### A vizsga nem idei, nem ennyi feladat lesz az idén, azonban a feladatok nehézségei mérvadóak, leginkább a közepétől a vége felé. Lehetnek már köztük ismerős feladatok, néhányat csinálhattunk a félévben.

---

# Előzetes tudnivalók

Használható segédanyagok:

- Haskell könyvtárak dokumentációja,
- Hoogle,
- a tárgy honlapja, és a
- Haskell szintaxis összefoglaló.

**Más segédeszköz nem használható.**

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

A feladatsor megoldására 90 perc áll rendelkezésre.

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

- Minden teszten átmenő, a feladat kikötéseinek megfelelő megoldás érhet teljes pontszámot.
- Funkcionálisan hibás (valamelyik teszteseten megbukó) megoldás nem ér pontot.
- Fordítási hibás kód esetén a teljes vizsga 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!*

---

# 1. Párok párja

Írd meg a `splitQuadruple :: (a,b,c,d) -> ((a,b),(c,d))` függvényt, mely egy négyest (4 elemű tuple) párok párjára képez úgy, hogy az első két elem kerüljön az első párba, és a második két elem kerüljön a második párba. Tartsd meg elemek eredeti sorrendjét.

```
splitQuadruple :: (a,b,c,d) -> ((a,b),(c,d))



splitQuadruple (1,2,3,4) == ((1,2), (3,4))
splitQuadruple ("a", "b", "c", "d") == (("a", "b"), ("c", "d"))
```

# 2. Számok távolsága

Add meg két tetszőleges szám távolságát (abszolútértékben vett különbségét) a számegyenesen.

```
dist1 :: Num a => a -> a -> a
```

```
dist1 1 3 == 2
dist1 3 1 == 2
dist1 1 1 == 0
dist1 1 (-1) == 2
dist1 (-1) 1 == 2
dist1 (-1) (-3) == 2
```

## 3. Kroenecker-delta

Implementáld a Kroenecker-delta függvényt, melynek értéke 1, ha paraméterei megegyeznek, 0, egyébként.

```
kroeneckerDelta :: Eq a => a -> a -> Int
```

```
kroeneckerDelta 0 0 == 1
kroeneckerDelta 0 1 == 0
kroeneckerDelta '#' '#' == 1
kroeneckerDelta '#' '@' == 0
kroeneckerDelta [1,2] [1,2] == 1
kroeneckerDelta [1,2] [3,4] == 0
```

## 4. Előfordulások

Számold meg hányszor fordul elő egy listában egy adott elem.

```
freq :: Eq a => a -> [a] -> Int
```

```
freq 'h' "hello world" == 1
freq 'o' "hello world" == 2
freq 'l' "hello world" == 3
freq 'x' "hello world" == 0
freq 5 [1..10]   == 1
freq 50 [1..10]   == 0
freq [1,2] [[0, 1, 2], [1,2], [2,1], [], [1,2]] == 2
```

## 5. Nagybetű

Döntsd el egy karakterláncról, hogy tartalmaz-e nagybetűt.

```
hasUpperCase :: String -> Bool
```

```
hasUpperCase "alma" == False
hasUpperCase "Alma_" == True
hasUpperCase "_amlA" == True
hasUpperCase "03 January" == True
hasUpperCase "" == False
```

## 6. Azonosító

Döntsd el egy karakterláncról, hogy azonosító-e, azaz teljesül-e rá, hogy csak kis- vagy nagybetűvel kezdődik és minden további eleme csak kisbetű, nagybetű, számjegy, vagy az alulvonás karakter.

```
identifier :: String -> Bool
```

```
identifier "alma" == True
identifier "a" == True
identifier "_" == False
identifier "9" == False
identifier "Alma_123" == True
identifier "Alma 123" == False
identifier "_amlA" == False
identifier "03 January" == False
identifier "" == False
```

## 7. Csere

Cseréld ki egy listának a megadott pozícióján levő elemét a paraméterben megadott elemre. Amennyiben a pozíció negatív, az elemet szúrd be a lista elejére. Amennyiben a pozíció legalább akkora, mint a lista elemszáma, az elemet a lista végére szúrd be. A listát 0-tól indexeljük.

```
replace :: Int -> a -> [a] -> [a]
```

```
replace 6 'W' "hello world" == "hello World"
replace 0 'H' "hello world" == "Hello world"
replace 2 9000 [1..4] == [1,2,9000,4]
replace (-10) '_' "hello world" == "_hello world"
replace (-10) '_' "_" == "__"
replace 9000 '*' "hello world" == "hello world*"
replace 0 'X' "" == "X"
```

## 8. Páros-páratlan

Döntsd el, hogy teljesül-e számok egy listájára, hogy a páros helyeken páros számok, a páratlan helyeken pedig páratlan számok állnak. A listákat 0-tól indexeljük.

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

```
paripos [2,9] == True
paripos [4,7,8,11,16] == True
paripos [2..50] == True
paripos [2..51] == True
paripos [1..50] == False
paripos [100,1,4,3,80,9] == True
paripos [100,1,4,3,80,10] == False
paripos [0] == True
paripos [2] == True
paripos [1] == False
paripos [] == True
```

## 9. Biztonságos osztás

Készíts függvényt, amely `Just` adatkonstruktorban megadja két egész szám lefele kerekített hányadosát, amennyiben a nevező nem 0. Amennyiben a nevező 0, a függvény értéke `Nothing`.

```
safeDiv :: Int -> Int -> Maybe Int
```

```
safeDiv 6 3 == Just 2
safeDiv 3 2 == Just 1
safeDiv 0 2 == Just 0
safeDiv 5 2 == Just 2
safeDiv 5 0 == Nothing
```

## 10. Elválasztójelek

Írj függvényt, ami egy szavakat pontosvesszőkkel elválasztva tartalmazó karakterláncból egy listába gyűjti a szavakat. Az üres sztringet is érvényes szónak tekintjük.

```
parseCSV :: String -> [String]
```

```
parseCSV "ELTE;2019" == ["ELTE","2019"]
parseCSV "ELTE;IK;2019" == ["ELTE","IK","2019"]
parseCSV "" == [""]
parseCSV ";" == ["",""]
parseCSV ";;" == ["","",""]
parseCSV "ELTE;IK;2019;" == ["ELTE","IK","2019",""]
parseCSV ";ELTE;IK;2019;" == ["","ELTE","IK","2019",""]
```

## 11. C kombinátor

Készíts magasabbrendű függvényt, mely "megcseréli" egy két paraméteres függvény argumentumait.

```
c :: (a -> b -> c) -> (b -> a -> c)
```

```
c (-) 3 4 == 1
c (-) 6 10 == 4
c div 4 12 == 3
c (++) " world" "hello" == "hello world"
c (c (-)) 10 6 == 4
c (c div) 12 4 == 3
c (c (++)) "hello" " world" == "hello world"
```

## 12. Kivéve, ha ...

Válaszd ki egy lista azon elemeit, melyekre teljesül az első paraméterben kapott feltétel, de nem teljesül a második paraméterben kapott feltétel.

```
selectUnless :: (t -> Bool) -> (t -> Bool) -> [t] -> [t]
```

```
selectUnless (>= 2) (==2) [1,2,3,4] == [3,4]
selectUnless even odd [1..50] == [2,4..50]
selectUnless ((<= 2) . length) null ["", "n", "xo", "", "alma"] == ["n", "xo"]
selectUnless ((0==) . (`mod` 2)) ((0/=) . (`mod` 3)) [1..50] == [6,12,18,24,30,36,42,48]
```

## 13. W kombinátor

Készíts magasabbrendű függvényt, mely általánosítja a négyzetre emelést: egy függvényt és egy értéket vár paraméterül és úgy alkalmazza a függvényt, hogy mindkét argumentuma az érték legyen.

```
w :: (a -> a -> a) -> a -> a
```

```
w (*) 2 == 4
w (*) 3 == 9
w (*) 5 == 25
w (+) 3 == 6
w (+) 5 == 10
w (++) "x" == "xx"
w (++) "ba" == "baba"
w (++) [] == ([] :: [Int])
```

## 14. Iteratív alkalmazás

Készítsd el az `ntimes` magasabbrendű függvényt, amely iteratívan alkalmaz egy függvényt egy értékre az eddigi számítások eredményének felhasználásával: `x `f` x `f` ... `f` x `f` x`, ahol `x` a megadott számú alkalommal fordul elő.

Az elkészített függvény valójában a hatványozás általánosítása, ahol a paraméterek:

- egy két paraméteres művelet (hatványozás esetén a szorzás),
- egy tetszőleges elem (a hatványozás alapja), és
- egy pozitív (nem 0) egész szám (a hatványozás kitevője).

Amennyiben a kitevő 1, az eredmény legyen a megadott elem. Feltehetjük, hogy a függvényt csak asszociatív műveletekkel paraméterezzük fel.

```
ntimes :: (a -> a -> a) -> a -> Int -> a
```

```
ntimes (*) 2 1 == (2  :: Int)
ntimes (*) 2 2 == (4  :: Int)
ntimes (*) 2 3 == (8  :: Int)
ntimes (*) 3 3 == (27 :: Int)
ntimes (+) 2 4 == (8  :: Int)
ntimes (+) 3 8 == (24 :: Int)
ntimes (++) "x" 5 == "xxxxx"
ntimes (++) "la" 5 == "lalalalala"
ntimes (++) "" 5 == ""
```

## 15. Binárisok I.

Definiáld a `Binary` adattípust, melynek két paraméternélküli adatkonstruktora van: `On` és `Off`. Írj `deriving (Eq,Show)` záradékot hozzá!

Definiálj függvényt, amely az `On` értéket `Off`, az `Off` értéket `On` értékre állítja

```
switch :: Binary -> Binary
```

```
switch (switch On) == On
switch (switch (switch Off)) == On
```

## 16. Binárisok II.

Definiáld a `bxor :: [Binary] -> [Binary] -> [Binary]` függvényt, amely visszaad egy listát, amely az `i`-edik pozíción `On` értéket tartalmaz, ha az `i`-edik pozíción mindkét paraméterben kapott listában egyaránt `On` vagy egyaránt `Off` érték szerepel.

Amennyiben adott pozíción különböző értékeket tartalmaz a két lista, az eredménylistában is adjunk vissza különböző értékeket. Feltehetjük, hogy a listák egyenlő hosszúak.

```
bxor :: [Binary] -> [Binary] -> [Binary]
```

```
bxor [On] [On]             == [On]
bxor [Off] [Off]           == [On]
bxor [On] [Off]            == [Off]
bxor [Off] [On]            == [Off]
bxor [On, Off] [On, Off]   == [On, On]
```

```
bxor [Off, Off] [Off, Off] == [On, On]
bxor [On, On] [On, On]     == [On, On]
bxor [Off, On] [Off, On]   == [On, On]
bxor [On, Off] [Off, Off]  == [Off, On]
bxor [On, Off, On, Off] [Off, On, Off, On] == [Off, Off, Off, Off]
bxor [] [] == []
```

## Tesztek

```
splitQuadruple (1,2,3,4) == ((1,2), (3,4))
splitQuadruple ("a", "b", "c", "d") == (("a", "b"), ("c", "d"))
dist1 1 3 == 2
dist1 3 1 == 2
dist1 1 1 == 0
dist1 1 (-1) == 2
dist1 (-1) 1 == 2
dist1 (-1) (-3) == 2
kroeneckerDelta 0 0 == 1
kroeneckerDelta 0 1 == 0
kroeneckerDelta '#' '#' == 1
kroeneckerDelta '#' '@' == 0
kroeneckerDelta [1,2] [1,2] == 1
kroeneckerDelta [1,2] [3,4] == 0
freq 'h' "hello world" == 1
freq 'o' "hello world" == 2
freq 'l' "hello world" == 3
freq 'x' "hello world" == 0
freq 5 [1..10]   == 1
freq 50 [1..10]   == 0
freq [1,2] [[0, 1, 2], [1,2], [2,1], [], [1,2]] == 2
hasUpperCase "alma" == False
hasUpperCase "Alma_" == True
hasUpperCase "_amlA" == True
hasUpperCase "03 January" == True
hasUpperCase "" == False
identifier "alma" == True
identifier "a" == True
identifier "_" == False
identifier "9" == False
identifier "Alma_123" == True
identifier "Alma 123" == False
identifier "_amlA" == False
identifier "03 January" == False
identifier "" == False
replace 6 'W' "hello world" == "hello World"
replace 0 'H' "hello world" == "Hello world"
replace 2 9000 [1..4] == [1,2,9000,4]
replace (-10) '_' "hello world" == "_hello world"
replace (-10) '_' "_" == "__"
replace 9000 '*' "hello world" == "hello world*"
replace 0 'X' "" == "X"
paripos [2,9] == True
paripos [4,7,8,11,16] == True
paripos [2..50] == True
paripos [2..51] == True
```

```
paripos [1..50] == False
paripos [100,1,4,3,80,9] == True
paripos [100,1,4,3,80,10] == False
paripos [0] == True
paripos [2] == True
paripos [1] == False
paripos [] == True
safeDiv 6 3 == Just 2
safeDiv 3 2 == Just 1
safeDiv 0 2 == Just 0
safeDiv 5 2 == Just 2
safeDiv 5 0 == Nothing
parseCSV "ELTE;2019" == ["ELTE","2019"]
parseCSV "ELTE;IK;2019" == ["ELTE","IK","2019"]
parseCSV "" == [""]
parseCSV ";" == ["",""]
parseCSV ";;" == ["","",""]
parseCSV "ELTE;IK;2019;" == ["ELTE","IK","2019",""]
parseCSV ";ELTE;IK;2019;" == ["","ELTE","IK","2019",""]
c (-) 3 4 == 1
c (-) 6 10 == 4
c div 4 12 == 3
c (++) " world" "hello" == "hello world"
c (c (-)) 10 6 == 4
c (c div) 12 4 == 3
c (c (++)) "hello" " world" == "hello world"
selectUnless (>= 2) (==2) [1,2,3,4] == [3,4]
selectUnless even odd [1..50] == [2,4..50]
selectUnless ((<= 2) . length) null ["", "n", "xo", "", "alma"] == ["n", "xo"]
selectUnless ((0==) . (`mod` 2)) ((0/=) . (`mod` 3)) [1..50] == [6,12,18,24,30,36,42,48]
w (*) 2 == 4
w (*) 3 == 9
w (*) 5 == 25
w (+) 3 == 6
w (+) 5 == 10
w (++) "x" == "xx"
w (++) "ba" == "baba"
w (++) [] == ([] :: [Int])
ntimes (*) 2 1 == (2  :: Int)
ntimes (*) 2 2 == (4  :: Int)
ntimes (*) 2 3 == (8  :: Int)
ntimes (*) 3 3 == (27 :: Int)
ntimes (+) 2 4 == (8  :: Int)
ntimes (+) 3 8 == (24 :: Int)
ntimes (++) "x" 5 == "xxxxx"
ntimes (++) "la" 5 == "lalalalala"
ntimes (++) "" 5 == ""
switch (switch On) == On
switch (switch (switch Off)) == On
bxor [On] [On]             == [On]
bxor [Off] [Off]           == [On]
bxor [On] [Off]            == [Off]
bxor [Off] [On]            == [Off]
bxor [On, Off] [On, Off]   == [On, On]
bxor [Off, Off] [Off, Off] == [On, On]
bxor [On, On] [On, On]     == [On, On]
bxor [Off, On] [Off, On]   == [On, On]
bxor [On, Off] [Off, Off]  == [Off, On]
```

```
bxor [On, Off, On, Off] [Off, On, Off, On] == [Off, Off, Off, Off]
bxor [] [] == []
```