Assignment 3

Task 1

a) The procedure "QuadraticEquation" works by taking in parameters a,b,c which represent the cofactors in the equation, computes the discriminant which determines whether there are any real solutions, if discriminant is negative set the RealSol parameter to false. If the discriminant is positive set RealSol to true and compute the solutions and assign them to the parameters X1 and X2.

A=2, B=1, C=-1 gives X1=0.5 X2=-1 and RealSol=true

A=2 B=1 C=2 gives RealSol=false while X1 and X2 remain unassigned.

b) 1. Procedural abstractions hides details, and lets us focus on higher level design. 2. Can be used as building blocks.

c) A procedure doesn't return a value, while a function does. Although a function really is just a procedural abstraction.

Task 2

The function Sum works by pattern matching to check for a list with a head and tail, takes the number in the head and recursively adds the sum of the tail. When the recursion meets the end of the list it returns 0.

Task 3

b) "fun {RightFold List Op U}" defines the function "RightFold" with the parameters "List" , "Op", and "U". "List" is the input list, "Op" is the function to perform on the elements in the list, and "U" is the value to be returned when reaching the end of the list.

```
"case List

    of Head|Tail then

      {Op Head {RightFold Tail Op U}}
    else
       U
    end"
```

This block of code pattern matches "List" looking for a list with a head and tail. In the case of a head and tail it calls the "Op" function with the head of the list as the first input, and recursively calls RightFold on the tail of the list as the second input.

c) "Sum" and "Length" are implemented as anonymous functions which are passed as arguments to the "RightFold" function. The "B" argument in both functions are supposed ro be recursive calls to the "RightFold" function, and the "A" argument can be used (or not used) to perform the desired operation with the result of the recursive call.

d) No, a left fold would not give different result for the Sum and Length operations because the order of the elements doesn't matter when purely adding and calculating length. The same goes for subtractions.

e) "U" should be 1 when using RightFold to calculate the product of list elements. because setting U to 0 would just give us 0 as a result, while multiplying with 1 doesn't change the result.

Task 4

Simply create a function which defines an anonymous function which calculates a quadratic eqation with the given parameters and return that function.

Task 5

b) My function "LazyNumberGenerator" returns a list where the first value is a start value given as a parameter and the second value is an anonymous function which calls "LazyNumberGenerator" with an incremented start value. The trick is to make the second element a function definition, not a function call. A limitation to this solution is that you have to call a function every time you want a new value.

Task 6

a) My orginal "Sum" function was not tail recursive because the last thing it did in each step was not the recursion, but adding the result of the recursion. Therefore I implemented a help function which has an accumulator which does the summation.

This new version is tail recursive because the last thing it does in each step is making a recursive call.

b) The benefit of tail recursion is that it is more memory efficient because it performs the calculation while it is going through the recursive stack instead of making one huge expression it calculates at the end.

c) A language has to support tail call optimization to benfit from tail recursion. Tail call optimization means replacing the old stack frame with the new stack frame instead of storing both since the old stack frame isn't needed when the recursive call is the last thing to happen at each step.