Funkcionális programozás 2. gyakorlat

Szám típusok, típusosztályok hierarchiája

```
Num (+,-,*)
    Fractional (/)
        Double
        Float
    Integral (div, mod)
        Int
        Integer
```

Feladat: Mi lehet a típusa az alábbi kifejezéseknek?

```
- 1
- 2 * 16 - 5
- 2 * 16 - (5 :: Int)
- True + True
- 16 / 5 + 18
- div 16 2 * 5
- div 16 2 / 5
```

Számok konverziója

```
fromIntegral :: (Integral a, Num b) => a -> b
truncate :: (RealFrac a, Integral b) => a -> b
round :: (RealFrac a, Integral b) => a -> b
ceiling :: (RealFrac a, Integral b) => a -> b
floor :: (RealFrac a, Integral b) => a -> b
```

Pl.:

```
fromIntegral (10 `div` 2) * pi
(round pi) `mod` 2
```

Mintaillesztés/pattern matching

- Több függvényalternatíva
- Függvényparaméterek helyén minták
- Minta lehet:
  - Konstruktor (pl.: konkrét érték, mint: 0,1.5,42,True,False,‘a’,‘b’,“alma” stb. egyéb példa majd később)
  - “változó” (pl.: eddigi függvények esetén)
  - joker (_ karakter)

Pl.:

```
isZero :: Int -> Bool
```

```
isZero 0 = True
isZero _ = False
```

Feladat:

a) Ha az `isZero` függvényt a `10` paraméterrel hívjuk mi fog történni?
b) Mi történne, ha a fentebb definiált `isZero` függvényben felcserélnénk a két függvényalternatívát?

c) Mi történne abban az esetben, ha hozzáadnánk a következő harmadik függvényalternatívát: `isZero 10 = True`?

Definiáld a `singleDigitPrime` függvényt, amely eldönti egy paraméterként kapott egész számról, hogy egy számjegyű prím-e!

```
oneDigitPrime :: Int -> Bool
```

Feladat: Definiáld újra a logikai "és" függvényt!

```
and' :: Bool -> Bool -> Bool
```

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

```
and' True  True  == True
and' True  False == False
and' False True  == False
and' False False == False
```

Feladat: Definiáld újra a logikai "vagy" függvényt!

```
or' :: Bool -> Bool -> Bool
```

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

```
or' True  True  == True
or' True  False == True
or' False True  == True
or' False False == False
```

Feladat: Definiáld újra a logikai "xor" függvényt!

```
xor' :: Bool -> Bool -> Bool
```

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

```
xor' True  True  == False
xor' True  False == True
xor' False True  == True
```

```
xor' False False == False
```

Feladat: Definiáljunk egy függvényt, amely egy sortörést egy szóközre cserél!

```
replaceNewline :: Char -> Char
```

Rendezett n-es/Tuple

```
- Összetett adattípus, komponensekből áll
- Bármilyen típusú értéket képes tárolni

tuple1 :: (Int, Int)
tuple1 = (2, 3)

tuple2 :: (Int, Char)
tuple2 = (2, 'c')

tuple3 :: (Int, Char, Bool)
tuple3 = (2, 'c', True)
```

Feladat: Definiáld az `isEvenTuple` függvényt, amely egy egész számot vár paraméterül és visszatérési értéke egy rendezett pár, amelynek első eleme maga a szám és második eleme `True`, ha páros és `False`, ha páratlan a szám!

```
isEvenTuple :: Int -> (Int, Bool)
```

Pl.:

```
isEvenTuple 2 == (2,True)
isEvenTuple 3 == (3,False)
```

Feladat: Definiálj függvényt, amely eldönti egy koordinátáról, hogy az origó-e!

```
isOrigo :: (Double, Double) -> Bool
```

Pl.:

```
isOrigo (0,0) == True
isOrigo (1,0) == False
```

Feladat: Definiálj függvényt, amely eldönti egy koordinátáról, hogy az x-tengelyen van-e!

```
isOnXAxis :: (Double, Double) -> Bool
```

Pl.:

```
isOnXAxis (0,0) == True
isOnXAxis (0,5) == True
isOnXAxis (1,0) == False
```

Feladat: Definiáld törtek szorzását!

```
mul :: (Int, Int) -> (Int, Int) -> (Int, Int)

mul (1, 2) (1, 2) == (1, 4)
mul (4, 3) (6, 5) == (24, 15)
```