A házi feladatot egy `Homework5` nevű modulként kell beadni. Figyeljetek arra, hogy a függvényeitek a module szóval egy "oszlopba" kerüljenek, azaz ne legyenek beljebb húzva! Minden definiálandó függvényhez adjuk meg a hozzá tartozó típus szignatúrát is! (Ezt most megadtam, a saját modulotokba is másoljátok be a definíciótok elé.)

Ismert magasabbrendű függvények

- `map :: (a -> b) -> [a] -> [b]`
- `filter :: (a -> Bool) -> [a] -> [a]`
- `zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]`
- `takeWhile :: (a -> Bool) -> [a] -> [a]` - addig szedi az elemeket egy listából, amíg a predikátum igazat ad. Pl.: `takeWhile (<3) [1,2,3,2,0] == [1,2]`
- `dropWhile :: (a -> Bool) -> [a] -> [a]` - addig dobja el az elemeket gy listából, amíg a predikátum igazad ad. Pl.: `dropWhile (<3) [1,2,3,2,0] == [3,2,0]`
- `span :: (a -> Bool) -> [a] -> ([a], [a])` - a fenti két függvény kombinációja. Pl.: `span (<3) [1,2,3,2,0] == ([1,2], [3,2,0])`

Rekurzió akkumulátorokkal

Az alábbi függvényeket vagy rekurzióval vagy a fenti magasabbrendű függvények használatával lehet megírni (kézzel megírt függvényeket is lehet használni).

- Definiálj egy függvényt ami minden elemhez az indexét csatolja! (`assocIndex :: [a] -> [(Int, a)]`)
- Definiálj egy függvényt ami egy függvényt és egy listát vár paraméterül. A függvény alkalmazza az első paraméterül kapott függvényt a lista összes elemére. A paraméterül kapott függvény egy listaelemet és annak indexét várja paraméterül. (`mapWithIndex :: (Int -> a -> b) -> [a] -> [b]`)

Magasabbrendű függvények írása

Az alábbi függvényeket vagy rekurzióval vagy a fenti magasabbrendű függvények használatával lehet megírni (kézzel megírt függvényeket is lehet használni).

- Definiálj egy függvényt ami megnézi egy adott predikátum igaz-e minden listaelemre. (`allTrue :: (a -> Bool) -> [a] -> Bool`)
- Definiálj egy függvényt ami páronként alkalmaz egy függvényt egy lista elemeire. (`pairwiseMap :: (a -> a -> b) -> [a] -> [b]`)
- Definiálj egy függvényt ami megkeresi az első olyan elemnek az indexét, amire egy predikátum igaz. Ha nincs ilyen adjunk vissza (-1)-et. (`findIndex :: (a -> Bool) -> [a] -> Int`)
- Definiálj egy függvényt ami egy végtelen listát ad vissza, egy elemet és egy függvényt vár paraméterül. A lista `k`-adik eleme tartalmazza a paraméterül kapott elemet és rá a paraméterül kapott függvényt `k`-szor alkalmazva. (`kMap :: a -> (a -> a) -> [a]`)

Magasabbrendű függvények használata

Az alábbi függvényeket sokkal egyszerűbb magasabbrendű függvényekkel (és egy kis rekurzióval) megírni, de nem kötelező.

- Definiálj egy függvényt ami az egymást követő elemeket csoportosítja. (`group :: Eq a => [a] -> [[a]]`)
- Definiálj egy függvényt ami a négyzetszám indexű elemeket kiszűri egy listából. (`removeSqIndices :: [a] -> [a]`)

Tesztek a működésre:

```
assocIndex [True, False] == [(0, True), (1, False)]
take 2 (assocIndex [1..]) == [(0,1), (1,2)]
mapWithIndex (\i _ -> i) [3,1,8] == [0,1,2]
mapWithIndex (+) [-10,-9..(-1)] == [-10,-8,-6,-4,-2,0,2,4,6,8]
take 2 (mapWithIndex (\i x -> sqrt (fromIntegral (i * i + x * x))) [3,4..]) == [3.0, sqrt
17]
allTrue id [True, True]
allTrue (> 2) [3,4,5]
not (allTrue (> 2) [5,4,3,2])
not (allTrue (> 2) [10,9..])
pairwiseMap (+) [1,3,3,1] == [4,6,4]
pairwiseMap (+) [1] == []
pairwiseMap (-) [1..10] == replicate 9 (-1)
findIndex (>2) [1,2,3,4] == 2
findIndex (>2) [1,2] == (-1)
findIndex (>2) [0..] == 3
take 3 (kMap 1 (+1)) == [1,2,3]
take 5 (kMap 2 (*2)) == [2,4,8,16,32]
take 4 (kMap True not) == [True, False, True, False]
group [1,1,2,2,2,3,4,3,3] == [[1,1], [2,2,2], [3], [4], [3,3]]
removeSqIndices [10,9..0] == [8,7,5,4,3,2,0]
```