Activity 5

Partial application of a function is using the function but not supplying all the arguments. Let's look at some examples directly in the console.

Enter the following directly into ghci:

```
> let add x y = x + y
> :type add
```

You should get
add :: Num a => a -> a -> a

The **Num a =>** part means that the type of a is a number so it could be an Int or Float or Double. Therefore the **a->a->a** indicates that x and y must be numbers and the return value is also a number.

Suppose we put in some parentheses on the type like **a -> (a -> a)**. If we write it this way, it looks like if we supply one argument of type a, we should get a function of type **(a-> a)**. That means that **add 3** should give us a function. Let's try it by naming f to be the function that **add 3** returns:

```
> let f = add 3
> :type f
```

You should get
f :: Integer -> Integer

This shows that the partial application of giving the function **add** just one argument rather than two gives us another function which takes an Integer and returns an Integer.

What would we expect if we used the function f?
```
> f 4
```

Try partial application on the function **twice**.

```
> let twice f x = f (f x)
> let square x = x * x
>:type twice

> let g = twice (+1)
> let h = twice square
```

What are the types for functions g and h and what do those functions do?

Try partial application on function from the Words module
>:load Words.hs

First take a look at the types of **every** and **accumulate**:
>:type every
>:type accumulate

Now we will try partial application on those functions:

> let f = every firstItem
> let g = accumulate (+++)

What are the types of the functions f and g and what do those functions do?