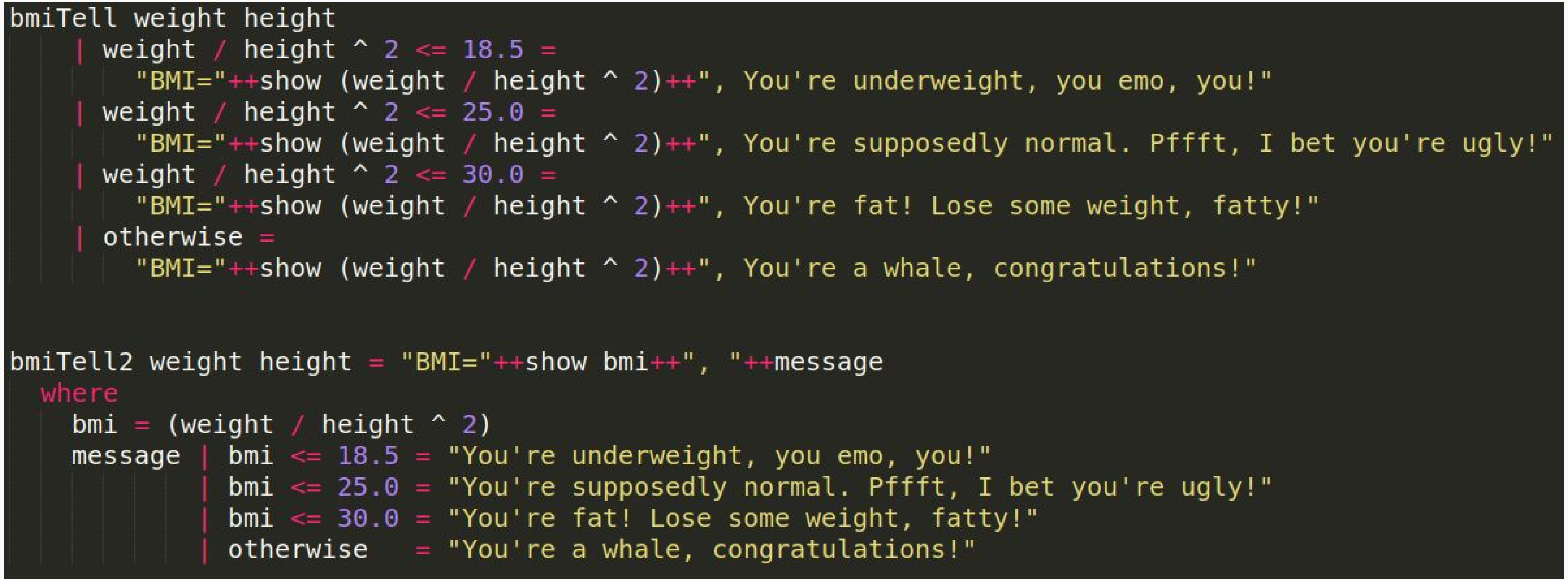
# A little more Haskell

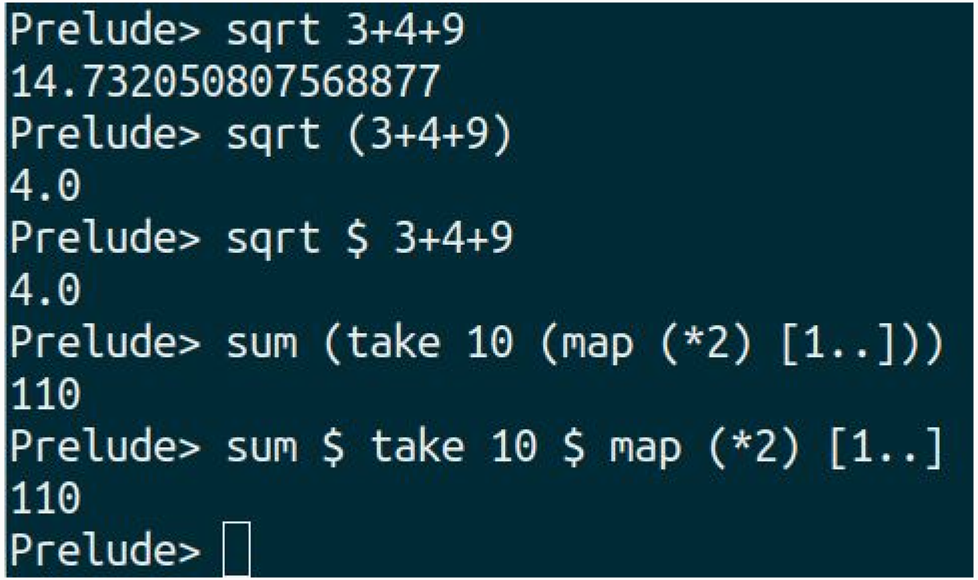
Σπύρος Αυλωνίτης - Γιάννος Χατζηαγάπης

# Where



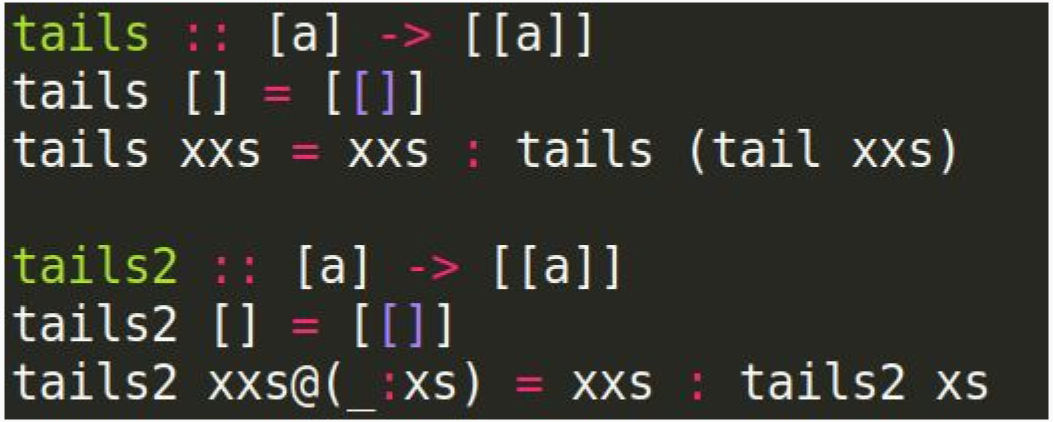
```haskell
bmiTell weight height
    | weight / height ^ 2 <= 18.5 =
        "BMI="++show (weight / height ^ 2)++", You're underweight, you emo, you!"
    | weight / height ^ 2 <= 25.0 =
        "BMI="++show (weight / height ^ 2)++", You're supposedly normal. Pffft, I bet you're ugly!"
    | weight / height ^ 2 <= 30.0 =
        "BMI="++show (weight / height ^ 2)++", You're fat! Lose some weight, fatty!"
    | otherwise =
        "BMI="++show (weight / height ^ 2)++", You're a whale, congratulations!"


bmiTell2 weight height = "BMI="++show bmi++", "++message
  where
    bmi = (weight / height ^ 2)
    message | bmi <= 18.5 = "You're underweight, you emo, you!"
            | bmi <= 25.0 = "You're supposedly normal. Pffft, I bet you're ugly!"
            | bmi <= 30.0 = "You're fat! Lose some weight, fatty!"
            | otherwise   = "You're a whale, congratulations!"
```

## Function application with ‘$’



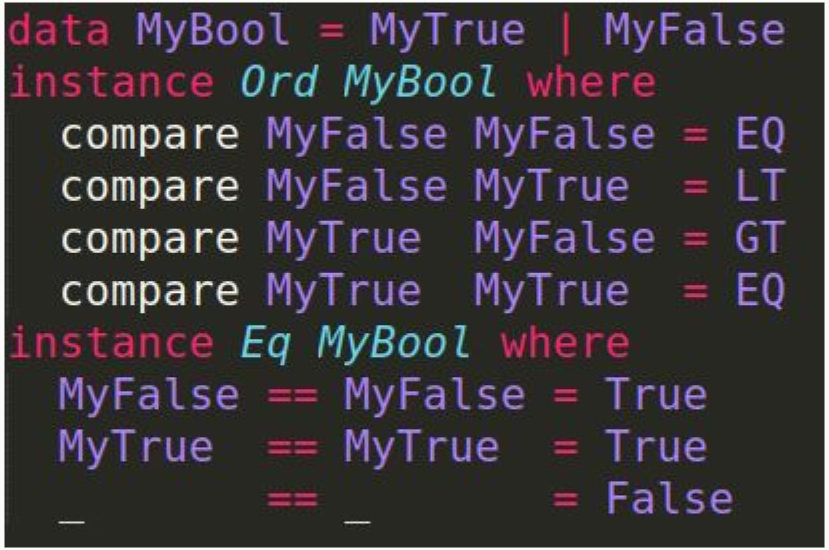
```
Prelude> sqrt 3+4+9
14.732050807568877
Prelude> sqrt (3+4+9)
4.0
Prelude> sqrt $ 3+4+9
4.0
Prelude> sum (take 10 (map (*2) [1..]))
110
Prelude> sum $ take 10 $ map (*2) [1..]
110
Prelude> 
```

## Parameter naming with ‘@’



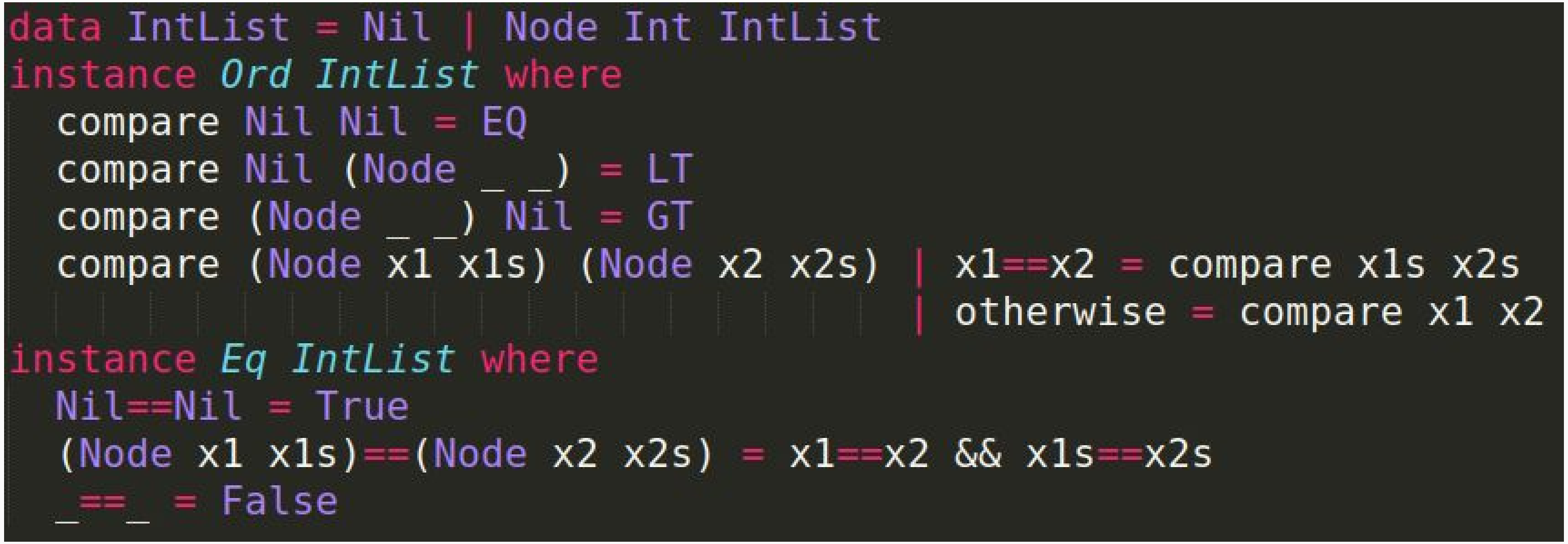
```haskell
tails :: [a] -> [[a]]
tails [] = [[]]
tails xxs = xxs : tails (tail xxs)

tails2 :: [a] -> [[a]]
tails2 [] = [[]]
tails2 xxs@(_:xs) = xxs : tails2 xs
```

# Ordering and equality between data types



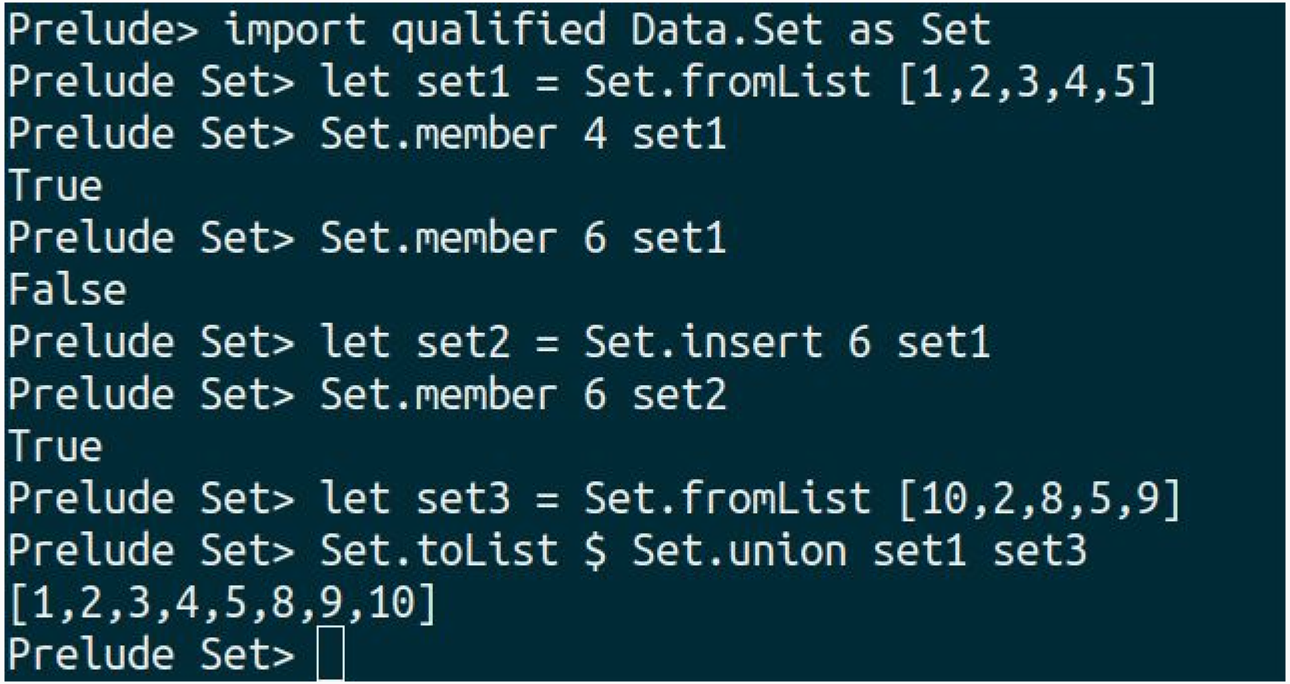
```haskell
data MyBool = MyTrue | MyFalse
instance Ord MyBool where
  compare MyFalse MyFalse = EQ
  compare MyFalse MyTrue  = LT
  compare MyTrue  MyFalse = GT
  compare MyTrue  MyTrue  = EQ
instance Eq MyBool where
  MyFalse == MyFalse = True
  MyTrue  == MyTrue  = True
  _       == _       = False
```

# Ordering and equality between data types



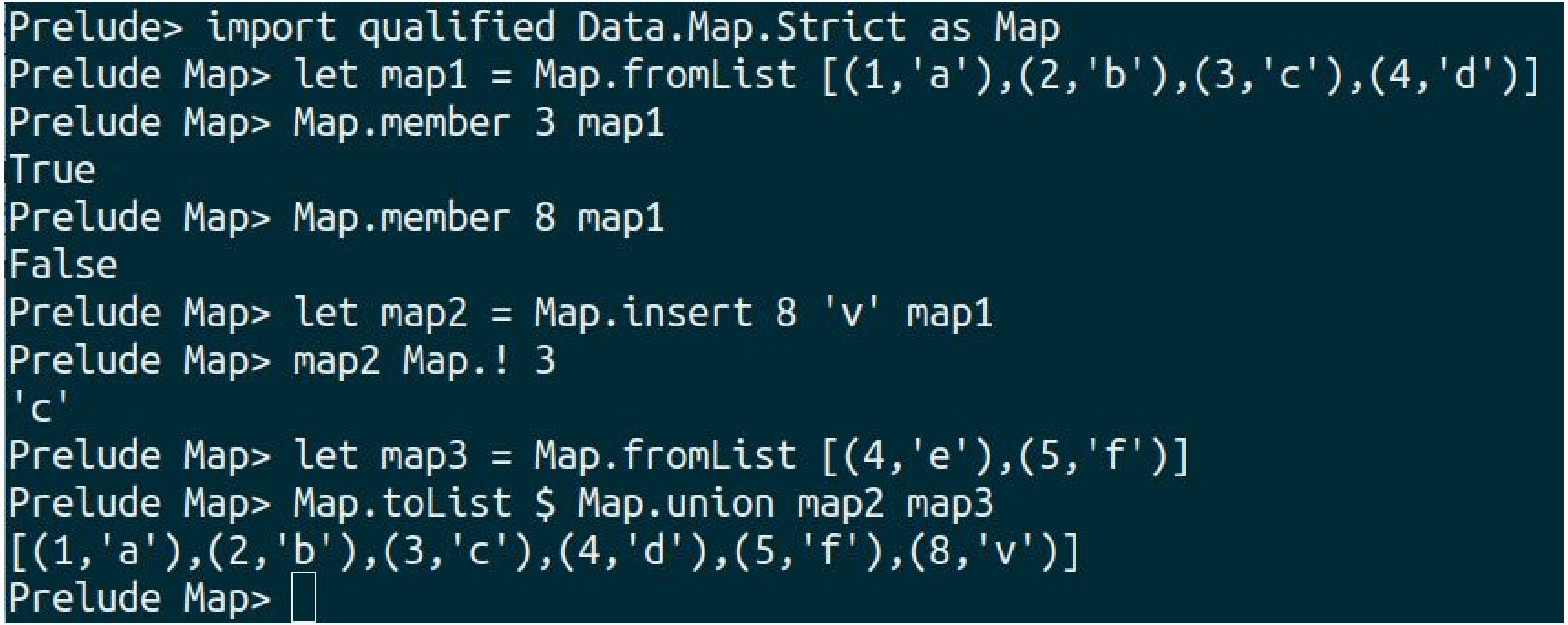
```haskell
data IntList = Nil | Node Int IntList
instance Ord IntList where
  compare Nil Nil = EQ
  compare Nil (Node _ _) = LT
  compare (Node _ _) Nil = GT
  compare (Node x1 x1s) (Node x2 x2s) | x1==x2 = compare x1s x2s
                                      | otherwise = compare x1 x2
instance Eq IntList where
  Nil==Nil = True
  (Node x1 x1s)==(Node x2 x2s) = x1==x2 && x1s==x2s
  _==_ = False
```

# Data.Set



```
Prelude> import qualified Data.Set as Set
Prelude Set> let set1 = Set.fromList [1,2,3,4,5]
Prelude Set> Set.member 4 set1
True
Prelude Set> Set.member 6 set1
False
Prelude Set> let set2 = Set.insert 6 set1
Prelude Set> Set.member 6 set2
True
Prelude Set> let set3 = Set.fromList [10,2,8,5,9]
Prelude Set> Set.toList $ Set.union set1 set3
[1,2,3,4,5,8,9,10]
Prelude Set> 
```

## Data.Map



```
Prelude> import qualified Data.Map.Strict as Map
Prelude Map> let map1 = Map.fromList [(1,'a'),(2,'b'),(3,'c'),(4,'d')]
Prelude Map> Map.member 3 map1
True
Prelude Map> Map.member 8 map1
False
Prelude Map> let map2 = Map.insert 8 'v' map1
Prelude Map> map2 Map.! 3
'c'
Prelude Map> let map3 = Map.fromList [(4,'e'),(5,'f')]
Prelude Map> Map.toList $ Map.union map2 map3
[(1,'a'),(2,'b'),(3,'c'),(4,'d'),(5,'f'),(8,'v')]
Prelude Map>
```

# Thank you

Some examples were taken from

learnyouahaskell.com

which I highly recommend as a learning resource

Thanks to Konstantinos Triantafyllou
for last year's presentation