Funkcionális programozás 10. gyakorlat

Magasabbrendű függvények

1. Definiáld újra a `zipWith` függvényt, mely hasonló a `zip` hez, azonban nem párokat készít, hanem egy kétparaméteres függvényt alkalmaz!

```
zipWith min [1,9,2,5] [5,0,3,8] == [1,0,2,5]
zipWith min [1,0,3] [5,2,10,1] == [1,0,3]
zipWith (*) [2,0,6] [1,5,4,9] == [2,0,24]
```

1. A zipWith felhasználásával készítsünk egy olyan függvényt, amely számok egy sorozatából előállítja azok páronkénti különbségeinek sorozatát!

```
differences [1..5] == [1, 1, 1, 1]
differences [5,4..1] == [-1, -1, -1, -1]
differences [0,1,4,9,16] == [1, 3, 5, 7]
```

1. Definiáljuk az `isUniform :: Eq b => (a -> b) -> [a] -> Bool` függvényt, amely eldönti egy megadott függvényről, hogy egyenletes eloszlású-e, azaz hogy mindig ugyanazzal az értékkel tér-e vissza a megadott értelmezési tartományon.

```
isUniform (*0) [1..100] == True
isUniform length ["alma", "toll", "baba"] == True
isUniform (*1) ([1..100]) == False
isUniform length ["alma", "barack", "baba"] == False
isUniform id ['a'..'z'] == False
```

1. Definiáld a `selectiveApply :: (a -> a) -> (a -> Bool) -> [a] -> [a]` függvényt, amely csak azokra a listaelemekre alalmazza a paraméterül kapott függvényt, amelyekre teljesül a szintén paraméterként kapott predikátum.

```
selectiveApply (*3) (>10) [] == []
selectiveApply (*3) (>10) [1,2,3,4,5] == [1,2,3,4,5]
selectiveApply (*3) (>5) [5,10,0,-2,6,11] == [5,30,0,-2,18,33]
selectiveApply (^2) (even) [4,8,2,1,9,7] == [16,64,4,1,9,7]
```

Dollár operátor

1. Definiáljuk újra a dollár operátort!

Kompozíció

1. Definiáljuk újra a függvénykompozíciót!
2. Definiáljuk a `dropSpaces` függvényt, mely szóközöket dob el egy `String` elejéről!

```
dropSpaces " hi h i " == "hi h i "
dropSpaces "alma fa " == "alma fa "
dropSpaces "" == ""
```

1. Definiáljuk a `trim` függvényt, mely szóközöket dob el egy `String` mindkét végéről!

```
trim " hello! " == "hello!"
trim "Haskell" == "Haskell"
trim "" == ""
```

1. Definiáljuk a `monogram` függvényt, mely egy név monogramját adja meg! Használd a `words`-öt és magasabbrendű függvényt.

```
monogram "Jim Carrey" == "J. C."
monogram "Ilosvai Selymes Péter" == "I. S. P."
```

1. Definiáljuk a `uniq :: Ord a => [a] -> [a]` függvényt, mely elhagyja az ismétléseket!

A megoldáshoz érdemes használni rendezést (`sort` a `Data.List` -ből) majd a `group` függvényt (`Data.List`).

```
uniq "Mississippi" == "Mips"
uniq "papagaj" == "agjp"
uniq "" == ""
```

Névtelen függvények (lambdák)

1. Írj a következő függvényekkel ekvivalens névtelen függvényt!

```
f1 :: Int -> Bool
f1 a = even a && a `mod` 5 == 0

f2 :: Num a => a -> a -> a
f2 a b = a^2 + b^2

f3 :: (b, a) -> (a, b, a)
f3 (a,b) = (b,a,b)

f4 :: [b] -> ([b], b)
f4 (x:xs) = (xs, x)
```

1. Adott egész számok listáinak listája. Add meg azon listák listáját az `evenList :: Integral a => [[a]] -> [[a]]` függvény segítségével, amelyekben a számok összege páros!
2. Definiáljuk a fibonacci végtelen listát az `iterate` függvény segítségével! A sorozat előállítási szabálya: `(a, b) -> (b, a+b)`.

Fogalmak

- Magasabbrendű függvény
  - Fogalom + adj példát, magasabb rendű függvény-e az `elem`?
- Szeletek
  - Mik a szeletek, adj 3 példát!/ Igaz-e, hogy a szeleteket csak magasabbrendű függvények paramétereként tudjuk használni?
- Parciális alkalmazás
  - Fogalom + adj példát, magasabb rendű-e a (map (+1)) függvény
- Névtelen függvény/lambda
  - Fogalom, írj egy függvénnyel ekvivalens lambdát vagy fordítva
- Függvénykompozíció
  - Mi a különbség a `(map (+1)) . (filter (<5))` és `(filter (<5)) . (map (+1))` között?