

Ajánlott függvények a 'lists' könyvtárból

all/2, flatten/1, foldl/3, map/2, member/2, nth/2, seq/2, seq/3, sublist/3,  
unzip/1, usort/1, zip/2

```
-spec all(F::fun((X::term()) -> boolean()), Xs::[term()]) -> B::boolean().  
% B igaz, ha P igaz Xs minden elemére.
```

```
-type deeplist() :: [term() | deeplist()].  
-spec flatten(Ds::Deeplist()) -> Ls::[term()].  
% Ds beágyazott elemeinek kilapított listája Ls.
```

```
-spec foldl(F::fun((X::term(), E::term()) -> E1::term()),  
           E0::term(), Xs::[term()]) -> R::term().  
% Az E0-ból és az Xs listában balról jobbra haladva az Xs elemeiből  
% a kétoperandusú F-fel képzett érték R.
```

```
-spec map(F::fun((X::term()) -> Y::term()),  
         Xs::list(term())) -> Ys::list(term()).  
% Az Xs lista X elemeiből az F transzformációval képzett Y elemek listája Ys.
```

```
-spec member(X::term(), Xs::[term()]) -> B::boolean().  
% B igaz, ha X előfordul Xs-ben.
```

```
-spec nth(N::integer(), Xs::list(term())) -> X::term().  
% X az Xs N-edik eleme.
```

```
-spec seq(F::integer(), T::integer()) -> Ns::[integer()].  
% Ns az F-től T-ig egyesével növekedő egész számok listája.
```

```
-spec seq(F::integer(), T::integer(), D::integer()) -> Ns::[integer()].  
% Ns D>0 esetén az F-től T-ig D értékkel növekedő, de T-nél nem nagyobb,  
% D<0 esetén az F-től T-ig D értékkel csökkenő, de T-nél nem kisebb  
% egész számok listája.
```

```
-spec sublist(Ls::list(), S::integer(), L::integer()) -> Rs::list().  
% Rs az Ls S-edik elemével kezdődő, legfeljebb N elemű lista.
```

```
-spec unzip(XYs::[{X::any(), Y::any()}]) -> P::{Xs::[any()], Ys::[any()]}.  
% A P pár első tagja, Xs az XYs listában lévő párok első tagjából (X),  
% második tagja, Ys az XYs listában lévő párok második tagjából (Y)  
% képzett lista.
```

```
-spec usort(Xs::[term()]) -> Rs::[term()].  
% Az Xs rendezett, azonos értékű elemeket nem tartalmazó változata Rs.
```

```
-spec zip(Xs::[X0::any()], Ys::[Y0::any()]) -> XYs::[X::any(), Y::any()].  
% Az XYs listában lévő párok első (X) és második (Y) tagja az egyforma  
% hosszú Xs és Ys lista azonos sorszámú X0 és Y0 eleme.
```

## LISTAKEZELÉS

### 1. Számsorozat jobbrekurzívan (vö. lists:seq/2)

```
-spec seq(F::integer(), T::integer()) -> S::[integer()].  
%% S = [F, F+1, ..., T].
```

```
seq(10,13) == [10,11,12,13].
```

### 2. Listaelemek különböző voltának vizsgálata

```
-spec all_different(Xs::[any()]) -> B::boolean().  
%% B igaz, ha az L listában csupa különböző értékű elem van.
```

A megoldást valósítsa meg többféleképpen:  
a) lists:member/2 felhasználásával (a rekurzív hívás  
lusta kiértékelésű legyen);  
b) lists:usort/1 felhasználásával.

```
all_different([1,2,3,1]) == false.  
all_different([1,2,3]) == true.
```

### 3. Listának legalább két eleme van -- hatékonyan, a length/1 függvény nélkül!

```
-spec van2eleme(L::list()) -> R::boolean().  
%% R == length(L) >= 2.
```

### 4. Lista részlistája listanézetrel, a lists:nth/2 és lists:seq/2 felhasználásával.

```
-spec sublist(L0::[any()], S::integer(), N::integer()) -> L::[any()].  
%% Az L0 lista S-edik elemétől kezdődő és N hosszú részlistája az L lista.
```

```
sublist([a,b,c], 2, 1) == [b].  
sublist([a,b,c], 2, 2) == [b,c].
```

### 5. Lista minden elemének ellenőrzése (vö. lists:all/2)

```
-spec all(Pred::fun((T::any()) -> boolean()), L::[T::any()]) -> B::boolean().  
%% B akkor és csak akkor igaz, ha Pred teljesül az L minden elemére.
```

A megoldást valósítsa meg többféleképpen. Melyik hatékonyabb és miért?  
a) lists:map/2 és lists:foldl/3 felhasználásával;  
b) lists:foldl/3 felhasználásával;  
c) rekurzívan, ügyelve a rekurzív hívás lusta kiértékelésére.

```
all(fun erlang:is_atom/1,[a,b]) and not all(fun erlang:is_atom/1,[a,1]).
```

### 6. Platók hossza

```
-spec platok_hossza(Xs::[any()]) -> PHs::[{P::any(), H::integer()}].  
%% Platónak nevezzük az egyenlő értékű elemek sorozatát egy listában.  
%% PHs olyan {P, H} párok listája, amelyekben P a platót képező  
%% érték, H pedig e plató hossza (= azonos értékű elemek száma).
```

```
platok_hossza([a,a,a,b,b,d,f,f,h]) == [{a,3},{b,2},{d,1},{f,2},{h,1}].
```

### 7. Lista ismétlődő elemei -- használja fel a lists:zip/2 és lists:sublist/3 függvényeket!

```
-spec duplak(Xs::[any()]) -> Ys::[any()].  
%% Ys az Xs azon elemeinek listája, amelyek az őket követő elemmel egyelők.
```

A feladathoz segítséget talál a feladatsor végén.

```
duplak([1,2,3]) == [].  
duplak([1,1,2,3,3,3]) == [1,3,3].
```

## 8. Mátrix részmatrixa

```
-spec koepe(M::list([any()])) -> M1::list([any()]).
%% M1 az M n*n-es négyzetes mátrix olyan (n/2)*(n/2) méretű részmatrixa,
%% mely az n/4+1. sor n/4+1. oszlopának elemétől kezdődik.
```

A megoldást valósítsa meg többféleképpen:

- Használja fel listanézetben a lists:sublist/3 függvényt:
- Használja fel listanézetben a lists:nth/2 függvényt.

```
M=[[a,b,e,f],
   [c,d,g,h],
   [i,j,m,n],
   [k,l,o,p]].
```

```
koepe(M) == [[d,g],[j,m]].
```

## 9. Mátrix középső elemei

```
-spec laposkoepe(M::lista([any()])) -> L::[any()].
%% Az L lista az M mátrix középső elemeinek listája (vö. 8. feladat).
```

A megoldást valósítsa meg többféleképpen:

- lists:flatten/1 és koepe/1 felhasználásával,  
Példa lists:flatten/1 használatára:  
lists:flatten([1,[2,3],[[4]],[5,[6]]) == [1,2,3,4,5,6].
- Használja fel listanézetben a lists:nth/2 függvényt.

```
laposkoepe(M) == [d,g,j,m].
```

## 10. Részmatrrix sor és oszlop elhagyásával

```
-spec pivot(M::list([any()]), R::integer(), C::integer())-> M1::list([any()]).
%% M1 az M mátrix R-edik sorának és C-edik oszlopának elhagyásával keletkezik.
```

A megoldáshoz használja fel listanézetben a lists:nth/2 függvényt.

```
pivot(M,2,3) == [[a,b,f],[i,j,n],[k,l,p]].
```

## 11. Mátrix transzponáltja

```
-spec transpose(M::list([any()])) -> MT::list([any()]).
%% M transzponáltja MT.
```

```
transpose([[a,b],[c,d],[e,f]]) == [[a,c,e],[b,d,f]].
```

## TOVÁBBI GYAKORLÓ FELADATOK OTTHONRA

### +1. Összetett számok

```
-spec osszetett(K::integer()) -> L::[integer()].
%% L tartalmazza az összetett számokat 4..K*K között, ismétlődés lehet.
```

"Eratoszthenész szitája": bejárjuk minden szám többszöröseit, és megjelöljük őket összetettként. Felhasználható a lists:seq/3 függvény:  
seq(F, T, D) == [F, F+D, F+2D, ..., F+KD], ahol F+KD =< T < F+(K+1)D.

```
osszetett(5) == [4,6,8,10,12,14,16,18,20,22,24,
                6,9,12,15,18,21,24,
                8,12,16,20,24,
                10,15,20,25].
```

### +2. Prímszámok

```
-spec primek(K::integer()) -> L::[integer()].
%% L tartalmazza a prímszámokat 2..K*K között.
```

```
primek(5) == [2,3,5,7,11,13,17,19,23].
```

### +3. Listák cipzárázása (vö. lists:zip/2, lists:unzip/1)

```
-spec zip(Xs::[any()], Ys::[any()]) -> XYs::[{any(), any()}].
-spec unzip(XYs::[{any(), any()}]) -> {Xs::[any()], Ys::[any()]}
```

%% XYs olyan párok listája, amelynek első eleme az Xs, második eleme az Ys lista azonos pozíciójú eleme.

```
zip([1,2,3], [a,b,c]) == [{1,a},{2,b},{3,c}].
unzip([1,a],[2,b],[3,c]) == [1,2,3], [a,b,c].
```

## SEGÍTSÉG A MEGOLDÁSHOZ

### 7. Lista ismétlődő elemei

A lista helyett tekintsük párok listáját. Cipzárazzuk össze a lista első, illetve utolsó elemének elhagyásával keletkező listákat, például az [1,1,2,3,3,3] esetén képezzük a [1,1,2,3,3], [1,2,3,3,3] listák cipzárásával keletkező listát: [{1,1},{1,2},{2,3},{3,3},{3,3}].

----- \$LastChangedDate: 2019-11-05 09:17:30 +0100 (k, 05 nov 2019) \$ -----