Tail Recursion

We have seen examples of linear recursion but there is another type of recursion that feels more like iteration (loops). Consider an alternate version of the factorial function:

```
factTail :: Int -> Int
factTail n = fTail n 1 where
         fTail n result =
                   if n == 0
                   then result
                   else fTail (n-1) (result * n)
```

This type of recursion is called *tail recursion* and we had to define a helper function `fTail` with two arguments within our function `factTail`. Let's look at the computation of `factTail 4` with the substitution model

Expression	Substitution explanation
$factTail$ 4	substitute into the body of $factTail$
$fTail$ 4 1	substitute for $fTail$ (computes 4! * 1)
$fTail$ 3 4	substitute for $fTail$ (computes 3! * 4)
$fTail$ 2 12	substitute for $fTail$ (computes 2! * 12)
$fTail$ 1 24	substitute for $fTail$ (computes 1! * 24)
$fTail$ 0 24	substitute for $fTail$ (computes 0! * 24)
24	

In tail recursion there is no winding and unwinding. Instead you see that this feels like we are looping and remembering and changing values each time through the loop.

Let's do a tail recursive version of `revWord`:

```
revWordTail :: Language -> Language
revWordTail w = revTail w (word "") where
         revTail w result = if (empty w)
                            then result
                            else revTail (butFirst w) ((firstItem w) +++ result)
```

Use the substitution model to look at the computation `revWordTail (word "cat")`.

Try writing tail recursive functions for the following:

```
-- compute the base value raised to the power of the exponent
powerTail :: Int -> Int -> Int
powerTail base exp = ...

-- compute the number of letters in a word
lengthTail :: Language -> Int
lengthTail w = ...
```