A házi feladatot egy `Homework3` 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.)

Listafüggvények emlékeztető:

- `head :: [a] -> a` - visszaadja egy lista első elemét.
- `tail :: [a] -> [a]` - eldobja egy lista első elemét.
- `take :: Int -> [a] -> [a]` - visszaadja egy lista első `k` elemét.
- `drop :: Int -> [a] -> [a]` - eldobja egy lista első `k` elemét.
- `reverse :: [a] -> [a]` - megfordítja az elemek sorrendjét egy listában.
- `last :: [a] -> a` - visszaadja egy lista utolsó elemét.
- `init :: [a] -> [a]` - eldobja egy lista utolsó elemét.
- `(++) :: [a] -> [a] -> [a]` - összefűz két listát.
- `(:) :: a -> [a] -> [a]` - egy elemet egy lista elejére fűz.
- `repeat :: a -> [a]` - végtelenszer ismétel egy elemet.
- `replicate :: Int -> a -> [a]` - `k`-szor ismétel egy elemet.
- `length :: [a] -> Int` - visszaadja hogy a lista hossza mennyi.
- `null :: [a] -> Bool` - megnézi egy lista üres-e (működik végtelen listára).
- `concat :: [[a]] -> [a]` - egy listának az összes allistáját összefűzi (Pl `concat [[1,2,3],[],[4]] == [1,2,3,4]`)

Listafüggvények használata

- Definiálj egy függvényt ami egy adott alsó és felső határ esetén visszaad egy listát amelyben az elemek a felső határtól az alsó határig csökkennek egyesével! (`descending :: Int -> Int -> [Int]`)
- Definiálj egy függvényt ami `k` darab 0-t rak egy lista elejére és végére! (`padWithZeros :: Int -> [Int] -> [Int]`)
- Definiálj egy függvényt ami egy listát végtelenszer saját magához fűz! (`repeatList :: [a] -> [a]`)
- Definiálj egy függvényt ami egy számot vár paraméterül és megnézi hogy lista hosszab-e annál! (`isLongerThan :: Int -> [a] -> Bool`)

Listagenerátorok

Segítség: Listagenerátorban a `<-` bal oldalán lehet az adott kifejezésre mintaillleszteni, pl `(a,b) <- xs`.

- Definiálj egy függvényt, amely kiszűri a "bokor" szavakat egy listából! (`trimTheBushes :: [String] -> [String]`)
- Definiálj egy függvényt amely egy számhoz hozzácsatolja a gyökét. Ha a szám negatív szűrjük ki a listából! (`addSqrt :: [Double] -> [(Double, Double)]`)

- Definiálj egy függvényt amely egy listányi karakter és számot kap paraméterül. Ezekből csináljon egy `String`-et ami az adott sorrendben tartalmazza a karaktereket annyiszor, amennyi a hozzácsatolt szám értéke! (`decompress :: [(Char, Int)] -> String`)
- Definiálj egy függvényt amely egy listányi tuple-t kap paraméterül és visszaadja a tupleök elemeit növekvő sorrendben! (`sortTuples :: Ord a => [(a,a,a)] -> [(a,a,a)]`)

Tesztek a működésre:

```
descending 3 1 == [3,2,1]
descending 0 0 == [0]
descending 0 1 == []
padWithZeros 2 [1,2,3] == [0,0,1,2,3,0,0]
padWithZeros 0 [] == []
take 3 (padWithZeros 1 [1..]) == [0,1,2]
take 10 (repeatList [1..5]) == [1,2,3,4,5,1,2,3,4,5]
take 3 (repeatList [1]) == [1,1,1]
not (isLongerThan 3 [1,2,3])
isLongerThan 2 [1,2,3]
isLongerThan 3 [1..]
trimTheBushes ["a"] == ["a"]
trimTheBushes ["bokor", "b"] == ["b"]
take 10 (trimTheBushes $ (replicate 10 "a") ++ (repeat "bokor")) == replicate 10 "a"
addSqrt [1,0,4] == [(1,1), (0,0), (4,2)]
addSqrt [(-1),2,(-10000)] == [(2, sqrt 2)]
decompress [('k',1),('a',1),('c',3),('k',1),('a',1),('c',2)] == "kaccckacc"
sortTuples [(1,2,3), (3,4,2)] == [(1,2,3), (2,3,4)]
```