Nested functions

Using let expressions we can define functions inside any let-expression, thats because functions are bindings.

`let b1 b2...bn in e end`

nested functions example

```
fun count_from1 (x:int) =
    let fun count(from:int, to:int)=

        if from = to

        then to::[]

        else from :: count(from+1, to)

    in

        count(1,x)

    end
```

Better function Functions can use bindings in the environment where they are defined:

- Bindings from "outer" enviroments
  - Such as parameters to the outer function
- Earlier bindings in the let-expression

Better example

```
fun countup_from1_better(x:int)=
    let fun count(from:int)
        if from = x
        then x:: []
        else from :: count(from+1)
    in
        count 1
    end
```

When to use nested functions

Good style to define helper functions, if you restrict functions to where they are useful then if makes it easier to make changes to these functions.