```
defmodule Dp.GyF1 do

  #import Dp.Gy1

  @moduledoc """
  FP 1 gyakorló feladatok

  @author "hanak@emt.bme.hu"
  @date   "$LastChangedDate: 2021-09-12 19:26:43 +0200 (v, 12 szept 2021) $$"
  """

  # 1.
  @spec conv(r::integer, i::integer) :: res::({:ok, ds::charlist} | :error)
  # Az i decimális egész r alapú számrendszerbe konvertált jegyeinek listája ds
  # res = {:ok, ds}, ha 2 <= r <= 16, egyébként :error
  def conv(r, i) when r >= 2 and r <= 16 do
    {:ok, (for x <- Dp.Gy1.dec2rad(r, i), do: Enum.at('0123456789abcdef', x))}
  end
  def conv(_r, _i), do: :error

  # 2.
  @spec prefixes(xs::[any]) :: zss::[[any]]
  # Az xs lista egyre hosszabbodó prefixumainak listája zss
  def prefixes(xs), do: prefixes(xs, 0)

  @spec prefixes(xs::[any], n::integer) :: zss::[[any]]
  # Az xs lista legalább n hosszú prefixumainak listája zss
  def prefixes(xs, n) do
    case length(xs) >= n do
      true  -> [Dp.Gy1.take(xs, n) | prefixes(xs, n+1)]
      false -> []
    end
  end

  # őrrel:
  @spec prefixes_2(xs::[any]) :: zss::[[any]]
  # Az xs lista egyre hosszabbodó prefixumainak listája zss
  def prefixes_2(xs), do: prefixes_2(xs, 0)

  @spec prefixes_2(xs::[any], n::integer) :: zss::[[any]]
  # Az xs lista legalább n hosszú prefixumainak listája zss
  def prefixes_2(xs, n) when length(xs) >= n, do:
    [Dp.Gy1.take(xs, n) | prefixes_2(xs, n+1)]
  def prefixes_2(_xs, _n), do: []

  # cond-dal:
  @spec prefixes_3(xs::[any]) :: zss::[[any]]
  # Az xs lista egyre hosszabbodó prefixumainak listája zss
  def prefixes_3(xs), do: prefixes_3(xs, 0)

  @spec prefixes_3(xs::[any], n::integer) :: zss::[[any]]
  # Az xs lista legalább n hosszú prefixumainak listája zss
  def prefixes_3(xs, n), do:
    (cond do
      length(xs) >= n -> [Dp.Gy1.take(xs, n) | prefixes_3(xs, n+1)]
      true -> []
    end)

  # 3.
  @spec sublists(xs::[any], n::integer) ::
                      pss::[{b::integer, ps::[any], a::integer}]
  # Az xs lista olyan (folytonos) részlistái az n hosszú ps listák
  # -- a pss eredménylista elemei --, amelyek előtt b és amelyek
  # után a számú elem áll xs-ben
  def sublists(xs, n), do: sublists(xs, n, 0)

  @spec sublists(xs::[any], n::integer, s::integer) ::
                      pss::[{b::integer, ps::[any], a::integer}]
  # Az xs lista olyan (folytonos) részlistái az n hosszú ps listák
  # -- a pss eredménylista elemeinek középső tagja --, amelyek előtt
```

```
  # (b-s) és amelyek után a számú elem áll xs-ben
  def sublists(xs, n, s) do
    case length(xs) < n do
      true  -> []
      false -> [{s, Dp.Gy1.take(xs, n), length(xs)-n} | sublists(tl(xs), n, s+1)]
    end
  end

  @spec sublists_2(xs::[any], n::integer) ::
                      [ {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
  def sublists_2(xs, n), do:
    for b <- 0..length(xs) - n, do:
      { b, Dp.Gy1.take( Dp.Gy1.drop(xs, b), n), length(xs)-b-n }

  # 4.
  @spec sublist(xs::[any], s::integer, n::integer) :: rs::[any]
  # Az xs lista s-edik elemétől kezdődő és n hosszú részlistája rs
  def sublist([], _, _),     do: []
  def sublist(_xs, 0, 0),    do: []
  def sublist([x|_],  0, 1), do: [x]
  def sublist([x|xs], 0, n), do: [x|sublist(xs, 0, n-1)]
  def sublist([_|xs], s, n), do: sublist(xs, s-1, n)

  def sublist_2(xs, s, n), do: Dp.Gy1.take(Dp.Gy1.drop(xs, s), n)

  @spec sublist_3(xs::[any], s::integer, n::integer) :: rs::[any]
  # Az xs lista s-edik elemétől kezdődő és n hosszú részlistája rs
  def sublist_3(xs, s, n) do
  # Enum.at nil-t ad vissza, ha index túlmutat a listán
    for i <- s..s+n-1, do: Enum.at(xs, i)
  end

  # 5.
  @spec append(xs::[any], ys::[any]) :: zs::[any]
  # Az xs lista ys elé fűzésének eredménye zs, azaz zs === xs ++ ys
  def append(xs, ys), do: List.foldr(xs, ys, &([&1|&2]))

  @spec revapp(xs::[any], ys::[any]) :: zs::[any]
  # A megfordított xs lista ys elé fűzésének eredménye zs,
  # azaz zs === Enum.reverse(xs) ++ ys
  def revapp(xs, ys), do: List.foldl(xs, ys, &([&1|&2]))

  # 6.
  @spec tails_2(xs::[any]) :: zss::[[any]]
  # Az xs lista egyre rövidülő szuffixumainak listája zss
  def tails_2(xs) do
    List.foldr(xs, [[]],
      (fn(y, [zs|zss]) -> [[y|zs], zs|zss] end))
  end

  @spec tails_3(xs::[any]) :: zss::[[any]]
  # Az xs lista egyre rövidülő szuffixumainak listája zss
  def tails_3(xs), do:
    for n <- 0..length(xs), do: Dp.Gy1.drop(xs, n)

  # 7.
  @spec flatten(dls::[any]) :: rs::[any]
  # A tetszőleges mélységű beágyazott listákból álló dls (deeplist)
  # lista elemeiből álló, listaelemeket már nem tartalmazó lista az rs
  def flatten([]), do: []
  def flatten([d|dls]) when is_list(d), do: flatten(d) ++ flatten(dls)
  def flatten([d|dls]), do: [d|flatten(dls)] # when not is_list(d)

  @spec flatten_2(dls::[any]) :: rs::[any]
  # A tetszőleges mélységű beágyazott listákból álló dls (deeplist)
  # lista elemeiből álló, listákat már nem tartalmazó lista az rs
  def flatten_2(dls), do: flatten_2(dls, [])
```

```
  @spec flatten_2(dls::[any], ts::[any]) :: rs::[any]
  # flatten_2(dls, ts) = a kilapítva a ts elé fűzött dls az rs lista
  def flatten_2([d|dls], ts) when is_list(d), do: flatten_2(d, flatten_2(dls, ts))
  def flatten_2([d|dls], ts), do: [d|flatten_2(dls, ts)]
  def flatten_2([], ts), do: ts

  # 8.
  @spec slash(xs::[any], (f::[any] -> {ps::[any], ms::[any]})) :: zss::[[any]]
  # Az xs listából az f ismételt alkalmazásával kivágott elemek listája zss, ahol
  # f eredménye a {ps, ms} pár: ps lesz a zss következő eleme, ms pedig az xs még
  # feldolgozásra váró része
  def slash([], _f), do: []
  def slash(xs, f) do
    {z, rs} = f.(xs)
    append([z], slash(rs, f))
  end

  @spec slash_2(xs::[any], (f::[any] -> {ps::[any], ms::[any]})) :: zss::[[any]]
  # Az xs listából az f ismételt alkalmazásával kivágott elemek listája zss, ahol
  # f eredménye a {ps, ms} pár: ps lesz a zss következő eleme, ms pedig az xs még
  # feldolgozásra váró része
  def slash_2(xs, f), do: slash_2(xs, f, [])

  def slash_2([], _f, zs), do: revapp(zs, [])
  def slash_2(xs, f, zs) do
      {z, rs} = f.(xs)
      slash_2(rs, f, [z|zs])
  end

  # 9.
  @spec rampa(xs::[any()]) :: psms::{ps::[any()], ms::[any]}
  # Az xs lista első monoton növekvő futamából (részlistájából),
  # ps-ből és az xs maradéklistájából, ms-ből álló pár psms
  def rampa(rs) do
    case rs do
      [x1,x2|xs] ->
        case x1 <= x2 do
          true ->
            {zs, ms} = rampa([x2|xs])
            {[x1|zs], ms}
          false ->
            {[x1], [x2|xs]}
        end
      rs ->
        {rs, []}  # length(rs) < 2
    end
  end

  # őrrel:
  def rampa_2([]), do: {[], []}
  def rampa_2([x1,x2|xs]) when x1 <= x2 do
    {zs, ms} = rampa_2([x2|xs])
    {[x1|zs], ms}
  end
  def rampa_2([x|xs]), do: {[x], xs}

  # magasabb rendű függvényekkel
  def rampa_3([]), do: {[],[]}
  def rampa_3(xs) do
    zs = Enum.zip(Dp.Gy1.take(xs, length(xs)-1), tl(xs))
    rs = Enum.take_while(zs, fn({x,y}) -> x <= y end)
    Enum.split(xs, (length(rs)+1))
  end

  # 10.
  @spec rampak(xs::[any()]) :: xss::[[any()]]
  # Az xs lista monoton növekvő futamainak (részlistáinak) listája xss
  def rampak([]), do: []
  def rampak(xs) do
    {rs, ms} = rampa(xs)
```

```
    [rs | rampak(ms)]
  end

  @spec rampak_2(xs::[any()]) :: xss::[[any()]]
  # Az xs lista monoton növekvő futamainak (részlistáinak) listája xss
  def rampak_2(xs), do: slash(xs, &rampa_2/1)

  # ---------------------------------------------------------------------------

  def t do
    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]
    rs = [[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]]
    [ (conv(16, 127) === {:ok, '7f'}),
      (conv(17, 127) === :error),
      (prefixes([:a,:b,:c]) === [[], [:a], [:a,:b], [:a,:b,:c]]),
      (prefixes_2([:a,:b,:c]) === [[], [:a], [:a,:b], [:a,:b,:c]]),
      (prefixes_3([:a,:b,:c]) === [[], [:a], [:a,:b], [:a,:b,:c]]),
      (sublists([:a,:b,:c], 1) === [{0,[:a],2}, {1,[:b],1}, {2,[:c],0}]),
      (sublists([:a,:b,:c], 2) === [{0,[:a,:b],1}, {1,[:b,:c],0}]),
      (sublists_2([:a,:b,:c], 1) === [{0,[:a],2}, {1,[:b],1}, {2,[:c],0}]),
      (sublists_2([:a,:b,:c], 2) === [{0,[:a,:b],1}, {1,[:b,:c],0}]),
      (sublist([1,2,3,4],1,2) === [2,3]),
      (sublist([1,2,3,4],1,3) === [2,3,4]),
      (sublist_2([1,2,3,4],1,2) === [2,3]),
      (sublist_2([1,2,3,4],1,3) === [2,3,4]),
      (sublist_3([1,2,3,4],1,2) === [2,3]),
      (sublist_3([1,2,3,4],1,3) === [2,3,4]),
      (append([:a,:b,:c], [1,2,3]) === [:a,:b,:c,1,2,3]),
      (revapp([:a,:b,:c], [1,2,3]) === [:c,:b,:a,1,2,3]),
      (tails_2([1,4,2]) === [[1,4,2],[4,2],[2],[]]),
      (tails_3([1,4,2]) === [[1,4,2],[4,2],[2],[]]),
      (flatten([1,[2,3],[[[4]]],[5,[6]]]]) === [1,2,3,4,5,6]),
      (flatten_2([1,[2,:a],[[[4]]],[5,['ab']]]]) === [1,2,:a,4,5,97,98]),
      (slash('jaaaj!!! nem jooo!', &(Dp.Gy1.split(&1, 3)))
        === ['jaa','aj!','!! ','nem',' jo','oo!']),
      (slash_2('jaaaj!!! nem jooo!', fn(ss) -> Dp.Gy1.split(ss, 3) end)
        === ['jaa','aj!','!! ','nem',' jo','oo!']),
      (rampa(xs) === {zs, ms}),
      (rampa_2(xs) === {zs, ms}),
      (rampa_3(xs) === {zs, ms}),
      (rampak(xs) === rs),
      (rampak_2(xs) === rs),
    ]
  end
end

IO.inspect Dp.GyF1.t()
```