

```

defmodule Dp.Gy1 do

  @moduledoc """
  FP 1 gyakorlat

  @author "hanak@emt.bme.hu"
  @date   "$LastChangedDate: 2022-09-11 19:56:16 +0200 (v, 11 szept 2022) $"
  """

  #-----
  ### Rekurzió, egyszerű listakezelés
  ### Variánsokban komprehenzió és magasabb rendű függvények is
  #-----

  # 1.
  @spec lnko(a :: integer, b :: integer) :: d :: integer
  # a és b legnagyobb közös osztója d
  def lnko(a, 0), do: a
  def lnko(a, b), do: lnko(b, (rem a,b))

  # 2.
  @spec len(xs :: [any]) :: n :: integer
  # Az xs lista hossza n
  def len([], do: 0
  def len([_|xs]), do: 1 + len(xs)

  @spec leni(xs :: [any]) :: n :: integer
  # Az xs lista hossza n
  def leni(xs), do: leni(xs, 0)

  defp leni([], n), do: n
  defp leni([_|xs], n), do: leni(xs, n+1)

  # 3.
  @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 + ...
  def pi(i), do: 4 * pi_4(i*2+1, 1, :p, 0)

  defp pi_4(n, k, :p, pi) when k <= n, do: pi_4(n, k+2, :m, pi+(1/k))
  defp pi_4(n, k, :m, pi) when k <= n, do: pi_4(n, k+2, :p, pi-(1/k))
  defp pi_4(n, k, _, pi) when k > n, do: pi

  # 4.
  @spec dec2rad(r :: integer, i :: integer) :: ds :: [integer]
  # Az i egész szám r alapú számrendszerbe konvertált, decimális számként
  # megadott számjegyeinek listája ds
  def dec2rad(r, i), do: dec2rad(r, i, [])

  defp dec2rad(_r, 0, ds), do: ds
  defp dec2rad(r, i, ds), do: dec2rad(r, (div i, r), [(rem i, r) | ds])

  # 5.
  # Erlang-stílusban
  @spec last_er(xs::[any]) :: r :: ( {:ok, x::any} | :error)
  # Ha xs üres, r == :error, különben { :ok, x}, ahol x az xs utolsó eleme
  def last_er([x]), do: { :ok, x}
  def last_er([]), do: :error
  def last_er([_|xs]), do: last_er xs

  # Elixir-stílusban
  @spec last_ex(xs::[any]) :: r :: ((x::any) | nil)
  # Ha xs üres, r == nil, különben x, ahol x az xs utolsó eleme
  def last_ex([x]), do: x
  def last_ex([]), do: nil
  def last_ex([_|xs]), do: last_ex xs

  # 6.
  @spec at(es::[any], n::integer) :: r :: ((e::any) | nil)
  # Az es lista n-edik eleme e (indexelés 0-tól)

```

```

def at([e|_], 0),          do: e
def at([_|es], n) when n > 0, do: at(es, n-1)
def at(_es, _n),          do: nil

# 7.
@spec split(ls::[any], n::integer) :: {ps::[any], ss::[any]}
# Az ls lista n hosszú prefixuma (első n eleme) ps, length(ls) - n
# hosszú szuffixuma (első n eleme utáni része) pedig ss
def split(ls, 0), do: {[], ls}
def split([l|ls], n) do
  {ps, ss} = split(ls, n-1)
  {[l|ps], ss}
end

# 8.
@spec take(xs::[any], n::integer) :: ps::[any]
# Az xs lista n hosszú prefixuma a ps lista
def take(xs, n) do
  {ps, _} = split(xs, n)
  ps
end

# vagy mintaillesztés nélkül BIF-fel:
@spec take_2(xs::[any], n::integer) :: ps::[any]
def take_2(xs, n), do: elem(split(xs, n), 0)

# vagy rekurzív módon
@spec take_3(xs::[any], n::integer) :: ps::[any]
def take_3([], _n),          do: []
def take_3([x|xs], n) when n>0, do: [ x | take_3(xs, n-1) ]
def take_3(_xs, _n),          do: []

# 9.
@spec drop(xs::[any], n::integer) :: ss::[any]
# Az xs lista első n elemét nem tartalmazó szuffixuma ss
def drop(xs, n) do
  {_, ss} = split(xs, n)
  ss
end

# vagy mintaillesztés nélkül BIF-fel:
@spec drop_2(xs::[any], n::integer) :: ss::[any]
def drop_2(xs, n), do: elem(split(xs, n), 1)

# vagy rekurzív módon
@spec drop_3(xs::[any], n::integer) :: ss::[any]
def drop_3([], _n),          do: []
def drop_3([_x|xs], n) when n>0, do: drop_3(xs, n-1)
def drop_3(xs, _n),          do: xs

# 10.
@spec tails(xs::[any]) :: zss::[[any]]
# Az xs lista egyre rövidülő szuffixumainak listája zss
def tails([_y|ys] = xs), do: [ xs | tails(ys) ]
def tails([]),          do: [ [] ]

# 11.
@spec parban(es::[any]) :: zs::[any]
# Az es lista összes olyan elemének listája zs, amely
# után vele azonos értékű elem áll
def parban([e,e|es]), do: [e|parban([e|es])]
def parban([_e1,e2|es]), do: parban([e2|es])
def parban(_), do: []

@spec parban_2(es::[any]) :: zs::[any]
# Az es összes olyan elemének listája zs, amely után
# vele azonos értékű elem áll
def parban_2(es), do: for [e,e|_] <- tails(es), do: e

@spec parban_3(es::[any]) :: zs::[any]

```

```

# Az es összes olyan elemének listája zs, amely után
# vele azonos értékű elem áll
def parban_3(es) do
  for {e,e} <- Enum.zip(tl(es), take(es, length(es)-1)), do: e
end

# 12.
@spec vertekék(xs::[any]) :: es::[any]
# Az xs lista elemei közül a {:v::atom, e::any} mintára
# illeszkedő párok 2. tagjából képzett lista es
# a lista "darálásával"
def vertekék([{:v, e}|xs]), do: [e | vertekék(xs)]
def vertekék([_|xs]),      do: vertekék(xs)
def vertekék([],           do: []

@spec vertekék_2(xs::[any]) :: es::[any]
# Az xs lista elemei közül a {:v, e::any} mintára
# illeszkedő párok második tagjából képzett lista es
# map-pel és filterrel
def vertekék_2(xs) do
  Enum.map(
    Enum.filter(xs,
      fn({:v, _}) -> true; _ -> false end
    ),
    fn({:v, e}) -> e end
  )
end

@spec vertekék_3(xs::[any]) :: es::[any]
# Az xs lista elemei közül a {:v, e::any} mintára
# illeszkedő párok második tagjából képzett lista es
# mintaillesztés nélkül
def vertekék_3(xs) do
  Enum.map(
    Enum.filter(
      xs,
      fn(x) -> is_tuple(x) and tuple_size(x) === 2 and
        elem(x, 0) === :v end
    ),
    fn({:v, e}) -> e end
  )
end

@spec vertekék_4(xs::[any]) :: es::[any]
# Az xs lista elemei közül a {:v, e::any} mintára
# illeszkedő párok második tagjából képzett lista es
# listajelölővel
def vertekék_4(xs), do: for {:v, e} <- xs, do: e

# 13.
@spec dadogo(xs::[any]) :: zss::[[any]]
# zss az xs lista összes olyan nemüres (folytonos) részlistájából
# álló lista, amelyet vele azonos értékű részlista követ
def dadogo(xs), do: dadogo0(tails xs)

def dadogo0([xs|xss]), do:
  dadogo0(xs, div(length(xs),2), []) ++ dadogo0(xss)
def dadogo0(_, do: []

def dadogo0(_xs, 0, zs), do: zs
def dadogo0(xs, n, zs) do
  {ps, ss} = split(xs, n)
  ns = if (take(ss,n) === ps) do [ps|zs] else zs end
  dadogo0(xs, n-1, ns)
end

@spec dadogo_2(xs::[any]) :: zss::[[any]]
# zss az xs lista összes olyan nemüres (folytonos) részlistájából
# álló lista, amelyet vele azonos értékű részlista követ
### Szolgálati közlemény:

```

```

### Az Elixirben a komprehenzió megszakad a kivétel miatt
### Az Erlang a kivételt false-nak tekinti, a komprehenziót folytatja
def dadogo_2(xs) do
  for ts <- tails(xs),
      ts != [], # split és take kivételt dob üres listára
      n <- 1..(div length(ts), 2),
      {ps, ss} = split(ts, n),
      ps != [],
      ss != [],
      ps == take(ss, n),
      do: ps
end

def t do
  [ (lnko(96,42) == 6),
    (len([]) == 0),
    (len('hosszú_karakterlista') == 20),
    (leni([]) == 0),
    (leni('hosszú_karakterlista') == 20),
    (pi(1000000) == 3.1415936535887745),
    (dec2rad(2, 13) == [1,1,0,1]),
    (dec2rad(17, 127) == [7,8]),
    (last_er([5,1,2,8,7]) == {:ok, 7}),
    (last_er([]) == :error),
    (last_ex([5,1,2,8,7]) == 7),
    (last_ex([]) == nil),
    (at([:a,:b,:c], 0) == :a),
    (at([:a,:b,:c], 1) == :b),
    (at([:a,:b,:c], 3) == nil),
    (split([:a,:b,:c,:d,:e], 3) == {[:a,:b,:c],[:d,:e]}),
    (take([10,20,30,40,50], 3) == [10,20,30]),
    (take_2([10,20,30,40,50], 3) == [10,20,30]),
    (drop([10,20,30,40,50], 3) == [40,50]),
    (drop_2([10,20,30,40,50], 3) == [40,50]),
    (tails([1,4,2]) == [[1,4,2],[4,2],[2],[[]]]),
    (parban([:a,:a,:a,2,3,3,:a,2,:b,:b,4,4]) == [:a,:a,3,:b,4]),
    (parban_2([:a,:a,:a,2,3,3,:a,2,:b,:b,4,4]) == [:a,:a,3,:b,4]),
    (parban_3([:a,:a,:a,2,3,3,:a,2,:b,:b,4,4]) == [:a,:a,3,:b,4]),
    (vertekek([:alma, {S,1}, {v,1}, 3, {v,2}]) == [1,2]),
    (vertekek_2([:alma, {s,1}, {v,1}, 3, {v,2}]) == [1,2]),
    (vertekek_3([:alma, {s,1}, {v,1}, 3, {v,2}]) == [1,2]),
    (vertekek_4([:alma, {s,1}, {v,1}, 3, {v,2}]) == [1,2]),
    (dadogo([:a,:a,:a,2,3,3,:a,:b,:b,:b,:b,:b]) ==
      [[:a],[[:a],[3],[[:b],[[:b,:b],[[:b],[[:b,:b],[[:b],[[:b]]]]]]],
    (dadogo_2([:a,:a,:a,2,3,3,:a,:b,:b,:b,:b,:b]) ==
      [[:a],[[:a],[3],[[:b],[[:b,:b],[[:b],[[:b,:b],[[:b],[[:b]]]]]]],
  ]
end
end

IO.inspect Dp.Gy1.t()

```