Haskell 2.

Lokális függvénydefiníció

- Előre: ***let ... in …***

  ker1 :: Int -> Int -> Int

  ker1 a b = let dupla x = x + x

  in dupla a + dupla b -- ez a visszatérési érték

- Utólag: **... where …**

  ker2 :: Int -> Int -> Int

  ker2 a b = dupla a + dupla b -- ez a visszatérési érték

  where dupla x = x + x

Lista típusok

- A lista **azonos típushoz tartozó elemek sorozata,** melyeket szögletes zárójelek ([és]) között sorolunk fel vesszővel (,) elválasztva.

  pl.:

  [1,2,3,4] 4 elemű lista.

  [] Üres lista

  [[1,2],[3],[4,5,6,7]] Listákat tartalmazó lista

- Infix op.:

  ++ - Listák konkatenálása

  : - Egy elem hozzáfűzése a lista elejére.

Műveletek listákkal

- Beszúrás lista elejére: **:**
- Listán belüli lista: ***[[1,2],[1,2]]*** típusa: **[Int]**
- Listák összefűzése: **++**
- A ***fuggveny [Int] - > …***: Int-listát vár paraméterként
- Head-tail felbontás: **(h:t)**
- pl.: *[1,2,3,4]* lista felbontása: h=1 és t=[2,3,4]
- Okos lista: ***[1 .. 10]***
- Végtelen lista: ***[1 ..]***, páros „végtelen” lista: ***[0, 2 ..]***

-- 1. - Lista elemszáma ----------(kg_lista.hs)-------

```
size :: [Int] -> Int
size [] = 0
size (h:t) = 1 + size t
```

-- 2. - Lista elemeinek összege ----------

```
summ :: [Int] -> Int                    -- input: [1..10]
summ [] = 0
summ (h:t) = h + summ t
```

-- 3. - Pozitív egész számok száma a listában ----------

```
pos :: [Int] -> Int
pos [] = 0
pos (h:t) = if h > 0
            then 1 + pos t
            else pos t
```

-- 4. - Lista megfordítása ----------

```
rev :: [Int] -> [Int]
rev [] = []
rev (h:t) = rev t ++ [h] -- h-t listába kell tenni
```

-- 5. - Lista első n db elemének kiválogatása ----------

```
elson :: [Int] -> Int -> [Int]          -- input: [1..], [1,2..]
elson [] x = []
elson _ 0 = []                          -- _ tetszőleges valami
elson (h:t) n = [h] ++ elson t (n-1)    -- h : elson... is jó
```

6. Határozzuk meg egy lista elemeinek az átlagát! **(hs_3.hs)**

```
atlag1 :: [Float] -> Float
atlag1 ls = (ossz ls) / (hossz ls)
    where
        ossz [] = 0
        ossz (h:t) = h + ossz t
        hossz [] = 0
        hossz (_:t) = 1 + hossz t
```

7. Rendezzünk egy egészeket tartalmazó tömböt gyorsrendezéssel! **(hs_5.hs)**

```
quicksort :: [Int] -> [Int]
quicksort [] = []
quicksort (h:t) = quicksort [x | x<-t, x=h]
```

8. Fibonacci számok előállítása az előadáson bemutatottnál hatékonyabb módon! Majd adjuk meg az n. elemét ennek a halmaznak! **(hs_6.hs)**

```
zip2 (x:xs) (y:ys) = (x, y): zip2 xs ys
zip2 xs ys = []

fib = 1 : 1 : [a+b | (a, b) <- zip2 fib (tail fib)]
get [] n = 0
get (x:xs) n = if (n == 1)
                    then  x
                    else  get xs (n-1)
fibn n = get fib n
```

9. Adott egy sztring. Add meg a benne tárolt szám értékét! **(hs_8.hs)**

```
import Char


ertek :: String -> Int
ertek s = ert s 0


ert :: String -> Int -> Int
ert []    n = n
ert (h:t) n = ert t ((10 * n) + ((ord h) - (ord '0')))
```