Some common list functions

1. Selecting parts of lists

`head :: [a] -> a` return the first element of a list

`tail :: [a] -> [a]` return all but the first element of a non-empty list

`last :: [a] -> a` return the last element of a list

`init :: [a] -> [a]` return all but the last element of a non-empty list

`take :: Int -> [a] -> [a]` return the first n elements of a list
`take 5 ( iterate (\x -> x^ 2) 2 ) = [2,4,16,256,65536]`

`drop :: Int -> [a] -> [a]` drops the first terms of a list
`drop 2 [[5,2,7],[ ],[0,1],[1..3]] = [[0,1],[1,2,3]]`

2. Getting information about a list

`length :: [a] -> Int`

`elem :: Eq a => a -> [a] -> Bool` tell whether the element is a term of the list
`elem [ ] [[5,2,7],[ ],[0,1],[1..3]] = True`

`(!!) :: [a] -> Int -> a` get the n^{th} element of a list, where the first element is the 0^{th}
`[1..10]!!7 = 8`

`and :: [Bool] -> Bool` logical conjunction

`or :: [Bool] -> Bool` logical disjunction

`sum :: Num a => [a] -> a`

`product :: Num a => [a] -> a`

3. Combining lists

`(++) :: [a] -> [a] -> [a]` join two lists into one
`[5,2,7] ++ [0,1] = [5,2,7,0,1]`

`concat :: [[a]] -> [a]` join list of lists into one
`concat [[5,2,7],[ ],[0,1],[1..3]] = [5,2,7,0,1,1,2,3]`

`zip :: [a] -> [b] -> [(a,b)]` return pairs of corresponding elements of two lists
`zip [1..4] "abcdefgh" = [(1,'a'),(2,'b'),(3,'c'),(4,'d')]`

`unzip :: [(a,b)] -> ([a],[b])` reverses the zipping process
`unzip [(1,'a'),(2,'b'),(3,'c'),(4,'d')] = ([1,2,3,4],"abcd")`

4. Creating and manipulating lists

`replicate :: Int -> a -> [a]` make a list of copies of one element
`replicate 3 'Z' = "ZZZ"`

`reverse :: [a] -> [a]` return list in reverse order

`sort :: Ord a => [a] -> [a]` (from the `List` library) return a sorted list

`splitAt :: Int -> [a] -> ([a],[a])` split the list into the first n and the rest
`splitAt 4 "abcdefg" = ("abcd","efg")`

`nub :: Eq a => [a] -> [a]` (from the `List` library) remove duplicates
`nub [1,3,1,4,3,3] = [1,3,4]`

`iterate :: (a -> a) -> a -> [a]` return an infinite list `[x, f(x), f(f(x)), ... ]`
`take 5 ( iterate (\x -> x^ 2) 2 ) = [2,4,16,256,65536]`

5. Using functions on lists

`map :: (a -> b) -> [a] -> [b]` apply a function to each term of a list
`map sqrt [1..5] = [1.0, 1.41421, 1.73205, 2.0, 2.23607]`

`filter :: (a -> Bool) -> [a] -> [a]` select elements of a list that satisfy a boolean function
`filter (\x -> length x > 2) [[5,2,7],[ ],[0,1],[1..3]] = [[5,2,7],[1,2,3]]`

`zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]` zip, then apply a function to each pair
`zipWith (*) [2,3,4] [5,5,0] = [10,15,0]`

`takeWhile :: (a -> Bool) -> [a] -> [a]` returns a list containing elements from the front of the list while the condition is satisfied.
`takeWhile (<1000) ( iterate (\x -> 2*x) 2 ) = [2,4,8,16,32,64,128,256,512]`

`foldr1 :: (a -> a -> a) -> [a] -> a` "fold" the list starting at the right
`foldr1 (-) [1,2,3,4] = (-2)` (calculates 1-(2-(3-4)))

`foldr :: (a -> b -> b) -> b -> [a] -> b` "fold" the list starting at the right, using a starting value
`foldr (-) 5 [1,2,3,4] = 3` (calculates 1-(2-(3-(4-5))))

```
foldr (+) 0 = sum
foldr (++) [ ] = concat
foldr (&&) True = and
foldr ((:).f) [ ] = map f
foldr (\x xs -> if p x then x:xs else xs) [ ] = filter p
```