Funkcionális programozás 6. gyakorlat

1. Definiálj egy `isPrime` függvényt, mely megvizsgálja egy természetes számról, hogy prím-e!

```
not (isPrime 0)
not (isPrime 1)
not (isPrime 6)
isPrime 2
isPrime 3
isPrime 7
isPrime 101
```

2. Definiálj egy `primes` konstanst, amely prímek végtelen listáját tárolja!

```
take 5 primes == [2, 3, 5, 7, 11]
```

3. Vizsgáld meg, hogy egy lista csak pozitív számokat tartalmaz-e! Érdemes bevetni a `null` függvényt a hatékonyság miatt.

```
allPositive [4, 5, 6, 8, 14]
allPositive [1..100]
allPositive []
not (allPositive [10, 9 ..])
not (allPositive [100, 98 .. 0])
```

4. Állítsd elő azt a listát, amely párokként tartalmazza az összes dominót:
`[(0,0), (0,1), ..., (0,6), (1,1), ..., (6, 6)]`

```
length dominoes == 28
```

5. *Sorold fel az összes természetes számpárt:
`[(0,0),(0,1),(1,0),(0,2),(1,1),(2,0), ...]`

6. Sorszámozd meg az angol ábécé betűit!

```
take 5 alphabet == [(0, 'a'), (1, 'b'), (2, 'c'), (3, 'd'), (4, 'e')]
length alphabet == 26
```

Tipp: itt jól jön a `zip` függvény.

7. Gyűjtsd ki minden harmadik betűt az angol ábécéből egy listakifejezéssel!

```
everyThird == "cfilorux"
```

Tipp: itt jól jön a `cycle` és a `zip` függvény.

8. Neptunban a tárgyakat egy listában tároljuk. Minden tárgynak van neve és feljelentkezett diákjai:

```
courses =
  [ ("Programozasi nyelvek II.", [("Horvath", "Istvan", "BDE91E"), ("Fodros", "Aniko", "DDA3KX")])
  , ("Imperativ programozas",    [("Nemeth", "Eniko", "ALX1K0"), ("Horvath", "Istvan", "BDE91E")])
  , ("Funkcionalis programozas", [("Kiss", "Elemer", "ABCDE6"), ("Nagy", "Jakab", "CDE560")])
  ]
```

Egyetlen listakifejezéssel sorold fel a Funkcionális programozást felvett diákok Neptun-kódját! Nem ér indexelést használni, nem tehetünk fel semmit a tárgyak helyéről a listában.

`students == ["ABCDE6", "CDE560"]`

Általánosítsd a kódot függvénnyé, azaz várd paraméterül a keresett tárgy nevét!

9. *Állítsuk elő azt a listát, amely sorrendben tartalmazza az összes (hónap, nap) párt egy 365 napos évben!

   Tipp: használjuk az `elem :: a -> [a] -> Bool` függvényt!

```
elem (1, 31) calendar
not (elem (1, 32) calendar)
elem (2, 28) calendar
not (elem (2, 29) calendar)
elem (4, 30) calendar
not (elem (4, 31) calendar)
length calendar == 365
```

10. Adott egy `"aaaabccaadeeee"` betűsorozat. Tömörítsd össze úgy, hogy az egymást követő betűk tárolása darabszám-betű pár legyen:

    `[(4,'a'), (1,'b'), (2,'c'), (2,'a'), (1,'d'), (4,'e')]`!

    Tipp: itt hasznos lehet a `group` függvény.

```
compress "aaaabccaadeeee" == [(4,'a'), (1,'b'), (2,'c'), (2,'a'), (1,'d'), (4,'e')]
compress "oh hello!!"     == [(1,'o'),(1,'h'),(1,' '),(1,'h'),(1,'e'),(2,'l'),(1,'o'),(2,'!')]
compress ""               == []
```

11. Definiálj egy `decompress` függvényt, mely visszaállítja a karaktersorozatot a tömörített formából!

    Tipp: itt hasznos lehet a `concat :: [[a]] -> [a]` és a `replicate :: Int -> a -> [a]` függvény.

```
decompress [(4,'a'), (1,'b'), (2,'c'), (2,'a'), (1,'d'), (4,'e')]           == "aaaabccaadeeee"
decompress [(1,'o'),(1,'h'),(1,' '),(1,'h'),(1,'e'),(2,'l'),(1,'o'),(2,'!')] == "oh hello!!"
decompress [] == ""
```