Haskell Workshop

FINN.no

FINN.no

Haskell

- Haskell / GHC
- Purely functional
- Statically Typed
- Lazy

Part 1

create a function

```
isPrime :: Int -> Bool
isPrime 0 = False
isPrime 1 = False
isPrime 2 = True
isPrime 3 = True
isPrime 4 = False
isPrime 5 = True
isPrime 6 = False
isPrime 7 = True
```

create a function 2

```
myFunction :: Int -> Int -> Int
myFunction    a      b   =  (a + 1) * b ^ 2
```

Applying arguments

```
> myFunction 1 2
8
```

Curried signatures

```
myFunction :: Int -> Int -> Int
```

```
myFunction :: Int -> (Int -> Int)
```

```
>:t myFunction 1
myFunction 1 :: Int -> Int
```

Everything is an expression

```
isNine :: Int -> Bool
isNine i = if i == 9
  then True
  else False
```

You must have an else

let .. in expressions

```
cylVolume :: Float -> Float -> Float
cylVolume diam h =
  let rad = diam / 2
      area = pi * rad^2
  in area * h
```

lists

```
listOfInts :: [Int]
listOfInts  = [1,2,3]

concat :: [a] -> [a] -> [a]
concat as bs = as ++ bs
```

REPL

```
$ stack repl ./src/Part1.hs
(...)
*Part1>
```

Unloading and Loading

```
*Loaded Modules> :l
Ok, no modules loaded.
```

```
Prelude>

Prelude> :l Part1
Ok, one module loaded.
*Part1>
```

Evaluating values and types

```
>1 + 1
2

>:t 1
1 + 1 :: Num a => a
```

Part 2 - 5

Recursion on lists

```haskell
uppercase :: [Char] -> [Char]
uppercase [] = []
uppercase (x:xs) = toUpper x :: uppercase xs
```

x is a Char but xs is a [Char]

Functions over lists

```haskell
inc :: [Int] -> [Int]
inc numbers = map (+1) numbers

inc2 :: [Int] -> [Int]
inc2 numbers = map (\x -> x + 1) numbers
```

Functions over lists 2

```haskell
firstFive :: [a] -> [a]
firstFive as = take 5 as

onlyEven :: [Int] -> [Int]
onlyEven xs = filter even xs
```

Part 6

Creating a type

```haskell
data TrafficLight = Red | Yellow | Green

safe :: TrafficLight -> Bool
safe Red    = False
safe Yellow = False
safe Green  = True
```

What to do

```
git clone https://github.com/mariatsji/haskell-workshop
open README.md
```