Primitiivtüübid

- Täisarvud: `Int`, `Integer`

```
15   :: Int   -18   :: Integer   0014 :: Integer
0o17 :: Int   -0x12 :: Integer   0XC  :: Integer
```

- Ujukomaarvud: `Float`, `Double`

```
3.14 :: Float  -7.3e5 :: Float  2.5E-3 :: Double
```

- Aritmeetilisi funktsioone:

```
negate, abs     :: Num a => a -> a
(+), (-), (*)   :: Num a => a -> a -> a
div, mod        :: Integral a => a -> a -> a
(/)             :: Fractional a => a -> a -> a
```


Primitiivtüübid

- Tõeväärtused: `Bool`

```
False :: Bool     True :: Bool
```

- Eeldefineeritud funktsioone tõeväärtustega:

```
(&&), (||)   :: Bool -> Bool -> Bool
not          :: Bool -> Bool
otherwise    :: Bool
(==), (/=)   :: Eq a => a -> a -> Bool
(<),  (<=)   :: Eq a => a -> a -> Bool
(>),  (>=)   :: Eq a => a -> a -> Bool
```

Primitiivtüübid

- Sümbolid: `Char`

```
'A'  '\32' '\x20' '\o40' '\n'  '\\'  '\BEL'
```

- Eeldefineeritud funktsioone sümbolitega:

```
isSpace, isDigit, isAlpha :: Char -> Bool
isUpper, isLower          :: Char -> Bool
toUpper, toLower          :: Char -> Char
ord :: Char -> Int
chr :: Int  -> Char
```

- NB! Peaksid asuma standardteegis `Char`

Eeldefineeritud tüübikonstruktorid

- Ennikud: `(t1, ... , tn)`

```
(1, fact2, 'A')        :: (Int, Int -> Int, Char)
((True, "Hello"), ()) :: ((Bool, String), ())
```

- Eeldefineeritud funktsioone paaridega:

```
fst :: (a,b) -> a
snd :: (a,b) -> b
```

- `fst`, `snd` on *polümorfsed* funktsioonid
- Ennikute dekonstrueerimine *näidiste sobitamisega*

```
fst (x,y) = x
```

Eeldefineeritud tüübikonstruktorid

- Funktsioonid `t1 -> t2`

```
fact2              :: Int -> Int
\ x y -> x + 5 * y :: Int -> Int -> Int
```

  - Nool on parem-, aplikatsioon vasakassotsiatiivne!

- Kõik funktsioonid on üheargumendilised!
- Mitmeargumendilisi funktsioone saab modelleerida
  - “pakkides” argumendid ühte ennikusse
  - kasutades “curried” funktsioone

Mitmeargumendilised funktsioonid

- Näide — gravitatsioonijõud kahe keha vahel

```
g = 6.67e-11

gravity (m1,r,m2)        -- Non-curried
     | r == 0.0  = 0.0
     | otherwise = (g*m1*m2)/(r*r)

gravity_c m1 r m2        -- Curried
     | r == 0.0  = 0.0
     | otherwise = (g*m1*m2)/(r*r)
```

Mitmeargumendilised funktsioonid

- "Curried" funktsioone saab osaliselt rakendada

```
mass_of_earth   = 5.96e24
radius_of_earth = 6.37e6
my_mass         = 80.0

earths_force    = gravity_c mass_of_earth

surface_force   = earths_force radius_of_earth

my_force        = surface_force my_mass
```


Operaatorid

- Erisümbolitest koosnevad identifikaatorid on infiks-operaatorid `!#$%&*+./<=>?@\^|-~:`
- Näide:

```
x ^ 0     = 1
x ^ (n+1) = x * x^n
```

- Vasak-, parem- ja mitteasotsiatiivsed operaatorid:

```
infixl 6 +    infixr 8 ^    infix 4 ==
```

- Lubatud prioriteedid 0–9 (aplikatsiooni prioriteet 10)
- Vaikimisi vasakassotsiatiivne, prioriteediga 9

Operaatorid

- Osaliselt rakendatud operaatorid — sektsioonid

```
(+1)      (/2)      (1/)      (==0)
```

- Prefiks-kujul operaatorid

```
(+) 1 3
```

- Operaator peab olema prefiks-kujul
  - tüübideklaratsioonis
  - teise funktsiooni argumendiks olles
- Binaarseid funktsioone saab kasutada infiks-kujul

```
3 `max` 5
```


Eeldefineeritud tüübikonstruktorid

- Listid: `[t]`

```
[1,2,3,4,5,6]       :: [Int]
[[1,2],[3],[4,5,6]] :: [[Int]]
```

- Stringid: `String`

```
"Hello!\n"
```

- NB! String on sümbolite list `[Char]`

```
['H', 'e', 'l', 'l', 'o', '!', '\n']
```

Listide konstrueerimine

- Listide konstrueerimise primitiivid:
  - tühilist (*nil*)

    ```
    []  :: [a]
    ```
  - listi konstruktor (*cons*)

    ```
    infixr 5 :
    (:) :: a -> [a] -> [a]
    ```
  - kõik listid on ehitatud nil ja cons abil

    ```
    Hugs> 1 : 2 : 3 : 4 : []
    [1,2,3,4]
    ```
  - NB! Mida vastab Hugs ??

    ```
    Hugs> []
    ```


Funktsioonid listidel

- Funktsioonide defineerimine näidiste sobitamisega

```
head :: [a] -> a
head (x:xs)  = x

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

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

Funktsioonid listidel

- Rekursiivselt defineeritud funtsioone

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

infixr 5  ++
(++) :: [a] -> [a] -> [a]
[]     ++ ys = ys
(x:xs) ++ ys = x : (xs ++ ys)
```


Funktsioonid listidel

- Rekursiivselt defineeritud funtsioone

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

-- infixl 9 !!
(!!) :: [a] -> Int -> a
(x:_)  !! 0         =  x
(_:xs) !! n | n > 0 =  xs !! (n-1)
```

Funktsioonid listidel

- Rekursiivselt defineeritud funtsioone

```
take, drop    :: Int -> [a] -> [a]
take 0 xs     = []
take n []     = []
take n (x:xs) = x : take (n-1) xs

drop 0 xs     = xs
drop n []     = []
drop n (x:xs) = drop (n-1) xs
```


Funktsioonid listidel

- Listide ümberpööramine

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

- Sabarekursiivne versioon

```
reverse xs = rev [] xs
    where rex ys []     = ys
          rev ys (x:xs) = rev (x:ys) xs
```

Listide konstrueerimine

- "Aritmeetilised" jadad:

```
Hugs> [1..5]
[1,2,3,4,5]
Hugs> [2,5..12]
[2,5,8,11]
Hugs> [3..1]
[]
Hugs> [1..]
[1,2,3,4,5,6,7, .......
Hugs> ['A'..'Z']
"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
```


Listide konstrueerimine

- Listide ZF-esitus (list comprehension)

```
Hugs> [x * x | x <- [1..10], even x]
[4,16,36,64,100]
Hugs> [i | (i,c) <- zip [1..] "AbCD", isUpper c]
[1,3,4]
Hugs> [(i,j) | i <- [1..4], j <- [i+1..4]]
[(1,2),(1,3),(1,4),(2,3),(2,4),(3,4)]
```

- Listide "lamendamine" uuesti

```
concat :: [[a]] -> [a]
concat xss = [ x | xs <- xss, x <- xs]
```