# HASKELL TUTORIAL 2

Prelude =

```
rmseLet :: [Float] -> [Float] -> Float
rmseLet la lb =
  let
    n = len la
    diff = zipWith (-) la lb
    diffSq = map (^2) diff
  in
    (foldr (+) 0 diffSq) / n
```

```
mseWhere :: [Float] -> [Float] -> Float
mseWhere la lb = (foldr (+) 0 diffSq)/n
  where
    n = len la
    diff = zipWith (-) la lb
    diffSq = map (^2) diff
```

:l / :load

:t / :type - typecoof object.

```
type Vector = [Float]
```

↑

similar to typedef.

```
type Fl' = Float

f :: Fl' -> Fl' -> Fl'
type Vector = [Float]

data Tree = Empty | Node Int Tree Tree

f :: Tree -> String
f Empty = "empty node"
f (Node a b c) = "full node"


f :: Tree -> Int
f Empty = -1
f (Node a _ _) = a


data Vector' = Vector' Int [Float]
data Vector'' = Vector'' {dim :: Int, vector :: [Float]}

getDims :: Vector' -> Int


data Tree = Empty | Node Int Tree Tree

f :: Tree -> Bool
f Empty = true
f _ = false.
```

```
data Vector'' = Vector'' {dim :: Int, vector :: [Float]}

class Show a where
  show :: a -> String

instance Show Vector'' where
  show v = (show $ dim a) ++ " " ++ (show $ vector v)
```

x hello world.