1 Forelesning 0

1.1 Praktiske Detaljer

- Ukesoppgaver er obligatoriske
  - Automatisk rettet
  - Poeng for oppgavene, 24 poeng totalt
  - Må ha 18 poeng for å ta eksamen
  - 3% Bonus for å ta eksamen

Ukesoppgaver: Fredag hver uke, første frist er 2 september

MittUIB: Modules, videoer og oppgaver

OBS: Følg meg på kunngjøringer på MittUIB. Viktig å se diverse videoer før forelesning.

1.2 Hva gjør Haskell spesielt som programmeringspråk?

- Funksjonellt programmeringsspråk
- Algebraisk data typer
- Typeutledning og polymorfi
- Verdier er uforanderlige → Memoisering
- Lat og strikt evaluering av verdier

Haskell Standarden - Definerer haskell som språk, nyeste versjon fra 2010. **GHC** - Den dominerende haskellkompilatoren, har gjevnlig nye utgivelser, og tilbyr mange utvidelser av språket.

2 Forelesning 1

2.1 Plan for forelesningen

- Gjennomgang av forventinger
- Online ressurser
- Eksempler på programmer i Haskell
- Funksjoner i Haskell
- Strukturen til et Haskellprogram

2.2 Online Haskellressurser

- *Learn You a Haskell*: learnyouahaskell.com
- *Haskell wikibok*: en.m.wikibooks.orgs/wiki/haskell
- *Hoogle*: hoogle.haskell.ord
- *Mer*: haskell.ord/documentation

2.3 Eksempler på programmer laget i Haskell

- Pandoc
- Xmonad
- Darcs
- GF - Grammatical Framework
- GitHub's semantic tool

Og andre diverse selskaper som Standard Chartered og Klarna.

2.4 Funksjoner

Hva er en funksjon? Vi bruker en funksjon ved å få en verdi ved å gi den et argument.

I matematikken brukes $f(x)$ for å bruke en funksjon f på en verdi x. Hvis funksjonen tar imot flere argumenter skriver man $f(x, y, z)$ for å gi dem.

I Haskell droppes parantesene, og man skriver bare `f x`, og dersom det er flere argumenter skrives det `f x y z`. For å sette sammen funksjoner, må vi likevel bruke paranteser: `f (g x)`. Dersom vi hadde skrevet `f g x` ville vi gitt to argumenter til funksjonen.

2.5 Haskellprogrammer

Filnavn i haskell slutter på `.hs` - ellers er hver fil ofte en *modul*, hvor filnavn ofte er det samme som modulnavn. Modulnavn kommer øverst i filen, og er på formen `module moduleName where`. Verdien `main` er en spesiell verdi som har typen `IO ()`. For å lage en kjørbar fil må main verdien ligge i modulen main.

2.5.1 Presidensregler

I Haskell binder funksjonene sterkes, det vil si at koden under tolkes på følgende måte.

3 Forelesning 2

3.1 Tall i Haskell

3.2 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`

3.3 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)`

3.4 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.

3.5 Neste tema: Vanlige typer

Vi skal gå gjennom følgende typer:

- Tupler
- Maybe
- Lister
- Either
- Map