Mintaillesztés

A feladatokban tessék használni mintaillesztést a 2,3,4. feladatokban, mást nem szabad. Elágazást, `if-then-else`-et nem szabad használni, mert nem tanultuk és nincs is rájuk szükség.

A tesztek között lesznek olyan esetek is, amiket még nem láttatok; azok egy részéről a következő órán mesélek. Akiket esetleg előbb érdekel, írjanak nyugodtan, elmondom szívesen. A másik részről is szívesen mesélek, de azok csak később kerülnek elő. Illetve akinek jól jön, írok egy függvényt, amit bemásolva a megoldásba ki lehet próbálni ghci-ben, hogy a tesztek működnek-e. És ha nem megy valamelyik, akkor ki is írja, hogy melyik az, amelyik nem megy.

0. Modul

Definiálj egy modult `Hazi3` néven.

1. Oszthatóság

Készíts egy olyan függvényt `divides` néven, ami paraméterül vár *két* ugyanolyan típusú *egész számot* és megmondja, hogy az első osztója-e a másodiknak. A matematikai definíció szerint a `0` osztója a `0`-ának! A negatív számokkal nem szükséges foglalkozni, feltehető, hogy negatív számot paraméterként ez a függvény soha nem fog kapni.

Megjegyzés: Ez a függvény szeretne lenni a matematikából ismert `|` jelölés. Pl. `2 | 4` jelenti matematikában azt, hogy "Kettő osztója négynek".

2a. Pontosan egy elemű

Írj egy olyan függvényt `isSingleton` néven, amely megállapítja egy listáról, hogy pontosan `1` elemű-e.

2b. Kettő vagy legalább négy elemű

Írj egy olyan függvényt `exactly2OrAtLeast4` néven, amely megállapítja egy listáról, hogy pontosan `2` vagy legalább `4` elemű-e.

3. Első két elem

Definiáld a `firstTwoElements` függvényt, amely visszaadja egy lista első két elemét listaként, ha van legalább két elem. Ha nincs két elem, akkor adj vissza üres listát.

4. Harmadik nélkül

Definiáld a `withoutThird` függvényt, amely egy legalább 3 elemű listából kihagyja a harmadik elemét. Ha listában nincs 3 elem, akkor amúgy sincs benne a harmadik elem, így az eredmény csak a paraméterül kapott lista legyen.

5. Különböző típusú egész számok

Készítsünk függvényt, amely két különböző típusú (pl. Int és Integer) egész számot össze tud adni és eredményül `Integer`-t ad vissza.
A függvény neve legyen `add`.
Adjuk meg a függvény lehető legáltalánosabb típusát a feladat értelme szerint. (Nem feltétlenül esik egybe azzal, amit a ghci mond, hiszen az nem tudja figyelembe venni jelen feladat szövegét.)
Órán elhangzott típusosztályok használata bőven elég.

Megjegyzés: A feladat értelme szerint nincs megkötve, hogy melyik típusú érték hányadik paraméterként szerepel, mindkettőre működnie kell úgy, hogy erre összesen egy darab függvényt írunk és nem két különbözőt.
Például:

```
add (2 :: Int) (3 :: Integer) == (5 :: Integer)
add (2 :: Int) (3 :: Int) == (5 :: Integer)
add (2 :: Integer) (3 :: Int) == (5 :: Integer)
add (2 :: Integer) (3 :: Integer) == (5 :: Integer)
```

6. Egy elemű listák

Definiáld azt a függvényt, amely egy listából megtartja a pontosan egy elemű listákat! A megoldásban használj listagenerátort!

```
onlySingletons :: [[a]] -> [[a]]
```

7. Szöveg visszaállítása

Egy tömörített szöveg úgy van megadva, hogy listában fel van sorolva az adott karakter és mellette hogy hány darab van az adott karakterből. Írjuk meg azt a függvényt, ami visszaállítja a tömörített formából az eredeti szöveget! A megoldáshoz használj listagenerátort!

Segítség: Csak nem sikerült megmutatnom, de listagenerátorban lehet ugyanúgy mintaillleszteni a <- bal oldali változóján.

```
decompress :: [(Char, Int)] -> String
```

Bónusz feladat: Logikai kizárt vagy egy sorban

Definiáld az `xor` függvényt egyetlen sorban minél rövidebben. A függvény akkor ad vissza igazat, ha pontosan egy igaz található a paraméterek között. Ehhez értelemszerűen nem kell mintaillesztést használni, azzal nehéz lenne egy sorban megírni. Elágazás, `if-then-else` továbbra sem kell, de ezeken kívül lehet kreatívkodni.

Segítség: A legrövidebb megoldás mindössze egy megfelelő függvény meghívását igényli.

Tesztesetek

```
0 `divides` (0 :: Int)
not (0 `divides` 12)
```

```
not (0 `divides` 100)
12 `divides` (0 :: Integer)
10 `divides` 0
120 `divides` 0
not (12 `divides` 3)
3 `divides` 12
1 `divides` 2
1 `divides` 3
1 `divides` 4
1 `divides` 13
not (13 `divides` 1)
isSingleton [1]
isSingleton "x"
not (isSingleton [])
not (isSingleton ['x','a'])
not (isSingleton [1..])
not (isSingleton [5,4,6])
not (isSingleton [6,5])
exactly2OrAtLeast4 "alma"
exactly2OrAtLeast4 "te"
exactly2OrAtLeast4 [1,10]
exactly2OrAtLeast4 [1.5,9.25]
exactly2OrAtLeast4 [1..]
not (exactly2OrAtLeast4 [])
not (exactly2OrAtLeast4 [1])
not (exactly2OrAtLeast4 [1,2,3])
not (exactly2OrAtLeast4 "nem")
not (exactly2OrAtLeast4 "a")
firstTwoElements [1,2] == [1,2]
firstTwoElements "alma" == "al"
firstTwoElements [20,9,8,7,6,5,4,3,2,1,0] == [20,9]
firstTwoElements [2,9,5,-3,-7,6,10,2,2,3,3] == [2,9]
firstTwoElements [10..] == [10,11]
firstTwoElements [1.5] == []
firstTwoElements [2] == []
firstTwoElements "a" == []
firstTwoElements ([] :: [Integer]) == []
withoutThird [1,2] == [1,2]
withoutThird [1,2,3] == [1,2]
withoutThird "alma" == "ala"
withoutThird [20,9,8,7,6,5,4,3,2,1,0] == [20,9,7,6,5,4,3,2,1,0]
withoutThird [2,9,5,-3,-7,6,10,2,2,3,3] == [2,9,-3,-7,6,10,2,2,3,3]
take 5 (withoutThird [10..]) == [10,11,13,14,15]
withoutThird [1.5] == [1.5]
withoutThird [2] == [2]
withoutThird "a" == "a"
withoutThird ([] :: [Int]) == []
add (2 :: Int) (3 :: Integer) == (5 :: Integer)
add (2 :: Int) (3 :: Int) == (5 :: Integer)
add (2 :: Integer) (3 :: Int) == (5 :: Integer)
add (2 :: Integer) (3 :: Integer) == (5 :: Integer)
onlySingletons [[],[1],[1,2],[1,2,3],[2],[2,3],[3]] == [[1],[2],[3]]
onlySingletons [[1..],[],[1..10]] == []
onlySingletons ["a","b","c","d","e","f"] == ["a","b","c","d","e","f"]
decompress [] == ""
decompress [('k',1),('a',1),('t',2)] == "katt"
decompress [('A',3),('b',2),('k',5),('D',3)] == "AAAbbkkkkkDDD"
```

Tesztfüggvények

```
allPassed :: Bool
allPassed = null allTests

allTests :: [(String, Bool)]
allTests = [ (x,y) |
    (x,y) <- [ ("0 `divides` (0 :: Int)", 0 `divides` (0 :: Int))
             , ("not (0 `divides` 12)", not (0 `divides` 12))
             , ("not (0 `divides` 100)", not (0 `divides` 100))
             , ("12 `divides` (0 :: Integer)", 12 `divides` (0 :: Integer))
             , ("10 `divides` 0", 10 `divides` 0)
             , ("120 `divides` 0", 120 `divides` 0)
             , ("not (12 `divides` 3)", not (12 `divides` 3))
             , ("3 `divides` 12", 3 `divides` 12)
             , ("1 `divides` 2", 1 `divides` 2)
             , ("1 `divides` 3", 1 `divides` 3)
             , ("1 `divides` 4", 1 `divides` 4)
             , ("1 `divides` 13", 1 `divides` 13)
             , ("not (13 `divides` 1)", not (13 `divides` 1))
             , ("isSingleton [1]", isSingleton [1])
             , ("isSingleton \"x\"", isSingleton "x")
             , ("not (isSingleton [])", not (isSingleton []))
             , ("not (isSingleton ['x','a'])", not (isSingleton ['x','a']))
             , ("not (isSingleton [1..])", not (isSingleton [1..]))
             , ("not (isSingleton [5,4,6])", not (isSingleton [5,4,6]))
             , ("not (isSingleton [6,5])", not (isSingleton [6,5]))
             , ("exactly2OrAtLeast4 \"alma\"", exactly2OrAtLeast4 "alma")
             , ("exactly2OrAtLeast4 \"te\"", exactly2OrAtLeast4 "te")
             , ("exactly2OrAtLeast4 [1,10]", exactly2OrAtLeast4 [1,10])
             , ("exactly2OrAtLeast4 [1.5,9.25]", exactly2OrAtLeast4 [1.5,9.25])
             , ("exactly2OrAtLeast4 [1..]", exactly2OrAtLeast4 [1..])
             , ("not (exactly2OrAtLeast4 [])", not (exactly2OrAtLeast4 []))
             , ("not (exactly2OrAtLeast4 [1])", not (exactly2OrAtLeast4 [1]))
             , ("not (exactly2OrAtLeast4 [1,2,3])", not (exactly2OrAtLeast4 [1,2,3]))
             , ("not (exactly2OrAtLeast4 \"nem\")", not (exactly2OrAtLeast4 "nem"))
             , ("not (exactly2OrAtLeast4 \"a\")", not (exactly2OrAtLeast4 "a"))
             , ("firstTwoElements [1,2] == [1,2]", firstTwoElements [1,2] == [1,2])
             , ("firstTwoElements \"alma\" == \"al\"", firstTwoElements "alma" == "al")
             , ("firstTwoElements [20,9,8,7,6,5,4,3,2,1,0] == [20,9]", firstTwoElements [20,9,8,7,6,5,4,3
             , ("firstTwoElements [2,9,5,-3,-7,6,10,2,2,3,3] == [2,9]", firstTwoElements [2,9,5,-3,-7,6,1
             , ("firstTwoElements [10..] == [10,11]", firstTwoElements [10..] == [10,11])
             , ("firstTwoElements [1.5] == []", firstTwoElements [1.5] == [])
             , ("firstTwoElements [2] == []", firstTwoElements [2] == [])
             , ("firstTwoElements \"a\" == []", firstTwoElements "a" == [])
             , ("firstTwoElements ([] :: [Integer]) == []", firstTwoElements ([] :: [Integer]) == [])
             , ("withoutThird [1,2] == [1,2]", withoutThird [1,2] == [1,2])
             , ("withoutThird [1,2,3] == [1,2]", withoutThird [1,2,3] == [1,2])
             , ("withoutThird \"alma\" == \"ala\"", withoutThird "alma" == "ala")
             , ("withoutThird [20,9,8,7,6,5,4,3,2,1,0] == [20,9,7,6,5,4,3,2,1,0]", withoutThird [20,9,8,7
             , ("withoutThird [2,9,5,-3,-7,6,10,2,2,3,3] == [2,9,-3,-7,6,10,2,2,3,3]", withoutThird [2,9,
             , ("take 5 (withoutThird [10..]) == [10,11,13,14,15]", take 5 (withoutThird [10..]) == [10,1
             , ("withoutThird [1.5] == [1.5]", withoutThird [1.5] == [1.5])
             , ("withoutThird [2] == [2]", withoutThird [2] == [2])
             , ("withoutThird \"a\" == \"a\"", withoutThird "a" == "a")
             , ("withoutThird ([] :: [Int]) == []", withoutThird ([] :: [Int]) == [])
             , ("add (2 :: Int) (3 :: Integer) == (5 :: Integer)", add (2 :: Int) (3 :: Integer) == (5 ::
             , ("add (2 :: Int) (3 :: Int) == (5 :: Integer)", add (2 :: Int) (3 :: Int) == (5 :: Integer
```

```
        , ("add (2 :: Integer) (3 :: Int) == (5 :: Integer)", add (2 :: Integer) (3 :: Int) == (5 ::
        , ("add (2 :: Integer) (3 :: Integer) == (5 :: Integer)", add (2 :: Integer) (3 :: Integer)
        , ("onlySingletons [[],[1],[1,2],[1,2,3],[2],[2,3],[3]] == [[1],[2],[3]]", onlySingletons [[
        , ("onlySingletons [[1..],[],[1..10]] == []", onlySingletons [[1..],[],[1..10]] == [])
        , ("onlySingletons [\"a\",\"b\",\"c\",\"d\",\"e\",\"f\"] == [\"a\",\"b\",\"c\",\"d\",\"e\",\
        , ("decompress [] == \"\"", decompress [] == "")
        , ("decompress [('k',1),('a',1),('t',2)] == \"katt\"", decompress [('k',1),('a',1),('t',2)]
        , ("decompress [('A',3),('b',2),('k',5),('D',3)] == \"AAAbbkkkkkDDD\"", decompress [('A',3),
, not y
]
```