Agenda

- Kjapt om syntax med eksempler og ghci-tour mm
- Lite, men ikke-trivielt eksempel (factor)
- Lite, men ikke-trivielt eksempen (calc)
- Kode og annet finnes på
  `git://github.com/kaaveland/fgg.git`

Hello, World

```haskell
-- Dette er en kommentar
module Main -- module Main blir en executable
        (main) -- eksporterte symboler her
where -- Definisjoner
  main :: IO () -- Frivillig type-deklarasjon
  main = putStrLn "Hello, World!"
```

```
$ ghc hw.hs -o hw
[1 of 1] Compiling Main                ( hw.hs, hw.o )
Linking hw ...
$ ./hw
Hello, World!
```

ghci

Note to self: kjapp demo (ting som er forskjellig i ghci)

Typen til x :t x

Informasjon om ting :i ting

Last inn fil.hs :l fil

Importer modul :m Data.List Data.Map

Hjelp :?

Kraftige greier, kan brukes til debugging mm om man vet hvordan.

Kjapt om type-deklarasjoner og partials

```
1  thing :: Int
2  thing = 9
3
4  -- Funksjoner er verdier med typer som alt annet
5  addInts :: Int -> (Int -> Int)
6  addInts x y = x + y
7  -- Trenger ikke angi parametre, "point-free"
8  add2 :: Int -> Int
9  add2 = addInts 2 -- Funksjonskall
10
11 eleven :: Int
12 eleven = add2 thing
```

Partials og typer del 2

```haskell
-- a kan vare alt som er et tall
double :: Num a => a -> a
double = (* 2) -- partial applikasjon av *

-- Med lister
doubleAll :: Num a => [a] -> [a]
doubleAll = map double
-- eller doubleAll xs = map double xs

-- elementvis minste element av to lister av elementer
smallestItems :: Ord a => [a] -> [a] -> [a]
smallestItems = zipWith (<)

-- Kan legge flere restriksjoner
silly :: (Num a, Ord a, Show a) -> a -> IO a
-- Num a impliserer Ord a og Show a...
```

Egendefinerte typer og pattern-matching

```haskell
--    Type        Constructor              Constructor
data Record = Statistics Int Int | Word String
            deriving (Show, Read, Eq)

isWord :: Record -> Bool
isWord (Word _) = True
isWord _ = False

leftNumber :: Record -> Int
leftNumber (Statistics left _) = left
-- Tryner med Word

factorial :: Int -> Int
factorial 0 = 1
factorial n = n * factorial (n - 1)
```

Mer om egendefinerte typer

```haskell
data Tree a =
  Leaf |
  Node { value :: a, left :: Tree a, right :: Tree a}
  deriving (Show, Eq)

-- Haskell genererer 'accessors' value, left, right
collect :: Tree a -> [a]
collect Leaf = []
collect t = [value t] ++
            collect (left t) ++
            collect (right t)

-- Sett verdi for node
setValue :: Tree a -> a -> Tree a
setValue Leaf v = Node v Leaf Leaf
setValue t v = t {value = v}
-- Oops, returnerer helt nytt tre
```

factor

factor [numbers]... Print the prime factors of each specified integer NUMBER. If none are specified on the command line, read them from standard input.

calc - reverse polish notation calculator

```
$ ./calc 3 4 +
7.0
$ ./calc 3 4 + lg
2.807354922057604
$ ./calc 3 4 1 + + lg
3.0
$ ./calc 3 4 1 + + lg 2 '*'
6.0
```