```
-module('dp20a-gy1').
-compile(export_all).
-author('patai@emt.bme.hu, hanak@emt.bme.hu, kapolnai@iit.bme.hu').
-vsn('$LastChangedDate: 2020-09-21 07:15:39 +0200 (h, 21 szept 2020) $$').

% 1.
-spec lnko(A :: integer(), B :: integer()) -> D :: integer().
%% A és B legnagyobb közös osztója D.
lnko(A, 0) ->
    A;
lnko(A, B) ->
    lnko(B, A rem B).

% 2.
-spec pi(I :: integer()) -> Pi :: float().
%% A pi, azaz a Ludolph-féle szám I-edik közelítő értéke Pi.
%% A Leibniz-féle sor: pi/4 = 1 - 1/3 + 1/5 - 1/7 + ...
pi(I) ->
    4 * pi_4(I*2+1, 1, p, 0).

pi_4(N, K, p, Pi) when K =< N ->
    pi_4(N, K+2, m, Pi+(1/K));
pi_4(N, K, m, Pi) when K =< N ->
    pi_4(N, K+2, p, Pi-(1/K));
pi_4(N, K, _E, Pi) when K > N ->
    Pi.

% 3.
-spec dec2rad(R :: integer(), I :: integer()) -> Ds :: [integer()].
%% Az I egész szám R alapú számrendszerbe konvertált, decimális számként
%% megadott számjegyeinek listája Ds.
dec2rad(R, I) ->
    dec2rad(R, I, []).

dec2rad(_R, 0, Ds) ->
    Ds;
dec2rad(R, I, Ds) ->
    dec2rad(R, I div R, [I rem R | Ds]).

% 4.
-spec conv(R :: integer(), I :: integer()) -> {ok, Ds :: string()} | error.
%% Az I egész szám R alapú számrendszerbe konvertált jegyeinek listája Ds,
%% ha 2 <= R <=16, egyébként error.
conv(R, I) when R >= 2 andalso R =< 16 ->
    {ok, [ lists:nth(X+1,"0123456789abcdef") || X <- dec2rad(R, I) ]};
conv(_R, _I) ->
    error.

% 5.
-spec safe_last(Xs::[any()]) -> {ok, X::any()} | error.
%% Ha Xs üres, akkor 'error', különben X az Xs lista utolsó eleme.
safe_last([_|_] = Xs) -> {ok, lists:last(Xs)};
safe_last(_)          -> error.

% 6.
-spec nth(N::integer(), L::[any()]) -> E::any().
%% Az L lista N-edik eleme E (1-től számozva az elemeket).
nth(1, [H|_]) -> H;
nth(N, [_|T]) -> nth(N-1, T).

% 7.
-spec split(N::integer(), L::[any()]) -> {P::[any()], S::[any()]}.
%% A P lista az L lista N hosszú prefixuma (első N eleme), az S lista az
%% L lista (length(L) - N) hosszú szuffixuma (első N eleme utáni része).
split(0, L) ->
    {[], L};
split(N, [H|T]) ->
    {PT,S} = split(N-1, T),
    {[H|PT], S}.
```

```
% 8.
-spec take(L0::[any()], N::integer()) -> L::[any()].
%% Az L0 lista N hosszú prefixuma az L lista.
take(L0, N) ->
    {L1,_L2} = split(N, L0),
    L1.

% vagy mintaillesztés nélkül BIF-fel:
-spec take_2(L0::[any()], N::integer()) -> L::[any()].
take_2(L0, N) ->
    element(1, split(N, L0)).

% 9.
-spec drop(L0::[any()], N::integer()) -> L::[any()].
%% Az L0 lista első N elemét nem tartalmazó szuffixuma az L lista.
drop(L0, N) ->
    {_L1,L2} = split(N, L0),
    L2.

% vagy mintaillesztés nélkül BIF-fel:
-spec drop_2(L0::[any()], N::integer()) -> L::[any()].
drop_2(L0, N) ->
    element(2, split(N, L0)).

% 10.
-spec tails(Xs::[any()]) -> Zss::[[any()]].
%% Az Xs lista egyre rövidülő szuffixumainak listája a Zss lista.
tails([_H|T]=Xs) ->  [Xs|tails(T)];
tails([]) ->         [[]].

% 11.
-spec prefixes(Xs::[any()]) -> Zss::[[any()]].
%% Az Xs lista egyre hosszabbodó prefixumainak listája a Zss lista.
prefixes(Xs) ->
    prefixes(Xs, 0).

-spec prefixes(Xs::[any()], N::integer()) -> Zss::[[any()]].
% Az Xs lista legalább N hosszú prefixumainak listája a Zss lista.
prefixes(Xs, N) ->
    case length(Xs) >= N of
        true  -> [take(Xs, N)|prefixes(Xs, N+1)];
        false -> []
    end.

% vagy őrrel:
-spec prefixes_2(Xs::[any()]) -> Zss::[[any()]].
%% Az Xs lista egyre hosszabbodó prefixumainak listája a Zss lista.
prefixes_2(Xs) ->
    prefixes_2(Xs, 0).

-spec prefixes_2(Xs::[any()], N::integer()) -> Zss::[[any()]].
% Az Xs lista legalább N hosszú prefixumainak listája a Zss lista.
prefixes_2(Xs, N) when length(Xs) >= N ->
    [take(Xs, N)|prefixes_2(Xs, N+1)];
prefixes_2(_Xs, _N) ->
    [].
```

```
% 12.
-spec sublists(N::integer(), Xs::[any()]) ->
                      [{B::integer(), Ps::[any()], A::integer()}].
%% Az Xs lista egy olyan (folytonos) részlistája az N hosszú Ps lista,
%% amely előtt B és amely után A számú elem áll Xs-ben.
sublists(N, Xs) ->
    sublists(N, 0, Xs).

-spec sublists(N::integer(), B0::integer(), Xs::[any()]) ->
                      [{B::integer(), Ps::[any()], A::integer()}].
%% Az Xs lista egy olyan (folytonos) részlistája az N hosszú Ps lista,
%% amely előtt (B-B0) és amely után A számú elem áll Xs-ben.
sublists(N, B0, Xs) ->
    case length(Xs) < N of
        true  -> [];
        false ->
            [{B0, take(Xs, N), length(Xs) - N} | sublists(N, B0+1, tl(Xs))]
    end.

% 13.
-spec parban(Xs::[any()]) -> Zs::[any()].
%% A Zs lista az Xs lista összes olyan elemének listája, amelyet
%% vele azonos értékű elem követ.
parban([E,E|T])    -> [E|parban([E|T])];
parban([_E1,E2|T]) -> parban([E2|T]);
parban(_)          -> [].

% 14.
-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([], _, _) -> [];
sublist([H|_], 1, 1) -> [H];
sublist([H|T], 1, N) -> [H|sublist(T, 1, N-1)];
sublist([_|T], S, N) -> sublist(T, S-1, N).

sublist_2(L, S, N) ->
    take(drop(L, S-1), N).

sublist_3(L, S, N) ->
    [lists:nth(I, L) || I <- lists:seq(S, S+N-1)].

% 15.
-spec vertekek(Xs::[any()]) -> Es::[any()].
%% Az Xs lista elemei közül a {v::atom(),E:any()} mintára illeszkedő
%% párok második tagjából képzett lista az Es lista.
vertekek(L) ->
    [E || {v,E} <- L].

% vagy listanézet nélkül:
-spec vertekek_2(Xs::[any()]) -> Es::[any()].
vertekek_2(L) ->
    lists:map(fun({v,E}) -> E end,
        lists:filter(fun({v,_}) -> true ; (_) -> false end, L)).

% vagy mintaillesztés nélkül:
-spec vertekek_3(Xs::[any()]) -> Es::[any()].
vertekek_3(L) ->
    lists:map(fun({v,E}) -> E end,
        lists:filter(fun(X) -> is_tuple(X) andalso size(X) =:= 2 andalso
                                  element(1, X) =:= v end, L)).

% vagy a lista "darálásával"
-spec vertekek_4(Xs::[any()]) -> Es::[any()].
vertekek_4([{v,E}|T]) ->
    [E|vertekek_4(T)];
vertekek_4([_|T]) ->
    vertekek_4(T);
vertekek_4([]) ->
    [].
```

```
% 16.
-spec cons(X::any(), Xs::[any()]) -> Zs::[any()].
%% Az X elem Xs elé füzésének az eredménye a Zs lista.
cons(X, Xs) ->
    [X|Xs].

-spec append(Xs::[any()], Ys::[any()]) -> Zs::[any()].
%% Az Xs lista Ys elé fűzésének eredménye a Zs lista (Zs =:= Xs ++ Ys).
append(Xs, Ys) ->
    lists:foldr(fun cons/2, Ys, Xs).

% 17.
-spec revapp(Xs::[any()], Ys::[any()]) -> Zs::[any()].
%% A megfordított Xs lista Ys elé fűzésének eredménye a Zs lista,
%% azaz Zs =:= lists:reverse(Xs)++Ys.
revapp(Xs, Ys) ->
    lists:foldl(fun cons/2, Ys, Xs).

% 18.
-spec tails_2(Xs::[any()]) -> Zss::[[any()]].
%% Az Xs lista egyre rövidülő szuffixumainak listája a Zss lista.
tails_2(Xs) ->
lists:foldr(fun(Y, [Zs|Zss]) -> [[Y|Zs],Zs|Zss] end, [[]], Xs).

-spec tails_3(Xs::[any()]) -> Zss::[[any()]].
%% Az Xs lista egyre rövidülő szuffixumainak listája a Zss lista.
tails_3(Xs) ->
    [drop(Xs, N) || N <- lists:seq(0, length(Xs))].

% 19.
-spec sublists_2(N::integer(), Xs::[any()]) ->
                      [{B::integer(), Ps::[any()], A::integer()}].
%% Az Xs lista egy olyan (folytonos) részlistája az N hosszú Ps lista,
%% amely előtt B és amely után A számú elem áll Xs-ben.
sublists_2(N, Xs) ->
    [{B,take(drop(Xs, B), N),length(Xs)-B-N} ||
        B <- lists:seq(0, length(Xs)-N)].

% 20.
-spec parban_2(Xs::[any()]) -> Zs::[any()].
%% A Zs lista az Xs lista összes olyan elemének listája, amelyet
%% vele azonos értékű elem követ.
parban_2(Xs) ->
    [E || [E,E|_] <- tails(Xs)].

-spec parban_3(Xs::[any()]) -> Zs::[any()].
%% A Zs lista az Xs lista összes olyan elemének listája, amelyet
%% vele azonos értékű elem követ.
parban_3(Xs) ->
    [E || {E,E} <- lists:zip(tl(Xs), take(Xs, length(Xs)-1))].

% 21.
-spec dadogo(Xs::[any()]) -> Zss::[[any()]].
%% A Zss lista az Xs lista összes olyan nemüres (folytonos) részlistájából
%% álló lista, amelyet vele azonos értékű részlista követ.
dadogo(Xs) ->
    [P || T <- tails(Xs),
          N <- lists:seq(1, length(T) div 2),
          begin {P,S} = split(N, T), P =:= take(S, N) end
    ].
```

```
% 22.
-spec flatten(DeepList::list()) -> L::list(). %% list() =:= [any()].
%% A tetszőleges mélységű beágyazott listákból álló DeepList lista
%% elemeiből álló, listákat már nem tartalmazó lista az L lista.
flatten([]) ->
    [];
flatten([H|T]) when is_list(H) ->
    flatten(H) ++ flatten(T);
flatten([H|T]) -> % when not is_list(H)
    [H|flatten(T)].

-spec flatten_2(DeepList::list()) -> L::list(). %% list() =:= [any()].
%% A tetszőleges mélységű beágyazott listákból álló DeepList lista
%% elemeiből álló, listákat már nem tartalmazó lista az L lista.
flatten_2(DL) -> flatten_2(DL, []).

-spec flatten_2(DeepList::list(), Tail::list()) -> L::list().
% flatten_2(DL, Tail) = Az L lista a kilapított s a Tail elé fűzött DL lista.
flatten_2([H|T], Tail) when is_list(H) ->
    flatten_2(H, flatten_2(T, Tail));
flatten_2([H|T], Tail) ->
    [H|flatten_2(T, Tail)];
flatten_2([], Tail) ->
    Tail.

% Gyakorlo

-spec slash(F::fun((list()) -> {any(),list()}), Xs::(list())) ->
                  Zs::(list()).
%% A Zs lista az Xs listából az F ismételt alkalmazásával előálló lista, ahol
%% F eredménye egy pár: Zs következő eleme és Xs még fel nem dolgozott része.
slash(_F, []) ->
    [];
slash(F, Xs) ->
    {Z, Rs} = F(Xs),
    lists:append([Z], slash(F, Rs)).

-spec slash_2(F::fun((list()) -> {any(),list()}), Xs::(list()))
             -> Zs::(list()).
%% A Zs lista az Xs listából az F ismételt alkalmazásával előálló lista, ahol
%% F eredménye egy pár: Zs következő eleme és Xs még fel nem dolgozott része.
slash_2(F, Xs) ->
    slash_2(F, Xs, []).

slash_2(_F, [], Zs) ->
    lists:reverse(Zs);
slash_2(F, Xs, Zs) ->
    {Z, Rs} = F(Xs),
    slash_2(F, Rs, [Z|Zs]).

rampa_1(L) ->
    case L of
        [X1,X2|Xs] -> case X1 =< X2 of
                          true  -> {Zs,Ms} = rampa_1([X2|Xs]),
                                   {[X1|Zs], Ms};
                          false -> {[X1], [X2|Xs]}
                      end;
        L          -> {L, []}  % vagyis ha length(L) < 2
    end.

% vagy őrrel:
rampa_1b([]) ->  % ha ures listaval hivjak meg
    {[], []};
rampa_1b([X1,X2|Xs]) when X1 =< X2 ->
    {Zs,Ms} = rampa_1b([X2|Xs]),
    {[X1|Zs], Ms};
rampa_1b([X|Xs]) ->
    {[X], Xs}.
```

```
% lásd http://www.erlang.org/doc/man/lists.html
rampa_2([]) ->
    {[],[]};
rampa_2(Ls) ->
    Rs = lists:takewhile(fun({X,Y}) -> X =< Y end,
                         lists:zip(lists:sublist(Ls, length(Ls)-1), tl(Ls))),
    lists:split(length(Rs)+1, Ls).

%% Alternatív megoldás tails/1 és lists:splitwith/2 használatával.
%%
%% rampa_3/1-ben tails/1 eredménye az Ls egyre rövidülo farkainak a listája,
%% pl.
%% tails([a,b,c,b,a]) =:= [a,b,c,b,a]
%%                          [b,c,b,a]
%%                            [c,b,a]
%%                              [b,a]
%%                                [a]
%%                                 []
%% splitwith/2 közülük azokat adja vissza Rss-ben, amelyeknek a feje nem
%% nagyobb a második elemüknél, Mss-ben pedig - az elso kivételével - azokat,
%% amelyekre ez nem áll fenn, pl.
%%    Rss =:= [[a,b,c,b,a],[b,c,b,a]]
%%    Mss =:= [[b,a],[a],[]]
%%
%% Ha ezek után az Rss lista üres, akkor egyedül Ls feje "képez" monoton
%% növekvo sorozatot, az Ls farka pedig a maradék.
%%
%% Ha az Rss nem üres, akkor elso részlistájának a fejébol és összes
%% részlistájának a második elemébol képezzük a monoton növekvo sorozatot
%% (azaz a futamot, az eredménypár elso tagját), az Mss minden nemüres
%% részlistájának az elso elemébol pedig a maradéklistát (az eredménypár
%% második tagját).
rampa_3([]) ->
    {[],[]};
rampa_3(Ls) ->
    F = fun([X1,X2|_]) -> X1 =< X2; (_) -> false end,
    {Rss,[_|Mss]} = lists:splitwith(F, tails(Ls)),
    case Rss of
        [] ->
            lists:split(1, Ls);
        [Es|_] ->
            % {futam (eredménypár 1. tagja),
            %  maradéklista (eredménypár 2. tagja}
            {[hd(Es)|[X || [_,X|_] <- Rss]], [X || [X|_] <- Mss]}
    end.

% ------------------------------------------------------------------------
```

```
t() ->
    Xs = [1,2,2,3,2,4,5,6,6,6,7,6,8,2,3,3,4,5,6,0,6,5,4,3,2,1],
    Zs = [1,2,2,3],
    Ms = [2,4,5,6,6,6,7,6,8,2,3,3,4,5,6,0,6,5,4,3,2,1],
        (lnko(96,42) =:= 6)
        and (pi(1000000) =:= 3.1415936535887745)
        and (dec2rad(2, 13) =:= [1,1,0,1])
        and (dec2rad(17, 127) =:= [7,8])
        and (conv(16, 127) =:= {ok, "7f"})
        and (conv(17, 127) =:= error)
        and (safe_last([5,1,2,8,7]) =:= {ok,7})
        and (safe_last([]) =:= error)
        and (nth(2,[a,b,c]) =:= b)
        and (split(3, [a,b,c,d,e]) =:= {[a,b,c],[d,e]})
        and (take([10,20,30,40,50], 3) =:= [10,20,30])
        and (take_2([10,20,30,40,50], 3) =:= [10,20,30])
        and (drop([10,20,30,40,50], 3) =:= [40,50])
        and (drop_2([10,20,30,40,50], 3) =:= [40,50])
        and (prefixes([a,b,c]) =:= [[], [a], [a,b], [a,b,c]])
        and (prefixes_2([a,b,c]) =:= [[], [a], [a,b], [a,b,c]])
        and (tails([1,4,2]) =:= [[1,4,2],[4,2],[2],[]])
        and (sublists(1,[a,b,c]) =:= [{0,[a],2}, {1,[b],1}, {2,[c],0}])
        and (sublists(2,[a,b,c]) =:= [{0,[a,b],1}, {1,[b,c],0}])
        and (parban([a,a,a,2,3,3,a,2,b,b,4,4]) =:= [a,a,3,b,4])
        and (sublist([1,2,3,4],2,2) =:= [2,3])
        and (sublist([1,2,3,4],2,3) =:= [2,3,4])
        and (sublist_2([1,2,3,4],2,2) =:= [2,3])
        and (sublist_2([1,2,3,4],2,3) =:= [2,3,4])
        and (sublist_3([1,2,3,4],2,2) =:= [2,3])
        and (sublist_3([1,2,3,4],2,3) =:= [2,3,4])
        and (vertekek([alma, {s,1}, {v,1}, 3, {v,2}]) =:= [1,2])
        and (vertekek_2([alma, {s,1}, {v,1}, 3, {v,2}]) =:= [1,2])
        and (vertekek_3([alma, {s,1}, {v,1}, 3, {v,2}]) =:= [1,2])
        and (vertekek_4([alma, {s,1}, {v,1}, 3, {v,2}]) =:= [1,2])
        and (append([a,b,c], [1,2,3]) =:= [a,b,c,1,2,3])
        and (revapp([a,b,c], [1,2,3]) =:= [c,b,a,1,2,3])
        and (tails_2([1,4,2]) =:= [[1,4,2],[4,2],[2],[]])
        and (tails_3([1,4,2]) =:= [[1,4,2],[4,2],[2],[]])
        and (sublists_2(1,[a,b,c]) =:= [{0,[a],2}, {1,[b],1}, {2,[c],0}])
        and (sublists_2(2,[a,b,c]) =:= [{0,[a,b],1}, {1,[b,c],0}])
        and (parban_2([a,a,a,2,3,3,a,2,b,b,4,4]) =:= [a,a,3,b,4])
        and (parban_3([a,a,a,2,3,3,a,2,b,b,4,4]) =:= [a,a,3,b,4])
        and (dadogo([a,a,a,2,3,3,a,b,b,b,b,b]) =:=
                 [[a],[a],[3],[b],[b,b],[b],[b,b],[b],[b]])
        and (flatten([1,[2,3],[[[[4]]],[5,[6]]]]) =:= [1,2,3,4,5,6])
        and (flatten_2([1,[2,a],[[[[4]]],[5,["ab"]]]]) =:= [1,2,a,4,5,97,98])
        and (slash(fun(Ss) -> lists:split(3, Ss) end, "jaaaj!!! nem jooo!")
                 =:= ["jaa","aj!","!! ","nem"," jo","oo!"])
        and (slash_2(fun(Ss) -> lists:split(3, Ss) end, "jaaaj!!! nem jooo!")
                 =:= ["jaa","aj!","!! ","nem"," jo","oo!"])
        and (rampa_1(Xs) =:= {Zs, Ms})
        and (rampa_2(Xs) =:= {Zs, Ms})
        and (rampa_3(Xs) =:= {Zs, Ms})
        and true.
```