `sigma()` függvényt, ami a $\sigma(n) = \sum_{1 \le d \le n, d|n} d$ számelméleti függvényt számítja ki. Ellenőrizze ezt néhány számra! Például:

```
>>> [(n,sigma(n)) for n in range (6,60,11)]
[(6, 12), (17, 18), (28, 56), (39, 56), (50, 93)]
```

Feladat 3.11. Írjon egy `is_perfect()` függvényt, amely a `True` értéket adja vissza ha az argumentuma tökéletes szám (ami azt jelenti, hogy egyenlő nála kisebb osztóinak összegével), egyébként pedig `False`-t. Ennek segítségével írja ki a 10000 alatti tökéletes számokat.

Feladat 3.12. Írjon egy `divisors_ival()` függvényt úgy, hogy `divisors_ival(m,n)` kiírja a m és n közötti összes egész szám valódi osztóinak listáját, így:

```
>>> divisors_ival(30,35)
30 -> [2, 3, 5, 6, 10, 15]
31 -> []
32 -> [2, 4, 8, 16]
33 -> [3, 11]
34 -> [2, 17]
35 -> [5, 7]
```

Feladat 3.13. Írjon egy `is_prime()` függvényt, ami `True`-t ad vissza ha az argumentuma prím, és `False`-t egyébként.

Feladat 3.14. Írjon egy `primes_between()` függvényt úgy, hogy `primes_between(m,n)` az $[m, n]$ intervallumbeli prímek listáját adja vissza.

Feladat 3.15. Írjon egy `prime_divisors()` függvényt, ami argumentuma prímosztóinak listáját adja vissza.

Feladat 3.16. Írjon egy `max_exp()` függvényt úgy, hogy `max_exp(m,n)` a legnagyobb olyan k-t adja vissza, amire $m^k \mid n$. Felteheti, hogy $m > 1$.

Feladat 3.17. Írjon egy `prime_decomp()` függvényt, amely az argumentuma prímfelbontását adja vissza olyan párok (vagy listák) listájaként, amelyek első tagja az argumentum prímosztója, a második pedig e prím kitevője az argumentum prímfelbontásában. Például:

```
>>> prime_decomp(90)
[(2, 1), (3, 2), (5, 1)]
```

Feladat 3.18. Írjon egy kétargumentumú `lookup()` függvényt. A második argumentum párok listája, az első pedig egy "kulcs". A `lookup(kulcs, lista)` a listabeli első olyan pár második tagját adja vissza, amelynek első tagja `kulcs`, vagy `None`-t, ha nincs ilyen.

Feladat 3.19. A két előző feladatbeli függvényt felhasználva írjon egy `gcd()` függvényt, amely két argumentumának legnagyobb közös osztóját adja vissza.

Feladat 3.20. Definiáljon egy, az Euler-féle φ függvényt kiszámító függvényt, azaz olyat, amely n-re az n-nél kisebb, n-hez relatív prím számok számát adja vissza.

Feladat 3.21. Definiáljon egy `separate()` nevű függvényt, amely az argumentumaként megadott számlista negatív ill. nemnegatív elemei listáinak párját adja vissza.

Feladat 3.22. Írjon egy egyargumentumú `is_sorted()` függvényt, amely `True`-t ad vissza, ha az argumentumaként megadott, számokból álló lista rendezett, és `False`-t ha nem.

Segítség: ha sehogy sem akar menni, nézze meg a `repeats()` függvény definícióját a jegyzetben!

Feladat 3.23. Írjon egy függvényt, amely az argumentumaként megadott, számokból álló lista legkisebb elemét adja vissza. Legyen a függvény neve `my_min()`, mert van `min()` nevű beépített függvény, ami ugyanezt teszi, és amit ezért aztán ne használjon. De gondolkodjon el rajta, hogyan lehetne még csalni! Egy ötlet: van beépített `max()` függvény is.

Feladat 3.24. Írjon egy `min_index()` függvényt, amely az argumentumaként megadott, számokból álló lista legkisebb eleme első előfordulásának indexét adja vissza. (Inkább módosítsa a fenti `my_min()` függvény definícióját, mint hogy meghívja azt. De az a legjobb, ha ilyen és olyan változatot is ír!)

Feladat 3.25. Írjon egy `min_indices()` nevű függvényt, amely az argumentumaként megadott, számokból álló lista legkisebb eleme előfordulásai indexeinek listáját adja vissza. Például:

```
>>> min_indices([1,3,4,2,1,3,1,2])
[0, 4, 6]
```

Feladat 3.26. Írjon egy `nearest_to_avg()` nevű függvényt, amely az argumentumaként adott számlista tagjai átlagához legközelebbi tagját adja vissza (ha több ilyen is van, akkor a legkisebb indexűt), ill. `None`-t, ha a lista üres. Használhatja az `abs()` függvényt.

Feladat 3.27. Írjon egy `has_duplicates()` nevű függvényt, amely `True`-t ad vissza, ha valamilyen objektum legalább kétszer fordul elő az argumentumaként megadott listában, és `False`-t, ha nincs ilyen.

Feladat 3.28. Írjon egy `longest_run()` függvényt, amely az argumentumaként megadott számlista konstans "részlistái" közül a leghosszabb hosszát adja vissza. Például:

```
>>> longest_run([])
0
>>> longest_run([1])
1
>>> longest_run([1,2,3])
1
>>> longest_run([1,2,3,3,4])
2
>>> longest_run([1,2,2,2,1,2,3,3,4])
3
```

Feladat 3.29. Írassa ki az alábbi, kissé kajla szorzótáblát!

```
1: 1 2 3 4 5 6 7 8 9
2: 2 4 6 8 10 12 14 16 18
3: 3 6 9 12 15 18 21 24 27
4: 4 8 12 16 20 24 28 32 36
5: 5 10 15 20 25 30 35 40 45
6: 6 12 18 24 30 36 42 48 54
```

```
7: 7 14 21 28 35 42 49 56 63
8: 8 16 24 32 40 48 56 64 72
9: 9 18 27 36 45 54 63 72 81
```

Segítség: `print(42, end='akármi')` 42-t ír ki, és aztán új sor helyett `akármi`-t. `print()` önmagában új sort ír ("sort emel").

Feladat 3.30. Írassa ki egyenként 012 összes permutációját, így:

```
0 1 2
0 2 1
1 0 2
1 2 0
2 0 1
2 1 0
```

Módosítsa a programját úgy, hogy az összes ilyen permutáció listáját írja ki!

```
[[0, 1, 2], [0, 2, 1], [1, 0, 2], [1, 2, 0], [2, 0, 1], [2, 1, 0]]
```

Feladat 3.31. Írjon függvényt amely egy természetes szám faktoriálisát adja vissza! Írjon rekurzív és iteratív változatot is! (V.ö. 2.4, ahol nem volt választási lehetőségünk!)

Feladat 3.32. Írjon egy `lucas(n)` függvényt, amely az n-edik Lucas-számot adja vissza. A Lucas-számok a következő rekurzóval vannak definiálva:

$$L_0 = 2, \qquad L_1 = 1, \qquad L_{n+2} = L_n + L_{n+1}$$

Próbálja megírni a függvényt ennek a definíciónak a mintáját követve, de írjon egy másik, nem rekurzív változatot is.

Házi feladat 3.1. Az előző gyakorlat `lucas()` függvényét használva nyomtassa ki a Lucas-számok azon részsorozatának első néhány tagját, amelynek indexei oszthatók 3-mal. Észrevesz valamilyen szabályszerűséget? Ha igen, ellenőrizze ezt az első néhány tucat tagra! (Használja a nem rekurzív verziót, mert a rekurzív már elég kis számokra is túl lassú!) Érvényes-e ez a szabályszerűség, ha az olyan indexűeke nézzük, amelyek maradéka 1 vagy 2 modulo 3? Próbáljon megfogalmazni egy állítást és ellenőrizze le az első néhány száz Lucas-számra! Ha ezekre igaz, próbálja meg bebizonyítani.

Feladat 3.33. A Goldbach-sejtés azt mondja, hogy minden 2-nél nagyobb páros szám két prím összege. Írjon egy `goldbach()` függvényt, amely $1 < n \in \mathbb{N}$-re $2n$ ilyen felbontásainak számát adja vissza. ($p_1 + p_2$ és $p_2 + p_1$ nem számít külön felbontásnak.)

Feladat 3.34. Az előző feladatbeli `goldbach()` függvény segítségével vizsgálja meg különféle intervallumokra, hogy mi az ottani páros számok "Goldbach-felbontásainak" minimális száma.

Például nézze meg ezt a 2000, a 20000 és a 200000 után következő első 100 páros számra!

Feladat 3.35. Írjon egy `pascal()` függvényt úgy, hogy `pascal(n)` a Pascal háromszög első n sorát (mint listák listáját) adja vissza. Használja a Pascal háromszög rekurzív definícióját (ami nem jelenti azt, hogy a definiált függvénynek feltétlenül rekurzívnak kell lennie, ld. pl. 3.31!): az n. sor n hosszú; az első sor egyetlen egyesből áll, és ha sorokat úgy toljuk el egymáshoz képest, hogy háromszöget alkossanak, akkor minden további sor minden eleme az előző sorban felette balra és jobbra található elemek összege, beleértve, hogy a hiányzó elemeket 0-nak tekintjük (Wikipedia). Például:

```
>>> pascal(5)
[[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1]]
```

Jól jöhet, hogy pl. `n*[1]` csupa egyesekből álló n hosszú listát hoz létre.

Feladat 3.36. Egy asszociatív tömb olyan adatszerkezet, amely kulcsokhoz rendel értékeket. (Python-ban van ilyen, `dict`-nek hívják, és lesz majd róla szó, de ebben a feladatban megmutatjuk, hogy nélküle is boldogulnánk.)

Egy asszociatív tömbbel négyféle dolgot szokás csinálni: létrehozni, kulcs-érték párokat tenni bele, megnézni, hogy egy kulcshoz milyen érték tartozik, és törölni belőle egy kulcs-érték párt.

Reprepzentáljunk egy asszociatív tömböt a benne szereplő kulcs-érték párok listájával (és egy ilyen pár legyen egy tuple). Akkor az utolsó előtti feladatot már megoldottuk 3.18-ben. Az első könnyű:

```
def make_new_aa(): return list()
```

A feladat a maradék két függvény megírása. `insert(key,value,aa)` `aa` egy másolatát adja vissza, de úgy, hogy ha `key` nem szerepel `aa`-ban, akkor hozzáadjuk a `(key,value)` párt, ellenkező esetben pedig a benne szereplő `(key,value')` párt lecseréljük `(key,value)`-ra.

Végül `delete(key,aa)` az `aa` olyan másolatával térjen vissza, amiben nincs `(key,value)` alakú pár.

Például:

```
>>> d = make_new_aa()
>>> d = insert('one',1,d); d = insert('two',2,d); d = insert('three',3,d)
>>> d
[('one', 1), ('two', 2), ('three', 3)]
>>> lookup('two',d)
2
>>> d = insert('two',22,d); d = insert('four',4,d); d
[('one', 1), ('three', 3), ('two', 22), ('four', 4)]
>>> d = delete('five',d); d
[('one', 1), ('three', 3), ('two', 22), ('four', 4)]
>>> d = delete('two',d); d
[('one', 1), ('three', 3), ('four', 4)]
```

Feladat 3.37. Irjon egy `insert_sort()` nevű függvényt, ami az argumentumaként megadott számlista rendezett változatát adja vissza.

Használja a beszúrásos rendezés algoritmust: “Az algoritmus k-adik lépése előtt az első k-1 elem már rendezett; a lépés során a k-adik elemet beszúrjuk az első k-1 elem közé az őt nagyság szerint megillető helyre, és a nála nagyobbakat eggyel eltoljuk.”

4. I/O

Feladat 4.1. Írjon egy programot, ami számokat kér egymás után, és ha a felhasználó `RETURN`-t nyom kétszer egymás után, akkor kinyomtatja az addig bevitt számok átlagát.

Tehát a program egy futása ehhez hasonlóan nézhet ki:

```
Írjon be egy számot: 1
Írjon be egy számot: 4
Írjon be egy számot: 12
Írjon be egy számot: 3
```

```
Írjon be egy számot:
Az átlag 5.0
```

Feladat 4.2. Írjon egy proramot, ami szóközökkel elválasztott számokat kér, és kiírja az átlagukat. Tehát a program egy futása ehhez hasonlóan nézhet ki:

```
>>> Írjon be néhány számot szóközökkel elválasztva: 1 4 13 2
Az átlag  5.0
>>>
```

Feladat 4.3. Írjon egy `read_first_lines()` nevű kétargumentumú függvényt, aminek az első argumentuma egy file neve, a második pedig egy n természetes szám. A függvény nyomtassa ki a file első n sorát (vagy az összeset, ha n-nél kevesebb sora van).

Feladat 4.4. Írjon egy `copy_first_lines()` nevű háromargumentumú függvényt, aminek az első két argumentuma egy input és egy output file neve, a harmadik pedig egy természetes szám. Csinálja azt a függvény, amit az előző feladatbeli, csak most az első argumentuma által megnevezett file-ból olvasson, és a második által megnevezetbe írjon. Ellenőrizze az eredményt egy szövegszerkesztővel, vagy parancssorból `cat` vagy `less` segítségével.

Feladat 4.5. Írjon egy `count_lines()` nevű egyargumentumú függvényt, ami egy file neve lesz. A függvény a file-ban található sorok számát adja vissza. (Mint parancsorban a `wc -l` parancs.)

Feladat 4.6. Írjon egy `read_to_string()` nevű függvényt, ami az argumentumaként megadott szövegfile tartalmát egyetlen string-ként adja vissza. Ez a string ne tartalmazzon újsor karaktereket (`\n`), de minden sor utolsó szavát válassza el legalább egy szóköz a következő sor első szavától.

Segítség: string-eket össze lehet fűzni a `+` operátorral. És használja az előadásbeli `.rstrip()`-et!

5. Tömbök

Feladat 5.1.* Miért működik az alábbi kódrészlet? Nem mond ez ellent annak, hogy a `tuple`-k nem módosíthatók?

```
>>> tup = ([0,1],[2]) ; tup[0][1] = 'a' ; tup
([0, 'a'], [2])
```

Segítség: próbáljuk ki itt!

Feladat 5.2. Írjon egy `list_diff()` nevű kétargumentumú függvényt, amelynek mindkét argumentuma lista lesz, és eredményül az első lista azon tagjainak listáját adja vissza (eredeti sorrendjükben), amelyek nem fordulnak elő a másodikban. Például:

```
>>> list_diff(list(range(10)),list(range(0,15,3)))
[1, 2, 4, 5, 7, 8]
```

Feladat 5.3. Nyomtassa ki a következőt:

```
1
22
333
4444
55555
666666
```

```
7777777
88888888
999999999
```

Feladat 5.4. Írjon egy kétargumentumú `wave()` függvényt, aminek első argumentuma a hullám (ld. alább!) amplitúdója (ami a hosszát is meghatározza), a második pedig a hullámok száma. Tehát hogy pl. `wave(5,2)` az alábbi ábrát nyomtassa:

```
o
oo
ooo
oooo
ooooo
ooooo
oooo
ooo
oo
o
o
oo
ooo
oooo
ooooo
ooooo
oooo
ooo
oo
o
```

Ha működik, írjon egy `new_wave()` függvényt, ami csak abban különbözik `wave()`-től, hogy csak egy maximális "magasságú" sor van. Pl. `new_wave(5,1)` ezt nyomtassa ki:

```
o
oo
ooo
oooo
ooooo
oooo
ooo
oo
o
```

Feladat 5.5. Írjon egy `merge()` függvényt, amit két listával lehet meghívni, és egy harmadik listát ad vissza, amit az argumentumai összefésültje, azaz: az eredmény első tagja az első argumentum első tagja, a második tagja a második argumentumának elsŐ tagja, a harmadik az első argumentumának második tagja, és így tovább.

Első menetben felteheti, hogy a két lista azonos hosszúságú. De a végső változatban az eredmény végén legyenek a hosszabb lista maradék elemei. Például:

```
>>> merge(list(range(5)),list(range(10,18)))
[0, 10, 1, 11, 2, 12, 3, 13, 4, 14, 15, 16, 17]
```

Feladat 5.6. Írjon egy `cat()` függvényt, ami az argumetumaként megadott nevű file sorait írja ki nagybetűkkel.

Feladat 5.7. Írjon egy függvény, amelynek egyetlenargumentuma egy szövegfile neve. A file-ban lebegőpontos számok vannak, soronként egy. A függvény e számot átlagát adja vissza.

Segítség: 1. Használja a `float()` függvényt a lebegőpontos számok string reprezentációjának `float`-tá konvertálására. 2. `enumerate()` szövegfile-okra is működik.

Feladat 5.8. Írjon egy `transpose()` függvényt, ami egy listák listájaként reprezentált kétdimenziós mátrix transzponáltját adja vissza. Például:

```
>>> transpose([[1,2,3],[4,5,6],[7,8,9]])
[[1, 4, 7], [2, 5, 8], [3, 6, 9]]
```

Az eredményül visszaadott mátrixot új listaként állítsa elő, ne változtassa meg a függvény argumentumát!

Feladat 5.9. Írjon egy `matrix_add()` függvényt, ami az argumentumaként megadott két (ugyanolyan típusú) mátrix összegét adja vissza. A mátrixokat a soraikból álló lista, a sorokat számlisták reprezentálják. Például:

```
>>> m1 = [[1,0,0],[0,1,0],[0,0,1]]
>>> m2 = [[1,2,3],[4,5,6],[7,8,9]]
>>> matrix_add(m1,m2)
[[2, 2, 3], [4, 6, 6], [7, 8, 10]]
```

Az eredményül visszaadott mátrixot új listaként állítsa elő, ne változtassa meg a függvény egyik argumentumát sem!

Feladat 5.10. Írjon egy `matrix_mult()` függvényt, ami az argumentumaként megadott két (megfelelő típusú) mátrix szorzatát adja vissza. A mátrixokat a soraikból álló lista, a sorokat számlisták reprezentálják. Például:

```
>>> matrix_mult([[1,2],[3,4],[5,6]],[[1,0,0],[0,1,0]])
[[1, 2, 0], [3, 4, 0], [5, 6, 0]]
>>> matrix_mult([[1,0,0],[0,1,0]],[[1,2],[3,4],[5,6]])
[[1, 2], [3, 4]]
```

Az eredményül visszaadott mátrixot új listaként állítsa elő, ne változtassa meg a függvény egyik argumentumát sem!

Feladat 5.11. Írjon egy kétargumentumú (mindkettő lista vagy string) `myzip2()` függvényt, amely egy tuple-kből álló listát ad vissza, aminek hossza a rövidebb argumentum hossza. A lista j-edik tagjának i-edik tagja az i-edik argumentum j-edik tagja legyen. Például:

```
>>> myzip2('abcdefg', list(range(4)))
[('a', 0), ('b', 1), ('c', 2), ('d', 3)]
```

Ne használja a `zip()` függvényt!

Feladat 5.12. Írjon egy `date_to_day()` nevű függvényt, amely, megadva egy hónapot és az adott hónapban egy napot, visszaadja az adott nap sorszámát az évben. Például:

```
>>> date_to_day(3,15)      #31+28+15
74
```

Felthetjük, hogy az év nem szökőév: azaz a hónapok hossza az, ami a következő listában található:

```
MONTHS = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
```

Feladat 5.13. Írjon egy `day_to_date()` nevű függvényt, amely a `date_to_day()` függvény inverze. Azaz, megadva egy 1 és 365 közötti számot, a megfelelő dátumot (hónap, nap párost) adja vissza. Például:

```
>>> day_to_date(74)
(3, 15)
```

Mint az előző feladatban, feltehetjük, hogy az év nem szökőév és használhatjuk az ottani `MONTHS` listát.

Feladat 5.14. Definiáljon egy `lindex(string, substring)` függvényt, ami −1-el tér vissza, ha `substring` nem része `string`-nek, ellenkező esetben pedig `string` azon karakterének indexével, ahol `substring` első előfordulása kezdődik. Ez majdnem az, amit az `.index()` nevű metódus csinál, úgyhogy azt ne használja. Például:

```
>>> lindex("A mai vacsora egy csoda", "cso")
8
>>> lindex("A mai vacsora egy csoda", "csom")
-1
```

Feladat 5.15. Írjon egy `count_occurrences()` nevű függvényt, amely két stringet kap argumentumként, és a második elsőben való előfordulásainak számát adja vissza. Például:

```
>>> count_occurrences("A mai vacsora egy csoda", "cso")
2
>>> count_occurrences("A mai vacsora egy csoda", "csom")
0
```

Feladat 5.16. Tegyük fel, hogy van egy `inventory.csv` nevű szövegfile-unk, amiben minden sor

áru, *mennyiség*, *egységár*

szerkezetű. Vagyis minden sorban három "mező" van, vesszővel elválasztva. (Ezt a formátumot `csv`-nek ("comma separated values") hívják.) Írjon egy rövid programot, ami a file minden sorát a

`Áru:` *áru*
`Mennyiség:` *mennyiség*
`Egységár:` *egységár*

formátumban írja ki.

Például ha `inventory.csv` ezt tartalmazza:

```
>>> import os
>>> print(os.popen("cat inventory.csv").read())
ball,570,0.13
table,3,2000
racket,12,185
net,17,23
```

akkor az eredmény így nézzen ki:

Áru: ball
Mennyiség: 570
Egységár: 0.13
Áru: table

Mennyiség: 3
Egységár: 2000
Áru: racket
Mennyiség: 12
Egységár: 185
Áru: net
Mennyiség: 17
Egységár: 23
Használjon `f-string`-et!

Feladat 5.17. Módosítsa az előző feladatra adott megoldását úgy, hogy a következő formában írja ki a sorokat:

`Áru:` *áru*
`Ár:` *mennyiség * egységár*

Ne felejtse el számmá alakítani a mennyiséget és az egységárat!

Feladat 5.18. Nyomtassa ki az alábbi szorzótáblát `f-string`-ek segítségével:

```
    1  2  3  4  5  6  7  8  9
  ++++++++++++++++++++++++++
1:  1  2  3  4  5  6  7  8  9
2:  2  4  6  8 10 12 14 16 18
3:  3  6  9 12 15 18 21 24 27
4:  4  8 12 16 20 24 28 32 36
5:  5 10 15 20 25 30 35 40 45
6:  6 12 18 24 30 36 42 48 54
7:  7 14 21 28 35 42 49 56 63
8:  8 16 24 32 40 48 56 64 72
9:  9 18 27 36 45 54 63 72 81
```

Feladat 5.19. Tegyük fel, hogy egy focibajnokság épp aktuális állását csv file-ként kapjuk meg. A mezők: a klub neve, az általa lejátszott meccsek száma, és a szerzett pontok. Írjon egy függvényt, aminek egyetlen argumentuma a csv file neve, és ami a *vesztett* pontok száma (vagy a meccsenként elért átlagpontszám) szerinti sorrendben nyomtatja ki a klubokat (a file-ban található további két paraméterükkel együtt). (Minden megnyert meccs 3 pontot ér.) Természetesen a kevesebb pontot vesztők állnak előbb.

Ennek persze csak akkor van értelme, ha nem minden klubok játszott ugyanannyi meccset (ami gyakran megtörténik). Ilyenkor viszont jobb képet ad a bajnokság állásáról, mint a szokásos, az elért pontok szerinti rendezés.

Például, ha így fest a csv file:

```
>>> import os
>>> print(os.popen("cat pl.csv").read())
Man City,23,57
Liverpool,22,48
Chelsea,24,47
Man Utd,22,38
West Ham,23,37
Arsenal,21,36
Tottenham,20,36
```

```
Wolverhampton,21,34
Brighton,22,30
Leicester,20,26
Aston Villa,21,26
Southampton,22,25
Crystal Palace,22,24
Brentford,23,23
Leeds,21,22
Everton,20,19
Norwich,22,16
Newcastle,21,15
Watford,20,14
Burnley,18,12
```

akkor a függvény ezt nyomtassa ki:

```
>>> by_lost_points("pl.csv")
   Club           GP Pts -Pts
 1 Man City       23 57  12
 2 Liverpool      22 48  18
 3 Tottenham      20 36  24
 4 Chelsea        24 47  25
 5 Arsenal        21 36  27
 6 Man Utd        22 38  28
 7 Wolverhampton  21 34  29
 8 West Ham       23 37  32
 9 Leicester      20 26  34
10 Brighton       22 30  36
11 Aston Villa    21 26  37
12 Southampton    22 25  41
13 Leeds          21 22  41
14 Everton        20 19  41
15 Crystal Palace 22 24  42
16 Burnley        18 12  42
17 Brentford      23 23  46
18 Watford        20 14  46
19 Newcastle      21 15  48
20 Norwich        22 16  50
```

Használjon `f-string`-eket!

Feladat 5.20. Írjon egy `is_palindrome()` függvényt, ami `True`-t ad vissza, ha az argumentuma, ami egy sorozat (lista, tuple, vagy string) palindróma, azaz egyenlő a megfordítottjával, és `False`-t egyébként. Ne fordítsa meg a sorozatot!

Feladat 5.21. Írjon egy `palindromes()` függvényt, amelynek egyetlen argumentuma egy olyan szövegfile (mint pl. `/usr/share/dict/words`) neve lesz, amelynek minden sorában egyetlen szó áll. A függvény nyomtassa ki a file-ban talált palindrómákat, és térjen vissza a számukkal.

5.1. `dict`.

Feladat 5.22. Írjon egy kétargumentumú `substitute()` függvényt, aminek első argumentuma egy string, a második egy szótár lesz, amelyben a kulcsok karekterek (1

hosszú stringek), az értékek pedig stringek. Olyan új stringet kell visszaadnia, ami az első argumentumának másolata, kivéve, hogy a szótárban kulcsként előforduló karakterek a megfelelő értékekre cserélődnek. Például:

```
>>> substitute("acbcade",{'a':'xyz','c':'zyx'})
'xyzzyxbzyxxyzde'
>>> substitute("acbcade",{'a':'c','c':'a'})
'cabacde'
```

El lehetne-e érni ugyanezt a `str` osztály `.replace()` metódusának ismételt alkalmazásával, különös tekintettel a fenti második példara? `s.replace(old,new)` `s` egy olyan másolatát adja vissza, amelyben `old` minden `s`-beli előfordulása `new`-ra van cserélve. (`old` és `new` stringek).

Feladat 5.23. Módosítsa az előző feladatbeli `substitute()` függvényt úgy, hogy ha a szótárban valamely karakterhez a `None` érték tartozik, akkor az törlődjön, azaz az üres stringre (`''`)) cserélődjön. Például:

```
>>> substitute("acbcade",{'a':'c','c': None})
'cbcde'
```

Feladat 5.24. Írjon egy `to_dict()` függvényt, amely párok (tuple-k) listájából olyan szótárt állít elő és ad vissza, amelynek kulcsai a párok első tagjai, értékei pedig a megfelelő második tagok. Például:

```
>>> to_dict([('egy',1), ('kettő',2), ('három',3)])
{'egy': 1, 'kettő': 2, 'három': 3}
```

Feladat 5.25. Írjon egy `word_count()` függvényt, amelynek egyetlen argumentuma egy szövegfile neve lesz, és ami a file-ban található szavak számát adja vissza. Ha már működik, csináljon belőle önálló, a parancssorból indítható programot.

```
python word_count.py szoveg.txt
```

ugyanazt az eredményt kell kinyomtassa, mint

```
wc -w text.txt
```

Probálja ki a programját hosszabb szövegfile-on, pl. ezen is.

Feladat 5.26. Írjon egy `top_freq()` nevű kétargumentumú függvényt, melynek első argumentuma egy szövegfile neve lesz, a második pedig egy n természetes szám. A függvény egy `(szó, gyakoriság)` párokból álló listát adjon vissza, amelyben az n leggyakrabban előforduló szó szerepel előfordulásainak számával.

A különféleképpen ragozott szavak különbözőknek számítanak, de kis- és nagybetűt ne különböztessen meg.

Ha rövid szövegfile-okon már működik, próbálja ki a függvényét ezen is. Erre

```
[('a', 10331), ('-', 3916), ('az', 3653), ('hogy', 2033), ('nem', 2003)]
```

az eredmény.

Segítség: használjon szótárat a szavak gyakoriságának tárolására. Használja a szótárakon működő `items()` metódust, amely `(kulcs, érték)` párokból álló listává konvertálja őket. Egy ilyen lista rendezésére, ha a párok második tagja szerint szeretne rendezni, használja a `sorted()` függvényt, így:

```
sorted(list_of_tuples, key=lambda x : x[1])
```

vagy így:

```
sorted(list_of_tuples, key=lambda x : x[1],reverse=True)
```

ha csökkenő sorrendbe szeretne rendezni.

Feladat 5.27. Az 5.24 és 5.26-beli függvények felhasználásával írjon egy `top_freq_dict()` függvényt, ami pontosan ugyanazt csinálja, mint `top_freq()`, de az eredményt szótár formájában adja vissza. Például erre ezt

```
{'a': 10331, '-': 3916, 'az': 3653, 'hogy': 2033, 'nem': 2003}
```

a szótárt.

Segítség: ha két sornál hosszabb a függvénye, akkor valamit feltehetőleg félreértett.

Feladat 5.28. Pythonban ugyan létezik `set` adattípus, de csináljunk egy újat! Reprezentáljon egy halmazt egy olyan szótár, melynek kulcsai a halmaz elemei, és minden értéke `None`. Akkor az úniót pl. így lehet definiálni:

```
>>> def union(s1,s2):
...     return dict.fromkeys(list(s1.keys()) + list(s2.keys()))
...
>>> union({1: None, 2: None}, {1: None, 3: None})
{1: None, 2: None, 3: None}
```

Itt a `dict.fromkeys()` ugyanazt teszi, mint `s1.fromkeys()` vagy `s2.fromkeys()` tenné, de mivel az eredményének (ami egy új szótár, melynek kulcsai az argumentumaként megadott iterálható objektumból (esetünkben listából) jönnek és aminek minden értéke `None`) semmi köze sem `s1`, sem `s2` értékéhez, írhatunk `dict.fromkeys()`. (Az ilyeneket *osztály metódusok*nak hívják, szemben az eddig látott *instancia metódusok*kal.)

A fenti `union()` mintájára definiálja az `intersection()` (metszet) és `difference()` (különbség) függvényeket, egy `set_add()` függvényt, ami az első (halmaz) argumentumába beteszi a második argumentumát, és egy `set_remove()` függvényt, ami az első (halmaz) argumentumából kiveszi a második argumentumát (ha benne van). Például:

```
>>> s1 = empty_set() ; set_add(s1,1) ; set_add(s1,2) ; s1
{1: None, 2: None}
>>> s2 = empty_set() ; set_add(s2,2) ; set_add(s2,3) ; s2
{2: None, 3: None}
>>> 1 in union(s1,s2)
True
>>> 1 in intersection(s1,s2)
False
>>> 1 in difference(s1,s2)
True
```

Szerencsére az `in` operátor magától úgy működik, ahogy azt várjuk tőle, mert ha *d* egy szótár, akkor `k in d` akkor ad vissza `True`-t, ha `k` kulcs `d`-ben.

Kivételkezelés.

Feladat 5.29. Csinálja meg újra az 5.14 feladatot, de most használja a beépített `.index()` metódust. Az ebben a kihívás, hogy `string.index(substring)` `ValueError` kivételt dob ha `substring` nem része `string`-nek, de `lindex()` ebben az esetben símán -1-et kell visszaadjon. Más kivételeket viszont nem rejthet el, pl. azzal, hogy -1-et (vagy bármilyen más értéket) ad vissza. Például:

```
>>> lindex("A mai vacsora egy csoda", "cso")
8
```

```
>>> lindex("A mai vacsora egy csoda", "csom")
-1
>>> lindex("A mai vacsora egy csoda", 42) #hibát kell dobjon
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in lindex
TypeError: must be str, not int
```

Feladat 5.30. Módosítsa az előző feladatra adott megoldását úgy, hogy ha `.index()` hívása `ValueError`-tól különböző kivételt generál, akkor `lindex()` kinyomtatja az "Ismeretlen hiba" üzenetet mielőtt a kivételt (épp úgy, mint az előző vátozatban) továbbpasszolná a hívónak. Például:

```
>>> lindex("A mai vacsora egy csoda", "cso")
8
>>> lindex("A mai vacsora egy csoda", "csom")
-1
>>> #Eddig minden úgy működik, mint az előbb.
>>> #Most viszont lesz egy apró különbség (a mi hibaüzenetünk).
>>> #De fontos, ez legyen az egyetlen különbség.
>>> lindex("A mai vacsora egy csoda", 42) #hibát kell dobjon
Ismeretlen hiba
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in lindex
TypeError: must be str, not int
```

5.2. **Listagenerálás.**

Feladat 5.31. Írjon egy `even_odd()` függvényt, amelynek két argumentuma két egész számokat tartalmazó lista, és amely egy olyan listát ad vissza, melyben az első argumentumbeli páros számokat a második argumentumbeli páratlanok követik. Használjon listagenerálást!

Feladat 5.32. Listagenerálást használva írjon egy `same_mod()` nevű függvényt, amely két természetes számokból álló listával és egy pozitív természetes számmmal hívható meg, és azoknak a pároknak a listáját adja vissza, amelyeknek első (második) koordinátájuk az első (második) lista tagja, és egyenlők modulo a harmadik argumentum. Például:

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

Feladat 5.33. Módosítsa az előző feladatra adott megoldását úgy, hogy a `same_mod()` által visszaadott lista ne csak az ottani feltételt elégítse ki, de az is igaz legyen rá, hogy minden pár első koordinátája nem osztható a harmadik argumentummal. Például:

```
>>> same_mod(list(range(4)), list(range(2,8)),3)
[(1, 4), (1, 7), (2, 2), (2, 5)]
```

Feladat 5.34. Írjon egy kétargumentumú `squares()` függvényt, melynek első argumentuma egy egészekből álló lista, a második egy pozitív természetes szám. `squares(lst,n)` listák listáját adja vissza; ebben az i. lista n egymás utáni négyzetszámot tartalmaz, melyek közül `lst` i. eleme az első. Például:

```
>>> squares(list(range(1,8,3)),3)
[[1, 4, 9], [16, 25, 36], [49, 64, 81]]
```

Use list comprehension(s)!

Feladat 5.35. Írjon egy listagenerálást használó `concatenate()` nevű függvényt, amelynek argumentuma listák egy listája lesz, és ezek egymás után fűzöttjét adja vissza. Például:

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

Feladat 5.36. Ugyanaz a feladat, mint 5.9-ban, de most használjon listagenerálást!
Segítség: használja a `zip()` függvényt!

Feladat 5.37.* Írjon egy `merge()` függvény, ami két rendezett listát fésül össze. Például

```
>>> merge([2,7,9], [1,5,6])
[1, 2, 5, 6, 7, 9]
```

Segítség: próbálja meg rekurzív függvénnyel, és gondolja meg, milyen esetek lehetnek!

6. Függvények

Feladat 6.1. Mi a `mylist` változó értéke az alábbi kód végrehajtása után, és miért?

```
def side_effect_or_not (l):
    l = [2*i for i in l]
    return l

mylist = [1,2,3] ; side_effect_or_not(mylist)
```

Feladat 6.2. Mi a `mylist` változó értéke az alábbi kód végrehajtása után, és miért?

```
def side_effect_or_not (l):
    l.append(len(l))
    return l

mylist = [1,2,3] ; side_effect_or_not(mylist)
```

Feladat 6.3. Ebben és a következő feladatban hívjunk fának egy objektumot ha szám, vagy ha fák listája. Tehát például `0`, `1` és `2` fák, mert számok, és így `[0, 1, 2]` is fa, mert fák listája. Ugyanezért

```
[0, 1, [0, 1, 2], 2]
```

is fa, és

```
[0, [0, 1, 2], [0, [0, 1, 2], 1, [0, 1, 2], 2], 2]
```

is az.

Írjon egy `sumtree()` nevű függvényt, amely egy fában szereplő számok összegét adja vissza. Például a fenti utolsó fára 14-et.

Feladat 6.4.* Írjon egy `flatten()` nevű függvényt, amely egy fában előforduló számok listáját adja vissza, mégpedig a "mélységi bejárás" szerinti sorrendben. Ez azt jelenti, hogy — ha a fa lista — az elejétől vizsgáljuk az elemeit, és ha listával találkozunk, akkor először azt vizsgáljuk, mielőtt az épp vizsgált lista további elemeire térnénk rá. Ha a fa szám, akkor az őt tartalmazó lista a visszatérési érték. Például:

```
>>> flatten(1)
[1]
>>> flatten([1])
[1]
```

```
>>> flatten([1, 2])
[1, 2]
>>> flatten([1, 2, [3, 4]])
[1, 2, 3, 4]
>>> flatten([0, [0, 1, 2], [0, [0, 1, 2], 1, [0, 1, 2], 2], 2])
[0, 0, 1, 2, 0, 0, 1, 2, 1, 0, 1, 2, 2, 2]
>>> flatten([[0, 1, 2], -1, [-2, [3, 4], [5, 6, 7, 8],
...     [9, 10, 11]], [13, 14, 15], [16, 17]])
[0, 1, 2, -1, -2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15, 16, 17]
```

Feladat 6.5.* Írjon egy `sublists()` nevű függvényt, amely az argumentumaként adott lista összes részlistájának listáját adja vissza (tetszőleges sorrendben). (Itt most `l1` részlistája `l2`-nek, ha kalkulus értelemben részsorozata, azaz ha `l1` minden tagja szerepel `l2`-ben, mégpedig ugyanabban a sorrendben.) Például:

```
>>> sublists([])
[[]]
>>> sublists([1])
[[1], []]
>>> sublists([1,2])
[[1, 2], [1], [2], []]
>>> sublists([1,2,3])
[[1, 2, 3], [1, 2], [1, 3], [1], [2, 3], [2], [3], []]
```

Feladat 6.6. Csinálja meg újra a 3.18 feladatot, de most úgy, hogy `lookup()`-nak megadható legyen egy `default` nevű "keyword only" argumentum, amit akkor ad vissza, ha nem találja a kulcsot. Ha nem adunk meg ilyen argumentumot, akkor továbbra is `None`-t adjon vissza ebben az esetben.

Feladat 6.7. Módosítsa a 5.35-beli `concatenate()` függvényt úgy, hogy listák listája helyett tetszőleges számú listát fogadjon el argumentumként. Például:

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

Segítség: egyetlen karaktertől eltekintve az új definíció megegyezhet a régivel.

Feladat 6.8. Írjon egy akárhány (lista vagy string) argumentummal hívható `myzip()` függvényt, amely egy tuple-kből álló listát ad vissza, aminek hossza a legrövidebb argumentum hossza. A lista j-edik tagjának i-edik tagja az i-edik argumentum j-edik tagja legyen. Például:

```
>>> myzip('abcdefg', list(range(3)), list(range(10,14)))
[('a', 0, 10), ('b', 1, 11), ('c', 2, 12)]
```

Ne használja a `zip()` függvényt!

Segítség: induljon ki 5.11 megoldásából!

Feladat 6.9. Mint a 5.8 feladatban, írjon egy `transpose()` függvényt, ami egy listák listájaként reprezentált kétdimenziós (nem feltétlenül négyzetes) mátrix transzponáltját adja vissza, de most úgy, hogy a transzponáltat listagenerálással számítja ki. Például:

```
>>> transpose([[1,2],[3,4],[5,6]])
[[1, 3, 5], [2, 4, 6]]
```

Segítség: használja a `*`-ot és a `zip()` függvényt!

Feladat 6.10.* Csinálja meg újra az 5.10 feladatot, de most csak listagenerálást használva. Az egyszerűség kedvéért használhatja a `transpose()` függvényt.

Feladat 6.11. Definiáljon egy kétargumentumú `apply()` függvényt, ami az első argumentumát (egy egyargumentumú függvény) alkalmazza a második argumentumára, és az eredményt adja vissza.

Feladat 6.12. Definiáljon egy kétargumentumú `self_compose` függvényt úgy, hogy ha `fun` egyargumentumú függvény és `n` természetes szám, akkor

```
self_compose(fun, n)
```

`fun` `n`. hatványát (azaz `n`-szeres kompozícióját saját magával) adja vissza. Speciálisan, `self_compose(fun, 1)` magát `fun`-t, `self_compose(fun, 2)` pedig `fun` kompozícióját magával. Mit kellene visszaadnia `self_compose(fun, 0)`-nak? Ha nem tudja, tegye fel (és ellenőrizze `assert`-tel), hogy a második argumentum mindig pozitív.

Például:

```
>>> (self_compose(lambda x: x+1,3))(0)
3
```

Feladat 6.13. Írjon egy kétargumentumú `compose12()` nevű függvényt, amely a következőt tudja: ha `f1` egyargumentumú, `f2` pedig kétargumentumú függvény, akkor `compose12(f1,f2)` az `f1(f2(_,_))` kétargumentumú függvényt adja vissza. Például:

```
>>> compose12(lambda x: 2*x,lambda x,y: x+y)(2,3)
10
```

Feladat 6.14. Írjon egy kétargumentumú `compose1n()` nevű függvényt, amely a következőt tudja: ha `f1` egyargumentumú, `f2` pedig n argumentumú függvény, akkor `compose1n(f1,f2)` az `f1(f2(_,_, ... , _))` n argumentumú függvényt adja vissza. Például:

```
>>> compose1n(lambda x: 2*x,lambda x,y: x+y)(2,3)
10
>>> compose1n(lambda x: 2*x,lambda x,y,z: z*(x+y))(2,3,4)
40
```