Lists

The major data type in any functional language is the *list*. You have already seen a list in the form of a *sentence* which is in fact a list of words. A list, however, can contain any type, not just words. We can even have lists that contain other lists!

Lists are written in square brackets and all elements in the lists must be the same type. Here are some examples given with their types:

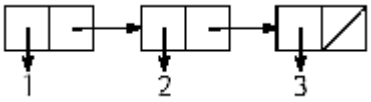
```
[1,2,3] :: [Int]

['h','e','l','l','o'] :: [Char]

[[1,2], [3]] :: [[Int]]

[(+), (*)] :: [Int -> Int -> Int]
```

A list with no elements is written as `[]` and is pronounce "nil". To add a single element x to the front of a list xs, we write `x : xs`. In actuality, the list `[1,2,3]` is equivalent to `1 : (2 : (3 : []))`. This list could be visualized as:



It is important to understand that a list is not symmetric. It is easy to get the first item in the list but to get to the last items requires marching through the entire list.

Let's consider how we can perform so list operations. In doing so the type `[a]` indicates a list of any type. List functions are often broken into cases: the empty list `[]` and the nonempty list `(x:xs)`. For the nonempty list the value `x` is the first thing in the list and `xs` is everything but the first item. You may find it helpful to use the substitution model on the functions below to see how they behave:

```
-- compute the length of a list
len :: [a] -> Int
len [] = 0
len (x:xs) = len xs + 1

-- return the nth item in the list (start counting at 0)
nth :: Int -> [a] -> a
nth n (x:xs) = if n==0
               then x
               else nth (n-1) xs
```

```
-- append two lists together
append :: [a] -> [a] -> [a]
append [] y = y
append (x:xs) y = x : (append xs y)
```

The operations that can be performed on lists is lengthy and is provided here. Let's zero in on just a few of the key operations:

Function	Explanation
`head`	gives the first item in the list
`tail`	gives all but the first item in the list
`++`	appends two lists together
`null`	tests to see if a list is empty

There are many higher order function that operate on lists. Here are some that should seem familiar:

Function	Explanation
`map`	applies a function to every member of the list
`filter`	applies a predicate to a list and keeps only the items satisfying the predicate
`foldl`	combines a list using a combiner function (applied left to right)
`foldr`	combines a list using a combiner function (applied right to left)