Funkcionális programozás 4. gyakorlat

Emlékeztető: Minták listákra

http://lambda.inf.elte.hu/Patterns.xml#mint%C3%A1k-list%C3%A1kra

- Üres lista minta: `[]`
- Egyelemű lista: `[a]`
- Kételemű lista: `[a,b]`
- Nemüres lista minta: `a:b`

1. Definiáljuk a `duplicateElements :: [a] -> [a]` függvényt, amely előállít egy olyan listát, hogy az eredeti lista minden elemét kétszer tartalmazza!

```
duplicateElements [] == []
duplicateElements "alma" == "aallmmaa"
duplicateElements [1..10] == [1,1,2,2,3,3,4,4,5,5,6,6,7,7,8,8,9,9,10,10]
```

2. Definiáljunk egy `everySecond :: [a] -> [a]` függvényt, amely kiválogat minden második elemet egy listából!

```
everySecond [] == []
everySecond [1..10] == [2,4,6,8,10]
```

3. Definiáljuk a `concat' :: [[a]] -> [a]` függvényt, amely összefűz egy listák listáját egyetlen listává!

```
concat' [] == []
concat' [[1],[2..5],[3]] == [1,2,3,4,5,3]
concat' ["ez ", "egy ", "lista"] == "ez egy lista"
```

4. Definiáljuk a `take' :: Int -> [a] -> [a]` függvényt, amely visszaadja egy lista első n elemét és eldobja a maradékát!

```
take' 0 [1..5] == []
take' 100 [1..5] == [1,2,3,4,5]
take' 5 "Hello Haskell" == "Hello"
```

5. Definiáljuk a `drop' :: Int -> [a] -> [a]` függvényt, amely eldobja egy lista első n elemét és visszaadja a maradékát!

```
drop' 0 [1..5] == [1,2,3,4,5]
drop' 100 [1..5] == []
drop 5 "Hello Haskell" == " Haskell"
```

6. Definiáljuk az `isPrefixOf :: Eq a => [a] -> [a] -> Bool` függvényt, amely eldönti, hogy az első paraméterként megadott lista prefixe-e a második listának!

```
isPrefixOf [] [1..5] == True
isPrefixOf "Hello" "Hello Haskell" == True
isPrefixOf "Hello Haskell" "Hello" == False
```

7. Definiáljuk az `insert :: Ord a => a -> [a] -> [a]` függvényt, amely beszúr egy paraméterként megadott elemet egy rendezett listába.

```
insert 5 [] == [5]
insert 5 [1,10] == [1,5,10]
insert 5 [1,2] == [1,2,5]
take 5 (insert 0 [1..]) == [0,1,2,3,4]
```

8. Definiáljuk a `merge :: Ord a => [a] -> [a] -> [a]` függvényt, amely összefésül két rendezett listát.

```
merge [] [1..5] == [1,2,3,4,5]
merge [1..5] [] == [1,2,3,4,5]
merge [1,3..9] [2,4..10] == [1,2,3,4,5,6,7,8,9,10]
```

9. Készítsünk egy `sublist :: Int -> Int -> [a] -> [a]` függvényt, ami kivág egy listából egy részt! Az első paraméter az index, ahonnan a részlista kezdődik és a második paraméter a vágás hossza.

```
sublist 1 2 [] == []
sublist 0 5 "Hello Haskell!" == "Hello"
sublist 6 7 "Hello Haskell!" == "Haskell"
sublist 6 100 "Hello Haskell!" == "Haskell!"
sublist 100 100 "Hello Haskell!" == []
```

10. Listában, rendezett párokban tároljuk egy pizza összetevőinek nevét és árát. Adjuk meg a kész pizza árát, ha tudjuk, hogy az alapanyagokon túl 500 Ft munkadíjat számolunk fel.

```
pizza :: [(String, Int)] -> Int

pizza [("teszta", 200), ("paradicsomszosz", 150), ("pepperoni", 200), ("cheddar", 300)] == 1
pizza [("teszta", 200), ("sajtszosz", 130), ("bacon", 200), ("chili", 300), ("tukortojas", 2
pizza [("vekony teszta", 180), ("besamel", 120), ("bazsalikom", 200), ("kaviar", 2000), ("go
```