Funkcionális programozás 6. gyakorlat

Egyéb hasznos függvények

1. Egy titkosított kódból nyerjük ki az első, olyan két karakter hosszú betűsort, amit szám követ. Az egyes karakterek azonosításához használjuk a Data.Char függvényeit. (`isLetter`, `isDigit`)

   ```
   cipher :: String -> String

   cipher "PYdg7iT4vdOOn4AgmGfUpRzogAf" == "dg"
   cipher "4vkYyAO174midQTt0" == "AO"
   cipher "BwxwEwqCKHuMTAaPn" == ""
   cipher "dM7" == "dM"
   cipher "Kmz" == ""
   cipher "Zk"  == ""
   cipher "T4"  == ""
   cipher "" == ""
   ```

2. Definiáljuk a `zip' :: [a] -> [b] -> [(a,b)]` függvényt, amely két listából párok listáját állít elő!

   ```
   zip' [1..] "almafa" == [(1,'a'),(2,'l'),(3,'m'),(4,'a'),(5,'f'),(6,'a')]
   ```

3. Definiáljuk a `numberedABC :: [(Int, Char)]` függvényt, amely egy párok listája formájában visszaadja az ábécé betűit megszámozva!

4. Definiáld a `stars :: String` függvényt, amely a következő végtelen szöveget állítja elő: `"* ** *** **** *****..."`

   ```
   take 20 stars == "* ** *** **** ***** "
   ```

Lokális definíciók

- Függvények definiálása korlátozott látókörrel
- Kulcsszó: `where`
- A lokális függvények nagyobb behúzással kezdődnek

pl.:

```
f :: Int
f = g where
    g :: Int
    g = 1
```

- A `g :: Int` függvény csak az `f :: Int` definícióján belül használható

```
ghci> g
<interactive>:31:1: error: Variable not in scope: g
```

- Kicsit értelmesebb példa:

```haskell
minToHour :: Int -> (Int, Int)
minToHour x = (h, m) where
    h = x `div` 60
    m = x `mod` 60
```

1. Definiáld a `toUpperFirsts :: String -> String` függvényt, amely egy tetszőleges szöveg minden szavát átalakít nagy kezdőbetűsre!

unzip megoldás - példa

1. Definiáljuk az `unzip' :: [(a,b)] -> ([a],[b])` függvényt, amely párok listájából egy olyan lista párt állít elő, ahol az első lista a párok első elemét és a második lista a második elemét tartalmazza!

   ```haskell
   unzip' [] == ([],[])
   unzip [('a',1), ('b',2), ('c',3)] == ("abc",[1,2,3])
   ```

   Megoldás:

   ```haskell
   unzip :: [(a,b)] -> ([a],[b])
   unzip [] = ([],[])
   unzip ((x,y):xs) = (x:a, y:b) where
       (a,b) = unzip xs
   ```

Hogyan működik?

```haskell
unzip [('a',1), ('b',2), ('c',3)] = ('a':a, '1':b) where
    (a,b) = unzip [('b',2), ('c',3)]
            ------------------------

unzip [('b',2), ('c',3)] = ('b':a, 2:b) where
    (a,b) = unzip [('c',3)]
            ---------------

unzip [('c',3)] = ('c':a, 3:b) where
    (a,b) = unzip []
            --------

unzip [] = ([],[])

unzip [('c',3)] = ('c':[], 3:[])

unzip [('b',2), ('c',3)] = ('b':'c':[], 2:3:[])

unzip [('a',1), ('b',2), ('c',3)] = ('a':'b':'c':[], '1':2:3:[])
```

1. Definiáljuk az `unzip3' :: [(a, b, c)] -> ([a], [b], [c])` függvényt!

```
unzip3' [] == ([],[],[])
unzip3' [(1,2,3),(4,5,6)] == ([1,4],[2,5],[3,6])
```