Exercises
Recursive Functions

** Exercise 1

Define a recursive function

```
sumdown :: Int -> Int
```

that returns the sum of all the non-negative integers down to zero. For example:

```
sumdown 3
```

will return $3+2+1+0=6$

** Exercise 2

Define the ***exponention*** (to the power of) function for non-negative numbers using the same pattern of recursion as the multiplication operator in notes, and show how the expression

```
exponention 2 3
```

is evaluated using your definition.

** Exercise 3

Define a recursive function

```
fibonacci :: Int -> Int
```

that calculates the Fibonacci number as per the following definition
$F_0 = 0, F_1 = 1$
$F_n = F_{n-1} + F_{n-2}$

** Exercise 4

Define a recursive function

```
myInit :: [a] -> [a]
```

that removes the last element from a non-empty list. Construct the definition using the 5 step process as discussed in lectures.

** Exercise 5

Without looking at the definitions from the standard Prelude, define the following library functions on lists using recursion:

1. Decide is all logical values in a list are ***True***

   ```
   myAnd :: [Bool] -> Bool
   ```

2. Concatenate a list of lists

   ```
   myConcat :: [[a]] -> [a]
   ```

3. Produce a list with ***n*** identical elements

   ```
   myReplicate :: Int -> a -> [a]
   ```

4. Select the n^{th} element of a list

   ```
   myNth :: [a]-> Int -> a
   ```

5. Decide if an value is an element of a list

   ```
   myElem  :: Eq a => a -> [a] -> Bool
   ```

*** Exercise 6

Using the five-step process, construct the library functions that:

1. calculate the ***sum*** of a list of numbers;
2. ***take*** a given number of elements from the start of a list;
3. select the ***last*** element of non-empty list.

*** Exercise 7

Define a recursive function

```
merge :: Ord a => [a] -> [a] -> [a]
```

that merges two sorted lists to give a single sorted list. Note : Your definition should not use other functions on sorted lists such as ***insert*** or ***isort***, but should be defined using explicit recursion.

**** Exercise 8

Using ***merge***, define a function

```
msort :: Ord a => [a] -> [a]
```

that implements *merge sort*, in which the empty list and singleton lists are already sorted, and any other list is sorted by merging together the two lists that result from sorting the two halves of the list separately.

Hint 1: First define a function

```
halve :: [a] -> ([a] , [a])
```

that splits a list into two halves whose lengths differ by at most one.
Hint 2: You can use the following functions (though you may not need to)

```
fst :: (a,b) -> a
snd :: (a,b) -> b
fst (x,y) = x
snd (x,y) = y
```

Solutions

Solutions to exercise 1

```
sumdown :: Int -> Int
sumdown 0 = 0
sumdown n = n + sumdown (n-1)
```

Solutions to exercise 2

```
exponention :: Int -> Int -> Int
exponention 0 _ = 0
exponention _  0 = 1
exponention m n = m * exponention m  (n-1)
```

Solutions to exercise 3

```
fibonacci :: Integral a => a -> a
fibonacci 0 = 0
fibonacci 1 = 1
fibonacci n = fibonacci (n-1) + fibonacci (n-2)
```

Solutions to exercise 4

```
myInit :: [a] -> [a]
myInit [x] = []
myInit (x:xs) = x: myInit xs
```

Solutions to exercise 5

1. Decide is all logical values in a list are ***True***

   ```
   myAnd :: [Bool] -> Bool
   myAnd [] = True
   myAnd (b:bs) = b && myAnd (bs)
   ```

2. Concatenate a list of lists

   ```
   myConcat :: [[a]] -> [a]
   myConcat []   = []
   myConcat (x:xs)   = x ++ (myConcat xs)
   ```

3. Produce a list with ***n*** identical elements

```
myReplicate :: Int -> a-> [a]
myReplicate 0 _ = []
myReplicate n x = x: myReplicate (n-1) x
```

4. Select the n^{th} element of a list

```
myNth ::[a] -> Int -> a
myNth (x:xs)  0     = x
myNth (x:xs) n = myNth xs (n-1)
```

5. Decide if an value is an element of a list

```
myElem :: Eq a => a -> [a] -> Bool
myElem x [] = False
myElem  x' (x:xs)  | x' == x = True
                   | otherwise = myElem x' xs
```

Solutions to exercise 6

1. calculate the ***sum*** of a list of numbers;

```
sum' :: Num a => [a] -> a
sum' [] = 0
sum' (x:xs) = x + sum xs
```

2. ***take*** a given number of elements from the start of a list;

```
take' :: Int-> [b] -> [b]
take' 0 _ = []
take' _ [] = []
take' n (x:xs) = x: take' (n-1) xs
```

3. select the ***last*** element of non-empty list.

```
last' :: [a] -> a
last' [x] = x
last' (_:xs) = last xs
```

Solutions to exercise 7

```
merge :: Ord a => [a] -> [a] -> [a]
merge xs [] = xs
merge [] ys = ys
merge (x:xs) (y:ys) | x <= y = x: merge xs (y:ys)
                    | otherwise = y: merge (x:xs) ys
```

Solutions to exercise 8

```
halve :: [a] -> ([a], [a])
halve [x] = ([x], [])
halve xs = (firsthalf, secondhalf)
   where
    firsthalf = take half xs
    secondhalf = drop half  xs
    half = div (length  xs) 2

msort :: Ord a => [a] -> [a]
msort [] = []
msort [x] = [x]
msort xs = merge  (msort left)  (msort right )
    where
     (left, right) = halve xs
```