# List Operations - Part 2

Implement the functions below. Note that many of these functions are available in the standard library,[1] but the goal of this exercise is to practice by implementing them from scratch. When writing a recursive function involving lists, put some thought into choosing the right base case.

Recall that the syntax of pattern-matching on a list is as follows (where `x` is the head of the list and `xs` is the tail):

```
function :: [...] -> ...
function []     = ...
function (x:xs) = ...
```

**Note that HLint (on E-Systant) may generate warnings, you can ignore these.**

- Write a function `myProduct :: [Integer] -> Integer`, which takes a list of integers and computes their product.
  **Note that we use `Integer` here instead of `Int`. The former can represent numbers of arbitrary size, whereas the latter will overflow.**

  ```
  Main> myProduct [1,2,3]
  6

  Main> myProduct []
  1

  Main> myProduct [-2,3,-4,5,-6]
  -720
  ```

- Write a function `insert :: Int -> [Int] -> [Int]`, which takes an integer and a list of integers and inserts the integer into the list at the first position where it is less than or equal to the next element.

  ```
  Main> insert 0 [1,2,3]
  [0,1,2,3]

  Main> insert 2 [1,0,3]
  [1,0,2,3]

  Main> insert 4 [1,2,3]
  [1,2,3,4]
  ```

- Write a function `myLast :: [Int] -> Int` which returns the last element of a list. Assume that the input lists are non-empty[2]

[1] For example, see module `Data.List`, which can be found at `http://downloads.haskell.org/~ghc/7.6.3/docs/html/libraries/base/`.

[2] You may return an error using the `error` function in case the list is empty.

```
Main> myLast [1,2,3,4,5]
5
```