Exercises
Tail Recursive Functions

Exercise 1:

In the previous exercise set, you wrote the code for:

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

that calculates the Fibonacci number as per the following definition (note for non-negative integers)
$F_0 = 0, F_1 = 1$
$F_n = F_{n-1} + F_{n-2}$ Now, write this fibonacci using tail recursion.

```
fibT :: Int -> Int
```

Solution 1:

```
fibT :: Int -> Int
fibT n = helper n 1 0  where
          helper 1 prev prevBut1 =  prev
          helper n prev prevBut1 = helper (n-1) (prev+prevBut1) prev
```

Exercise 2:

Noting our definition of add from the previous lecture (this is actually tail recursive)

```
myAdd :: Int -> Int -> Int
myAdd x 0 = x
myAdd 0 y = y
myAdd x y = myAdd (x-1) (y+1)
```

write a function

```
myMult :: Int -> Int -> Int
```

which takes two positive integers and returns the product of the two numbers. This calcluation can only use + and −. Ensure that the solution is tail recursive

Solution 2:

non-tail recursive:

```
recMult :: Int -> Int-> Int
recMult x 1 = x
recMult 1 y = y
recMult x y = x + recMult x (y-1)
```

and using tail recursion:

```
tailMult :: Int->Int-> Int
tailMult x y  = helperMult x y 0
where helperMult x y acc
              | y ==0 = acc
              | otherwise = helperMult x (y-1) (acc+x)
```

Exercise 3:

Given the recursive function

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

that returns the reverse of a list, but this time, using tail recursion.

Solution 3:

```
reverseAccum :: [a] -> [a]
reverseAccum  list = revHelper [] list
     where revHelper  accum []       = accum
           revHelper  accum (x:xs)   = revHelper (x:accum) xs
```