```
(+), (*), (-) :: Num a => a -> a -> a
(/) :: Fractional a => a -> a -> a
div, mod :: Integral a => a -> a -> a
13 `div` 5 = 2    13 `mod` 5 = 3    13 / 5 = 2.6

(^) :: (Num a, Integral b) => a -> b -> a
even, odd :: Integral a => a -> Bool

(<), (<=), (>), (>=) :: Ord a => a -> a -> Bool
(==), (/=) :: Eq a => a -> a -> Bool
(&&), (||) :: Bool -> Bool -> Bool
not :: Bool -> Bool
max, min :: Ord a => a -> a -> a
max 3 7 = 7    min 3 7 = 3

round :: (RealFrac a, Integral b) => a -> b
fromIntegral :: (Integral a, Num b) => a -> b
round 2.3 = 2    fromIntegral(length [1,2]) + 3.2 = 5.2

To use the following functions:    import Data.Char

isAlpha, isLower, isUpper, isDigit :: Char -> Bool
isAlpha 'a' = True    isAlpha '3' = False
isLower 'a' = True    isLower 'A' = False

toLower, toUpper :: Char -> Char
toLower 'A' = 'a'     toUpper 'a' = 'A'

digitToInt :: Char -> Int
intToDigit :: Int -> Char
digitToInt '3' = 3    intToDigit 3 = '3'
```

Figure 1: Some functions on basic data

```
sum, product :: (Num a) => [a] -> a
sum [1.0,2.0,3.0] = 6.0
product [1,2,3,4] = 24

maximum, minimum :: (Ord a) => [a] -> a
maximum [3,1,4,2] = 4
minimum [3,1,4,2] = 1

concat :: [[a]] -> [a]
concat ["go","od","bye"] = "goodbye"

(!!) :: [a] -> Int -> a
[9,7,5] !! 1 = 7

head :: [a] -> a
head "goodbye" = 'g'

init :: [a] -> [a]
init "goodbye" = "goodby"

takeWhile :: (a->Bool) -> [a] -> [a]
takeWhile isLower "goodBye" = "good"

dropWhile :: (a->Bool) -> [a] -> [a]
dropWhile isLower "goodBye" = "Bye"

elem :: (Eq a) => a -> [a] -> Bool
elem 'd' "goodbye" = True

and, or :: [Bool] -> Bool
and [True,False,True] = False
or [True,False,True] = True

reverse :: [a] -> [a]
reverse "goodbye" = "eybdoog"

(++) :: [a] -> [a] -> [a]
"good" ++ "bye" = "goodbye"

length :: [a] -> Int
length [9,7,5] = 3

tail :: [a] -> [a]
tail "goodbye" = "oodbye"

last :: [a] -> a
last "goodbye" = 'e'

take :: Int -> [a] -> [a]
take 4 "goodbye" = "good"

drop :: Int -> [a] -> [a]
drop 4 "goodbye" = "bye"

replicate :: Int -> a -> [a]
replicate 5 '*' = "*****"

zip :: [a] -> [b] -> [(a,b)]
zip [1,2,3,4] [1,4,9] = [(1,1),(2,4),(3,9)]

To use the following function:    import Data.List

isPrefixOf :: Eq a => [a] -> [a] -> Bool
isPrefixOf "abc" "abcde" = True
```

Figure 2: Some functions on lists