# 1 Forelesning 2

## 1.1 Tall i Haskell

## 1.2 Funksjonsdefinisjoner

Matematisk skriver vi $f : A \to B$ for å si at funksjonen $f$ tar input av typen $A$ og returnerer $B$. Vanligvis skrives funksjoner som

$$f(x) = x^2 + 4$$

Her er det at implisitt at det er en funksjon av typen

$$f : \mathbb{R} \to \mathbb{R}$$

I haskell deklareres funksjoner på følgende måte

- Type deklarasjon: `myFun :: A -> B`

## 1.3 Polymorfi

Ordet *polymorfi* kommer fra de greske røttene polus og morphe, i.e. *mange* og *form*. Så kort oversatt til norsk: *flerformet*.

I haskell er det to typer polymorfi:

- Parametrisk polymorfi (aka. typevariabler)
- Ad-Hoc polymorfi (aka. typeklasser)

Idag skal vi se på den første av disse.

En parametrisk, polymorf funksjon i Haskell er en funksjon som bruker typevariabler til å defineres for alle typer samtidig.
OBS:

- Typevariables begynner alltid med liten forbokstav.
- Konkrete typer (ikke variabler) har stor forbokstav.
- Noen innebygde konkrete typer har speisell syntaks: lister, tupler osv.

**Eksempel.** Et veldig enkelt eksempel

```
id :: t -> t
id a = a
```

Denne funksjonen tar et element inn og spytter samme element ut. Ofte bruker vi (forvirrende nok) samme bokstav for typen og elementene i typen

```
const :: a -> b -> a
const a b = a
```

◇

**Eksempel.** Klarer vi å finne en type for funksjonen fra forrige forelesning?

```
h ::  ?
h z x = z (z x)
```

**Svar:**

```
h ::  (a -> a) -> a -> a
h z x = z (z x)
```

I lambdakalkyle er dette representasjonen av tallet 2. ◇

**Eksempel.** Selv de enkleste ting kan være polymorfe.

```
[] ::  [a]
```

Den tomme listen er en liste av alle typer. ◇

**Oppgave.** Lag en funksjon med typen

1. `f ::  a -> a -> a`
2. `g ::  a -> [[a]]`
3. `t ::  (a -> b) -> (b -> c) -> (a -> c)`
4. `s ::  (a -> b -> c) -> (a -> b) -> a -> c`

**Svar:**

1. `f x y = x`
2. `g a = [[a,a], [a,a]]`
3. `t f g a = g (f a)`
4. `s f g a = f a (g a)`

## 1.4 Uforanderlige verdier

En verdi er *uforanderlige* dersom den ikke kan endres etter at den er opprettet. I Haskell er *alle* verdier uforanderlige. Forandring uttrykkes istedet ved hjelp av funksjoner.

I begynnelsen kan det være litt forvirrende fordi noe som heter "aliasing", gjør at det ser ut som om verdier kan forandre seg i GHCi.

### 1.5 Neste tema: Vanlige typer

Vi skal gå gjennom følgende typer:

- Tupler
- Maybe
- Lister
- Either
- Map