Számok

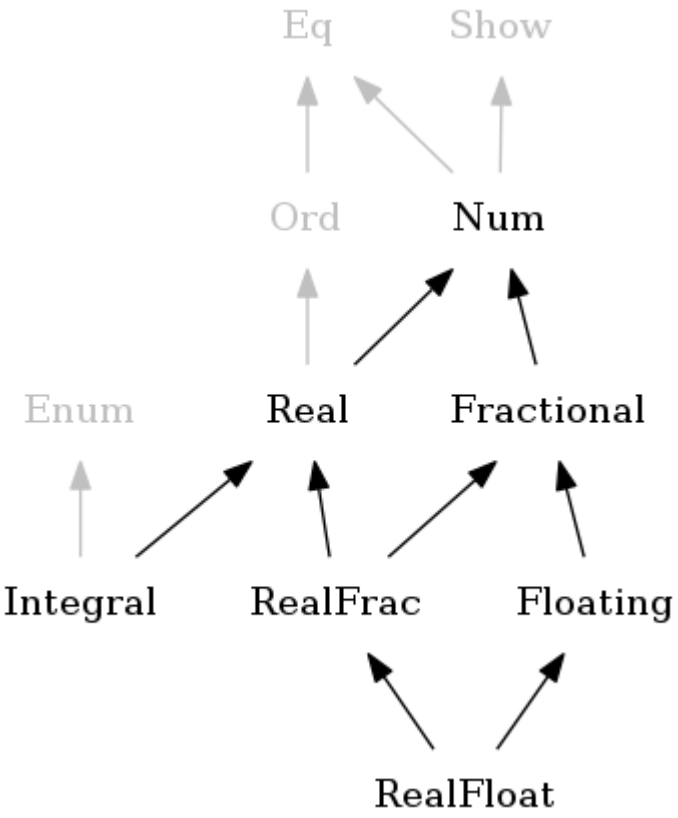
Típusok

```
Int :: *          -- korlátos egész szám (32 bites platformon –2147483648..2147483647)
Integer :: *      -- egész szám
Rational :: *     -- racionális szám
Float :: *        -- egyszeres pontosságú lebegőpontos szám
Double :: *       -- dupla pontosságú lebegőpontos szám
```

Típusosztályok

```
Integral   =  {Int, Integer}
Num        =  {Int, Integer, Rational, Float, Double}
Real       =  {Int, Integer, Rational, Float, Double}
Fractional =  {Rational, Float, Double}
RealFrac   =  {Rational, Float, Double}
Floating   =  {Float, Double}
RealFloat  =  {Float, Double}
```

Típusosztály hierarchia



Számliterálok

115 :: Num p => p -- decimális
0xAB :: Num p => p -- hexadecimális
0o776 :: Num p => p -- oktális
3.14 :: Fractional p => p -- tizedestört
2.2e-10 :: Fractional p => p -- tudományos jelölésmód

Konstansok

pi :: Floating a => a -- π = 3.14..

Konverziók

fromIntegral :: (Integral a, Num b) => a -> b -- egészből számba konvertálás
realToFrac :: (Real a, Fractional b) => a -> b -- valósból törtbe konvertálás

Kerekítések

truncate :: (RealFrac a, Integral b) => a -> b -- nulla felé kerekítés
round :: (RealFrac a, Integral b) => a -> b -- legközelebbihez kerekítés
ceiling :: (RealFrac a, Integral b) => a -> b -- felfele kerekítés
floor :: (RealFrac a, Integral b) => a -> b -- lefele kerekítés

Operátorok

infixr 8 ^, ^^, **
infixl 7 *, /, `rem`, `mod`, `div`, `quot`
infixl 6 -, +, `negate

(+) :: Num a => a -> a -> a -- összeadás
(*) :: Num a => a -> a -> a -- szorzás
(-) :: Num a => a -> a -> a -- kivonás
negate :: Num a => a -> a -- negálás, '-' jellel is jelölhető
(/) :: Fractional a => a -> a -> a -- osztás
(^) :: (Integral b, Num a) => a -> b -> a -- pozitív egész kitevős hatványozás
(^^) :: (Fractional a, Integral b) => a -> b -> a -- egész kitevős hatványozás
(**) :: Floating a => a -> a -> a -- valós kitevős hatványozás

Függvények

abs :: Num a => a -> a -- abszolút érték
sqrt :: Floating a => a -> a -- négyzetgyök
log :: Floating a => a -> a -- e alapú logaritmus
exp :: Floating a => a -> a -- e hatványára emelés
sin :: Floating a => a -> a -- szinusz

```
cos ::   Floating a => a -> a          -- koszinusz
tan ::   Floating a => a -> a          -- tangens
asin :: Floating a => a -> a           -- arkuszszinusz
acos :: Floating a => a -> a           -- arkuszkoszinusz
atan :: Floating a => a -> a           -- arkusztangens
sinh :: Floating a => a -> a           -- hiperbolikus szinusz
cosh :: Floating a => a -> a           -- hiperbolikus koszinusz
tanh :: Floating a => a -> a           -- hiperbolikus tangens
asinh :: Floating a => a -> a          -- hiperbolikus arkuszszinusz
acosh :: Floating a => a -> a          -- hiperbolikus arkuszkoszinusz
atanh :: Floating a => a -> a          -- hiperbolikus arkusztangens
quot :: Integral a => a -> a -> a      -- maradékos osztás (multiplikatív)
div ::   Integral a => a -> a -> a     -- maradékos osztás (additív)
rem ::   Integral a => a -> a -> a     -- maradékképzés (multiplikatív)
mod ::  Integral a => a -> a -> a      -- maradékképzés (additív)
gcd ::   Integral a => a -> a -> a     -- legnagyobb közös osztó
```

Logikai érték

Típusok

```
Bool :: *                              -- logikai érték
Ordering :: *                          -- összehasonlítás eredménye
```

Típusosztályok

```
Eq   = {Int, Double, Char, ...}        -- egyenlőségvizsgálat
Ord  = {Int, Double, Char, ...}        -- összehasonlítás
```

Konstansok

```
True :: Bool                           -- igaz
otherwise :: Bool                      -- ugyanaz mint True
False :: Bool                          -- hamis
GT :: Ordering                                 -- nagyobb
LT :: Ordering                         -- kisebb
EQ :: Ordering                                 -- egyenlő
```

Logikai összekötők

```
(&&) :: Bool -> Bool -> Bool           -- logikai ÉS
(||) :: Bool -> Bool -> Bool           -- logikai VAGY
not :: Bool -> Bool                    -- tagadás
```

Operátorok

infix 4 ==, /=, <, >, <=, >=
infixr 3 &&
infixr 2 ||

(==) :: Eq a => a -> a -> Bool -- egyenlő-e
(/=) :: Eq a => a -> a -> Bool -- nem egyenlő-e
(<) :: Ord a => a -> a -> Bool -- kisebb-e
(>) :: Ord a => a -> a -> Bool -- nagyobb-e
(<=) :: Ord a => a -> a -> Bool -- kisebb vagy egyenlő-e
(>=) :: Ord a => a -> a -> Bool -- nagyobb vagy egyenlő-e

Függvények

compare :: Ord a => a -> a -> Ordering -- összehasonlítás
even :: Integral a => a -> Bool -- páros-e
odd :: Integral a => a -> Bool -- páratlan-e
min :: Ord a => a -> a -> a -- két érték minimuma
max :: Ord a => a -> a -> a

N-esek

Típusok

(,) :: * -> * -> * -- pár típuskonstruktor, használat: (,) Int Char vagy (Int,Char)
(,,) :: * -> * -> * -> * -- hármas típuskonstruktor
(,,,) :: * -> * -> * -> * -> * -- négyes típuskonstruktor

Függvények

(,) :: a -> b -> (a, b) -- pár konstruktor, használat: (,) 1 2 vagy (1,2)
(,,) :: a -> b -> c -> (a, b, c) -- hármas konstruktor, használat: (,,) 1 2 3 vagy (1,2,3)
(,,,) :: a -> b -> c -> d -> (a, b, c, d) -- négyes konstruktor, használat: (,,,) 1 2 3 4 vagy (1,2,3,4)
fst :: (a, b) -> a -- pár első eleme (first)
snd :: (a, b) -> b -- pár második eleme (second)

```haskell
not :: Bool -> Bool
not True  = False
not False = True

&& :: Bool -> Bool -> Bool
True  && x     = x
_     &&  _     = False
```

```haskell
||  :: Bool -> Bool -> Bool
False  || x     = x
_      ||  _     = True

fst:: (a,b) -> a
fst (a,b) = a

snd:: (a,b) -> a
snd (a,b) = b

even :: Integer -> Bool
even x = mod x 2 == 0

odd :: Integer -> Bool
odd x = mod x 2 == 1

null :: [a] -> Bool
null [] = True
null _  = False

head :: [a] -> a
head (x:xs) = x
head [] = error "head of []"

tail :: [a] -> [a]
tail (x:xs) = xs
tail [ ]= error "tail of []"

last :: [a] -> a
last [a] = a
last (x:xs) = last xs
last [] = error "last of []"



init :: [a] -> [a]
init [] = error "empty"
init [a] = []
init (x:xs) = x : init xs

take :: Int -> [a] -> [a]
take _ [] = []
take n xs | n<=0 = []
take n (x:xs) = x: take (n-1) xs

drop :: Int -> [a] -> [a]
drop _ [] = []
drop n xs | n<=0 = (xs)
drop n (x:xs) = drop (n-1) xs


sum :: Num a => [a] -> a
sum [] = 0
sum (x:xs) = x + sum xs

length :: [a] -> Int
length [] = 0
length (_:xs) = 1 + length xs

zip :: [a] -> [b] -> [(a,b)]
zip (x:xs) (y:ys) = (x,y): zip xs ys
```

```haskell
zip _ _ = []

min :: a -> a -> a
min x y
    | x <= y    =  x
    | otherwise =  y

max :: a -> a -> a
max x y
    | x >= y    =  x
    | otherwise =  y

minimum :: [a] -> a
minimum [a] = a
minimum (x:xs) = min x (minimum xs)

maximum :: [a] -> a
maximum [a] = a
maximum (x:xs) = max x (maximum xs)

concat :: [[a]] -> [a]
concat [] = []
concat (x:xs) = x ++ concat xs

(++) :: [a] -> [a] -> [a]
[] ++ ys = ys
(x:xs) ++ ys = x: ( xs ++ ys)

isPrefixOf :: Eq a => [a] -> [a] -> Bool
isPrefixOf [] _ = True
isPrefixOf _ [] = False
isPrefixOf (x:xs) (y:ys) = x==y && isPrefixOf xs ys

elem :: Eq a => a -> [a]{-véges-} -> Bool
elem _ [] = False
elem a (x:xs) = a == x || elem a xs

(^) :: Num a => a -> Integer -> a
x ^ 0 = 1
x ^ n | odd n = x *( x ^ (n-1))
x ^ n = (x ^ (div n 2)) *  (x ^ (div n 2))
```