Házi feladat

A házi feladatot egy `Homework5` nevű modulként kell beadni. Minden definiálandó függvényhez adjuk meg a hozzá tartozó típus szignatúrát is! A feladatok után a zárójelben lévő név azt jelzi, milyen néven kell definiálni az adott függvényt, kifejezést. A feladatok megoldása során törekedjetek arra, hogy a gyakorlaton tanult módszereket, megoldási meneteket használjátok.

A bónusz feladatokat nem kötelező megoldani, megoldásukhoz a gyakorlaton vett anyag továbbgondolása szükséges.

Lista konstrukció

Adjuk meg azt a függvényt, amely a kapott paraméterét belerakja egy listába! A végeredmény egy egy elemű lista legyen, amelyben a kapott paraméter szerepel.

```
putIntoList :: a -> [a]
```

Adjuk meg azt a függvényt, amely két egész számot kap, és előállítja a köztük értelmezett intervallumot! Például `interval 2 5 == [2,3,4,5]`. Abban az esetben, ha az első argumentum nagyobb, mint a második, akkor üres listát adjon vissza!

```
interval :: Int -> Int -> [Int]
```

Lista dekonstrukció

Adjuk meg azt a függvényt, amely egy rendezett párt állít elő egy listából. A pár első eleme a lista első eleme, a pár második eleme pedig a lista farok része legyen. Feltételezhetjük, hogy a lista nem üres.

```
headTail :: [a] -> (a, [a])
```

Adjuk meg azt a függvényt, amely kap két listát, és egy rendezett párt állít elő, amelynek első eleme az első lista első eleme, a második eleme a második lista első eleme! Feltételezhetjük, hogy a listák közül egyik sem üres.

```
doubleHead :: [a] -> [b] -> (a, b)
```

Rekurzív függvények

Adjuk meg azt a függvényt, amely eldönti egy listáról, hogy szerepel-e benne a `0`!

```
hasZero :: [Int] -> Bool
```

Adjuk meg azt a függvényt, amely eldönti egy listáról, hogy szerepel-e benne az üres lista!

```
hasEmpty :: [[a]] -> Bool
```

Adjuk meg azt a függvényt, amely egy lista összes elemét megszorozza kettővel!

```
doubleAll :: [Int] -> [Int]
```

Adjuk meg azt a függvényt, amely a kapott két listáról eldönti, hogy az első hosszabb-e, mint a második! Feltételezhetjük, hogy legalább az egyik lista mindig véges hosszú. Viszont ügyeljünk arra, hogy minden ilyen esetben terminál jon a függvényünk.

```
isLonger :: [a] -> [b] -> Bool
```

Az alábbi tesztesetek közül mindegyiknek `True`-t kell adnia:

```
isLonger [] []         == False
isLonger [] [0]        == False
isLonger [0] []        == True
isLonger [1..9] [1..5] == True
isLonger [1..5] [1..9] == False
isLonger [1..]  [1..9] == True
```

Bónusz

Adjuk meg azt a függvényt, amely egy listából kiválogatja a páros számokat!

```
evens :: [Int] -> [Int]
```

Adjuk meg azt a függvényt, amely összeadja egy lista első `n` elemét! Például `sumFirst 2 [1,2,3] == 3`.

```
sumFirst :: Int -> [Int] -> Int
```

Adjuk meg azt a függvényt, amely egy listából kiválogat minden második elemet, kezdve az első elemmel!

```
everySecond :: [a] -> [a]
```

Segítség: Definiáljunk egy segédfüggvényt, amely egy extra paraméterében tárolja, hogy éppen hanyadik elemet dolgozzuk fel!