

Deklaratív programozás, 3. gyakorlat
Erlang programozás
2012. 10. 12.

Írjon olyan Erlang-függvényt, amely megfelel az adott fejkommentnek. A feladat megoldásához felhasználhat korábbi sorszámú feladatokban definiált eljárásokat. Néhány feladathoz segítséget talál a feladatsor végén.

Javasoljuk, hogy félidőben (az óra 45. percében) térjen át a 12. feladatra.

1. Számlista minden elemének növelése

```
%% @spec lista_noveltje(L0::[number()]) -> L::[number()].
%% Az L egészlista az L0 egészlistának olyan másolata, amelynek
%% ugyanannyi eleme van, mint az L0-nak, de minden eleme pontosan
%% eggyel nagyobb értékű, mint az L0 megfelelő eleme.
```

```
lista_noveltje([1,5,2]) == [2,6,3].
```

2. Lista utolsó elemének meghatározása (vö lists:last/1)

```
%% @spec last(Xs::[term()]) -> X::term().
%% X az Xs nemüres lista utolsó eleme.
```

```
last([5,1,2,8,7]) == 7.
```

3. Lista utolsó elemének meghatározása hibakezeléssel

```
%% @spec safe_last(Xs::[term()]) -> {ok, X::term()} | error.
%% Ha Xs üres, akkor 'error', különben X az Xs lista utolsó eleme.
```

```
safe_last([5,1,2,8,7]) == {ok,7}.
safe_last([]) == error.
```

Megjegyzés: az Erlangban gyakori, hogy a helyes eredményt egy kételemű ennesbe csomagolja a függvény, a hibát pedig egy atommal jelzi.
Példa: compile:file/1 (fordítás, shellből: c/1).

Mutassa be, hogyan használható a safe_last függvény (tipp a végén).

4. Lista kettévágása (vö lists:split/2)

```
%% @spec split(N::integer(), L::[term()]) -> {P::[term()], S::[term()]}.
%% P lista az L lista N hosszú prefixuma,
%% S lista az L lista (length(L) - N) hosszú szuffixuma.
```

```
split(3, [10,20,30,40,50]) == {[10,20,30],[40,50]}.
```

5. Lista adott hosszúságú prefixuma (vö lists:sublist/2)

```
%% @spec take(L0::[term()], N::integer()) -> L::[term()].
%% Az L lista az L0 lista N hosszú prefixuma.
```

```
take([10,20,30,40,50], 3) == [10,20,30].
```

6. Lista szuffixuma (vö lists:nthtail/2)

```
%% @spec drop(L0::[term()], N::integer()) -> L::[term()].
%% Az L0 lista olyan szuffixuma L, amely az L0 első B elemét
%% nem tartalmazza.
```

```
drop([10,20,30,40,50], 3) == [40,50].
```

7. Lista összes prefixumának listája

```
%% @spec prefixes(Xs::[term()]) -> Zss::[[term()]]
```

```
prefixes([a,b,c]) == [[], [a], [a,b], [a,b,c]].
```

8. Lista egyre rövidülő szuffixumainak listája

```
%% @spec tails(Xs::[term()]) -> Zss::[[term()]].
%% A Zss lista az Xs listát és egyre rövidülő szuffixumait tartalmazza.
```

```
tails([1,4,2]) == [[1,4,2],[4,2],[2],[]].
tails([a,b,c,d,e]) == [[a,b,c,d,e],[b,c,d,e],[c,d,e],[d,e],[e],[]].
```

9. Lista adott hosszúságú összes részlistáját tartalmazó lista

```
%% sublists(N::integer(), Xs::[term()]) ->
%%   [{B::integer(), Ps::[term()], A::integer()}].
%% Az Xs lista egy olyan (folytonos) részlistája az N hosszúságú Ps lista,
%% amely előtt B és amely után A számú elem áll Xs-ben.
```

```
sublists(1,[a,b,c]) == [{0,[a],2}, {1,[b],1}, {2,[c],0}].
sublists(2,[a,b,c]) == [{0,[a,b],1}, {1,[b,c],0}].
```

10. Lista összes nemüres részlistáját tartalmazó lista

```
%% sublists(Xs::[term()]) -> [{B::integer(), Ps::[term()], A::integer()}].
%% Az Xs lista egy olyan (folytonos), nemüres részlistája a
%% Ps lista, amely előtt B és amely után A számú elem áll Xs-ben.
```

```
sublists([a,b]) == [{0,[a],1}, {1,[b],0}, {0,[a,b],0}].
```

11. Listában párosával előforduló elemek listája

```
%% @spec parban(Xs::[term()]) -> Zs::[term()].
%% A Zs lista az Xs lista összes olyan elemét tartalmazza, amelyet
%% vele azonos értékű elem követ.
```

```
parban([a,a,a,2,3,3,a,2,b,b,4,4]) == [a,a,3,b,4].
```

12. Listák összefűzése a lists:foldr/3 függvénnyel

```
%% @spec append(Xs::[term()], Ys::[term()]) -> Zs::[term()].
%% A Zs lista az Xs és Ys összefűzésével áll elő (Zs == Xs ++ Ys).
```

```
append([a,b,c], [1,2,3]) == [a,b,c,1,2,3].
```

13. Egy lista megfordítása egy másik elé fűzve a lists:foldl/3 függvénnyel

```
%% @spec revapp(Xs::[term()], Ys::[term()]) -> Zs::[term()].
%% A Zs lista az Xs megfordítottjának az Ys elé fűzésével áll elő,
%% azaz Zs == lists:reverse(Xs)++Ys.
```

```
revapp([a,b,c], [1,2,3]) == [c,b,a,1,2,3].
```

14. Az 8. feladat (tails/1) újbóli megoldása a foldr/3 függvénnyel

15. A közismert map/2, filter/2 függvények megvalósítása listanézetrel

```
Paros = fun(X) -> X rem 2 == 0 end.
map(Paros, [1,2,3,4]) == [false,true,false,true].
```

```
filter(Paros, [1,2,3,4]) := [2,4].
```

```
-----  
16. Az 1. feladat (lista_noveltje/1) újbóli megoldása listanézettel  
-----
```

```
17. A 8. feladat (tails/1) újbóli megoldása listanézettel  
-----
```

```
18. A 9-10. feladatok (sublists/*) újbóli megoldása listanézettel  
-----
```

```
19. A 11. feladat (parban/1) újbóli megoldása listanézettel  
-----
```

```
20. Listában párosával előforduló részlisták listája
```

```
%% @spec dadogo(Xs::[term()]) -> Zss::[[term()]].  
%% A Zss lista az Xs lista összes olyan nemüres (folytonos) részlistáját  
%% tartalmazza, amelyet vele azonos értékű részlista követ.
```

```
dadogo([a,a,a,2,3,3,a,b,b,b]) := [[a],[a],[3],[b],[b,b],[b],[b]].
```

```
-----  
21. Lista első monoton növekvő részlistája (futama)
```

```
%% @spec rampa(Xs::[term()]) -> {Zs::[term()], Ms::[term()]}.  
%% A Zs lista az Xs lista első monoton növekvő futama, az Ms az Xs maradéka.
```

```
rampa([1,2,2,3,2,4,5,6,6,7,6,8,2,3,3,4,5,6,0,6,5,4,3,2,1]) :=  
{[1,2,2,3],[2,4,5,6,6,7,6,8,2,3,3,4,5,6,0,6,5,4,3,2,1]}.
```

SEGÍTSÉG A MEGOLDÁSHOZ

```
-----  
3. Lista utolsó elemének meghatározása hibakezeléssel  
Mutassa be, hogyan használható a safe_last függvény (tipp a végén).  
Például a case szerkezettel eldönthető, hogy történt-e hiba.
```

```
-----  
7. Lista összes nemüres prefixumának listája  
Javasolt segédfüggvény: lista legalább N hosszú prefixumainak listája.  
Hasznos BIF: length(L) az L lista hossza.
```

```
-----  
12. Listák összefűzése a lists:foldr/3 függvénnyel  
Érdeemes összevetni az előadáson szerepelt append/2 és a foldr/3  
függvények kódját.  
Javasolt segédfüggvény: a Céklából ismert cons/2.
```

```
-----  
13. Egy lista megfordítása egy másik elé fűzve a lists:foldl/3 függvénnyel  
Hasonlóan 12. feladathoz.
```

```
-----  
17. A 8. feladat (tails/1) újbóli megoldása listanézettel  
"Ciklus" szervezésére használja a lists:seq/2 függvényt.  
%% @spec lists:seq(N::integer(), M::integer()) -> S::[integer()].  
%% S := [N,N+1,...,M], pl. lists:seq(1, length("abc")) := [1,2,3].
```

```
-----  
21. Lista első monoton növekvő részlistája (futama)  
Alternatív megoldás: rampa/1 splitwith függvénnyel.  
%% splitwith(Pred, List) -> {takewhile(Pred, List), dropwhile(Pred, List)}.  
%% lists:splitwith(fun erlang:is_atom/1, [a,b,c,d,1,2,3,4,e,f,5,6,7]) :=  
%% { [a,b,c,d], [1,2,3,4,e,f,5,6,7] }.
```

TOVÁBBI GYAKORLÓ FELADATOK OTTHONRA

```
-----  
+1. Beszúrás listába adott helyre  
%% @spec insert_nth(Ls::[term()], E::term(), N::integer()) -> Rs::[term()].  
%% Az Rs lista az Ls lista olyan másolata, amelybe az Ls lista N-edik és
```

```
%% (N+1)-edik eleme közé be van szúrva az E elem (a lista számozása 1-től  
%% kezdődik).  
insert_nth([1,8,3,5], 6, 2) := [1,8,6,3,5].  
insert_nth([1,3,8,5], 3, 3) := [1,3,8,3,5].
```

```
+2. Beszúrás rendezett listába  
%% @spec insert_ord(S0s::[term()], E::term()) -> Ss::[term()].  
%% Az Ss szigorúan monoton növekvő egészlista az S0s szigorúan monoton  
%% növekvő egészlistának az E egészszel bővített változata, feltéve hogy  
%% E nem eleme az S0s listának; egyébként Ss == S0s.  
insert_ord([1,3,5,8], 6) := [1,3,5,6,8].  
insert_ord([1,3,5,8], 3) := [1,3,5,8].
```

```
+3. Adott lista adott sorszámú eleme (vö lists:nth)  
%% @spec nth(N::integer(), Ls::[term()]) -> E::term().  
%% Az Ls lista N-edik eleme E (1-től számozva az elemeket).  
nth(3, [a,b,c]) := c.
```

```
+4. Adott lista sorszámozott elemeiből álló lista (vö lists:zip, lists:seq)  
%% @spec zipseq(Ls::[term()]) -> {N::integer(), E::term()}.  
%% Az Ls lista N-edik eleme E (1-től számozva az elemeket).  
zipseq([a,b,c]) := {[1,a},{2,b},{3,c]}.
```

```
+5. Lista darabokra szabdalása  
%% @spec slash(F::fun([term()]) -> {term(),[term()]}, Xs::([term()]))  
%% -> Zs::([term()]).  
%% A Zs lista az Xs listából az F ismételt alkalmazásával előálló lista,  
%% ahol F párban visszaadja a Zs következő elemét és az Xs még nem  
%% feldolgozott részét.  
slash(fun(Xs) -> lists:split(3, Xs) end, "jaaaj!!! nem jooo!") ==  
["jaa","aj!","!!","nem","jo","oo!"].
```

```
+6. Lista monoton növekvő részlistáinak (futamainak) listája  
%% @spec rampak(Xs::[term()]) -> Xss::[[term()]].  
%% Az Xss az Xs monoton növekvő futamainak listája.  
rampak([1,2,2,3,2,4,5,6,7,6,8,2,3,3,4,5,6,0,6,5,4,3,2,1]) :=  
[[1,2,2,3],[2,4,5,6,7],[6,8],[2,3,3,4,5,6],[0,6],[5],[4],[3],[2],[1]].  
%% Alternatív megoldás: rampak/1 slash függvénnyel.
```

```
+7. Lista számtani sorozatot alkotó prefixuma  
%% @spec dif(Ls::[number()]) -> {Ds::[number()], Zs::[number()]}.  
%% A Ds lista az Ls lista számtani sorozatot alkotó prefixuma, a Zs az Ls  
%% maradéka (Zs utáni része).  
%% %% Segédfüggvény gyujtóargumentummal.
```

```
%% @spec dif(Ls::[number()],N::number(),Rs::[number()]) ->  
%% {Ds::[number()], Zs::[number()]}.  
%% A Ds lista az Ls lista N különbségű számtani sorozatot alkotó prefixuma  
%% az Rs lista fordítottja mögé fűzve, a Zs az Ls maradéka (Zs utáni része).  
dif([1,2,3,4,8,16,32,33,34]) == {[1,2,3,4],[8,16,32,33,34]}.  
dif([1,2,3,4,8,16,24,32,33,34]) == {[1,2,3,4],[8,16,24,32,33,34]}.
```

```
%% Alternatív megoldás: dif/1 feltétel helyett mintaillesztéssel.  
+8. Lista számtani sorozatot alkotó részlistáinak listája  
%% @spec difek(Xs::[number()]) -> Dss::[[number()]].  
%% A Dss lista az Xs lista számtani sorozatot alkotó részlistáinak listája.  
difek([1,2,3,4,8,16,32,33,34]) == [[1,2,3,4],[8,16],[32,33,34]].  
difek([1,2,3,4,8,16,24,32,33,34]) == [[1,2,3,4],[8,16,24,32],[33,34]].  
%% Alternatív megoldás: difek/1 slash függvénnyel.
```

```
%% Feladat: úgy módosítani dif/1-et, hogy ha egy szám az egyik  
%% számtani sorozat utolsó és egyben a következő első eleme is  
%% lehet, akkor mindkettőben vegyük figyelembe  
%% NB. Ilyenkor egy sorozat záróeleme mindenképpen egy következő  
%% sorozat kezdőeleme is lesz egyben, hiszen már két elem is  
%% számtani sorozatot alkot.
```

```
slash(fun dif1/1,[1,2,3,4,5,6,7,8,16,24,32,33,34]) ==  
[[1,2,3,4,5,6,7,8],[8,16,24,32],[32,33,34]].  
slash(fun dif1/1,[1,2,3,4,5,6,7,16,24,32,33,34]) ==  
[[1,2,3,4,5,6,7],[7,16],[16,24,32],[32,33,34]].
```

```
+9. Listák összefűzése (vö laposítás, lists:flatten/1)  
%% @spec flatten(Xss::[[term()]]) -> Xs::[term()].  
%% Xs lista elemei az Xss-ben lévő listák elemei egymás után fűzve.
```

```
+10. A 10. feladat (sublists/1) újbóli megoldása listanézettel,  
felhasználva a sublists/2 és flatten/1 függvényeket.
```

```
----- $LastChangedDate: 2012-10-09 18:52:19 +0200 (Tue, 09 Oct 2012) $ -----
```