1 Forelesning 7

1.1 Dypere mønstermatching

Eksempel. Skriv en funksjon som sjekker om en typer er sortert.

1. Hva er typen til en slik funksjon?
2. Hva må funksjonen gjøre for å sjekke at en liste er sortert?
3. Hvis vi vil bruke rekursjon, hvilke mønster kan vi bruke?

```
sorted :: (ord a) => [a] -> Bool
```

◇

Eksempel. Vi kan også matche hvert enkelt element i listen, basert på typen

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

◇

1.2 Gjensidig rekursjon

To funksjoner kan v1're definert i termer av hverandre:

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

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

1.3 Egendefinerte datatyper

I haskell kan man innføre nye datatyper ved hjelp av nøkkelordet data:

```
data CelestialObject = Star String Integer
                     | Planet String String
                     | Moon String String
```

Deklarasjonen består av

- Navn på datatypen: `CelestialObject`
- Liste med konstruktører: `Star, Planet, Moon`

- Argumenter til konstruktøren (data som lagres i elementene)

Når en datatyper er definert kan vi lage elementer i den:

```
solarSystem :: [CelestialObject]
solarSystem = [Star "The Sun" 4600000000,
               Planet "Mercury" "The Sun",
               Planet "Venus" "The Sun",
               Planet "Earth" "The Sun", Moon "The Moon" "Earth"
               Planet "Mars" "The Sun", Moon "Phobos" "Mars"
                                      , Moon "Deimos" "Mars"]
```

Vi kan også definere funksjoner ved hjelp av mønster

```
displayInfo :: CelestialObject -> String
displayInfo (Star name age)
= "The star " ++ name ++ " is " ++ age ++ " years old."
```