**Előzetes tudnivalók**

**Használható segédanyagok:**

* [Haskell könyvtárak dokumentációja](http://lambda.inf.elte.hu/haskell/doc/libraries/),
* [Hoogle](http://lambda.inf.elte.hu/haskell/hoogle/),
* [a tárgy honlapja](http://lambda.inf.elte.hu/), és a
* [Haskell szintaxis összefoglaló](http://lambda.inf.elte.hu/CheatSheet.xml).

**Ha bármilyen kérdés, észrevétel felmerül, azt a felügyelőknek kell jelezni, nem a diáktársaknak!**

**A feladatok tetszőleges sorrendben megoldhatóak. A pontozás szabályai a következők:**

* Minden teszten átmenő megoldás ér teljes pontszámot.
* Funkcionálisan hibás (valamelyik teszteseten megbukó) megoldás nem ér pontot.
* Fordítási hibás vagy hiányzó megoldás esetén a teljes megoldás 0 pontos. Ha hiányos/hibás részek lennének a feltöltött megoldásban, azok kommentben szerepeljenek.

*Tekintve, hogy a tesztesetek, bár odafigyelés mellett íródnak, nem fedik le minden esetben a függvény teljes működését, határozottan javasolt még külön próbálgatni a megoldásokat beadás előtt vagy megkérdezni a felügyelőket!*

**Feladatok**

**Ötös maradékok (1 pont)**

**Definiáljuk azt a függvényt, amely kiszámolja egy** *n* **egész szám öttel vett osztási maradékát!**

f5 :: Integral a => a -> a

f5 0 == 0

f5 1 == 1

f5 403 == 3

**Egyező elemek (1 pont)**

**Definiáljuk azt a függvényt, amely akkor ad vissza igaz értéket, ha a paraméterül kapott értékek közül legalább kettő megegyezik!**

matchingArgs :: Eq a => a -> a -> a -> Bool

matchingArgs 'a' 'b' 'a' == True

matchingArgs 1 1 1 == True

matchingArgs 'c' 'a' 'f' == False

matchingArgs 'i' 'b' 'b' == True

matchingArgs 'b' 'b' 'g' == True

**Osztás jobbról (1 pont)**

**Definiáljuk azt a függvényt, amely egy rendezett hármas elemeit elosztja a következő szerint: veszi a második és harmadik szám osztási maradékát és ezzel elosztja az első számot, majd az osztás egész részét adja meg! A függvény eredménye legyen Nothing, ha nullával osztanánk!**

division :: Integral a => (a, a, a) -> Maybe a

division (21, 31, 0) == Nothing

division (21, 30, 3) == Nothing

division (21, 31, 2) == Just 21

division (21, 31, 3) == Just 21

division (21, 31, 4) == Just 7

[division (x, y, z) | x <- [1,4,6], y <- [2,4,6,3], z<-[1,5]] == [Nothing,Just 0,Nothing,Just 0,Nothing,Just 1,Nothing,Just 0,Nothing,Just 2,Nothing,Just 1,Nothing,Just 4,Nothing,Just 1,Nothing,Just 3,Nothing,Just 1,Nothing,Just 6,Nothing,Just 2]

**Páros sorszámú elem-e (2 pont)**

**Adjuk meg azt a függvényt, amely egy elemről eldönti, hogy a listában páros indexű pozíción megtalálható-e! Az indexelés 1-től induljon.**

elemOnEvenIdx :: Eq a => a -> [a] -> Bool

elemOnEvenIdx 2 [] == False

elemOnEvenIdx 2 [1] == False

elemOnEvenIdx 1 [1] == False

elemOnEvenIdx 2 [1,2,3,4] == True

elemOnEvenIdx 2 [1,3,3,2] == True

elemOnEvenIdx 1 [0,1,3] == True

elemOnEvenIdx 5 [5,5,3] == True

elemOnEvenIdx 4 [4,5,3] == False

elemOnEvenIdx 5 (cycle [1,5]) == True

elemOnEvenIdx 7 (cycle [1..9]) == True

elemOnEvenIdx 7 (cycle [1,3..10]) == True

***n*-edik elemek törlése (2 pont)**

**Adjuk meg azt a függvényt, amely egy lista összes** *n***-edik elemét törli! A lista indexelését kezdjük egytől. Feltehetjük, hogy az** *n* **értéke pozitív.**

dropEveryNth :: Int -> [a] -> [a]

dropEveryNth 1 [1..10] == []

dropEveryNth 2 [1..10] == [1,3,5,7,9]

dropEveryNth 3 [1..10] == [1,2,4,5,7,8,10]

dropEveryNth 3 "Hello world!" == "Helowold"

dropEveryNth 3 "The quick brown fox jumps over the lazy dog" == "Th qic bow fx ums ve te az dg"

take 10 (dropEveryNth 3 [2,4..]) == [2,4,8,10,14,16,20,22,26,28]

**Szimmetrikus különbség (2 pont)**

**Adjuk meg két lista szimmetrikus különbségét! Az eredményben azon elemeknek kell szerepelnie, amelyek vagy az egyik, vagy a másik listában benne vannak, de egyszerre mindkettőben nem találhatóak meg. Feltehetjük, hogy a listák (egyenként tekintve) nem tartalmaznak ismétlődő elemeket!**

simDiff :: Eq a => [a] -> [a] -> [a]

simDiff [] [] == []

simDiff [] [3,2,1] == [3,2,1]

simDiff [1..5] [3..10] == [1,2,6,7,8,9,10]

simDiff [5,4..0] [3..10] == [2,1,0,6,7,8,9,10]

**Egész szám a szövegben (3 pont)**

**Adjuk meg azt a függvényt, amely egy decimális számot olvas be egy szövegből, amennyiben az lehetséges!**

**Egy számot akkor tekintünk beolvashatónak, ha:**

* legalább egy számjegye van,
* előjellel (-, + ) vagy számjeggyel kezdődik és az összes többi elem számjegy.

*Segítség:* **Használjuk a Data.Char modul függvényeit.**

parseNum :: String -> Maybe Integer

parseNum "" == Nothing

parseNum "+" == Nothing

parseNum "-" == Nothing

parseNum "-234" == Just (-234)

parseNum "+3423" == Just 3423

parseNum "342321" == Just 342321

parseNum "1+" == Nothing

parseNum "21231+12" == Nothing

parseNum "+almafa1" == Nothing

**Elem kiemelése (3 pont)**

**Definiáljuk azt a függvényt, amely egy adott elemet keres a listában és kiemeli azt a lista elejére (csak az első előfordulását)! Ha a lista nem tartalmazza a keresett elemet, adjuk vissza az eredeti listát változatlanul.**

**Ha szükséges, használjunk segédfüggvényt!**

elevate :: Eq a => a -> [a] -> [a]

elevate 1 [] == []

elevate 1 [1] == [1]

elevate 1 [3,1] == [1,3]

elevate 1 [1,2] == [1,2]

elevate 1 [2,3] == [2,3]

elevate 'f' "almafa" == "falmaa"

elevate 'e' "kecske" == "ekcske"

take 10 (elevate 123 [1..]) == [123,1,2,3,4,5,6,7,8,9]

**Lokális maximum (3 pont)**

**Készíts egy függvényt, ami függvényeket alkalmaz sorra a paraméterként megadott értéken és meghatározza az értékek első lokális maximumát! Azt a függvényértéket (f x) tekintjük lokális maximumnak, ahol a soron következő függvény értéke (g x) szigorúan kisebb az aktuális függvény értékénél, azaz g x < f x. A listát az elejétől kezdve keressük a maximumot.**

localMax :: Ord b => [(a -> b)]{- nem üres -} -> a -> b

localMax [(+3),(+1)] 0 == 3

localMax [(+3),(+4)] 0 == 4

localMax [(+1)] 0 == 1

localMax [(+1),(+2),(+3),(+1)] 0 == 3

localMax [(+1),(+2),(+1),(+1)] 0 == 2

localMax [(+3),(^2)] 0 == 3

localMax [(+3),(^2), (\*4), (^3)] 2 == 5

**Párok feldolgozása (2 pont)**

**Definiáljuk azt a függvényt, amely párok listáját és egy függvényt kapva alkalmazza a függvényt minden pár minden elemére!**

pairMap :: (a -> b) -> [(a,a)] -> [(b,b)]

pairMap (+1) [(1,2),(3,4)] == [(2,3),(4,5)]

pairMap (+100) (zip [1..5] [100..120]) == (zip [101..105] [200..220])

pairMap ("Hello " ++) [("general","Kenobi")] == [("Hello general","Hello Kenobi")]

**Kicsinyítő függvényalkalmazás (2 pont)**

**Adjuk meg azt a függvényt, amely egy függvényt alkalmaz egy lista minden elemére, de csak akkor, ha az az adott elem értékét csökkenti!**

applyIfReduces :: Ord a => (a -> a) -> [a] -> [a]

applyIfReduces (\*2) [1,2,-1,-2,3] == [1,2,-2,-4,3]

applyIfReduces abs [1,2,-1,-2,3] == [1,2,-1,-2,3]

applyIfReduces (\x -> x -1) [-5..5] == [-6,-5..4]

applyIfReduces not [True,False,True,False,False,False,True,False] == [False,False,False,False,False,False,False,False]

**Növények**

**Adatszerkezet definiálása (1 pont)**

**Definiáld a Plant adatszerkezetet, amelynek konstruktorai legyenek Flower és Tree. Mindkét konstruktor rendelkezzen egy String és egy Int paraméterrel, amelyben eltárolhatjuk az adott növény nevét és a napi vízigényét. Kérd az Eq és a Show típusosztályok automatikus példányosítását!**

**Túlélő növények (1 pont)**

**Definiáld a survive függvényt, amely paraméterül megkapja a növényeknek egy listáját, valamint az adott napi csapadékmennyiséget. A függvény adja vissza azoknak a virágoknak a nevét, amelyek a vízigényüknek megfelelő csapadékoz jutottak.**

survive :: [Plant] -> Int -> [String]

survive [] 100 == []

survive [Tree "Tölgyfa" 50 ] 150 == []

survive [Flower "Gyöngyvirág" 10] 20 == ["Gyöngyvirág"]

survive [(Tree "Alma fa" 60), (Tree "Körte fa" 40), (Flower "Ibolya" 5), (Flower "Hóvirág" 15 ), (Tree "Fenyőfa" 23), (Flower "Pénzecske" 3)] 10 == ["Ibolya", "Pénzecske"]

**Fák átlagos vízigénye (2 pont)**

**Definiáld a avgTreeWater függvényt, amely paraméterül megkapja a növényeknek egy listáját, és visszatér a listában lévő fák átlagos vízigényével.**

avgTreeWater :: [Plant] -> Maybe Double

avgTreeWater [] == Nothing

avgTreeWater [Flower "Rózsa" 9] == Nothing

avgTreeWater [Tree "Almafa" 47] == Just 47.0

avgTreeWater [(Tree "Alma fa" 60), (Tree "Körte fa" 40), (Flower "Ibolya" 5), (Flower "Hóvirág" 15 ), (Tree "Fenyőfa" 23), (Flower "Pénzecske" 3)] == Just 41.0

**Belső szavak tükrözése (2 pont)**

**Add meg azt a függvényt, amely egy szöveg első és utolsó szavain kívül az összes szó betűinek sorrendjét megfordítja!**

reverseWordsInside :: String -> String

reverseWordsInside "" == ""

reverseWordsInside "Haskell" == "Haskell"

reverseWordsInside "Haskell is" == "Haskell is"

reverseWordsInside "Haskell is good" == "Haskell si good"

reverseWordsInside "Haskell is a good language" == "Haskell si a doog language"

**Lista hatványozás (2 pont)**

**Add meg azt a függvényt, amely egy lista minden elemét hatványozza, ahol a kitevő a kettővel utána következő szám lesz! Az utolsó két elemet hagyd változatlanul!**

strangePow :: [Int] -> [Int]

strangePow [] == []

strangePow [6] == [6]

strangePow [6,6] == [6,6]

strangePow [2,2,2] == [4,2,2]

strangePow [1,2,3,4,5,6] == [1,16,243,4096,5,6]