# Isabelle/HOL Exercises
# Lists

## Sum of List Elements, Tail-Recursively

**(a)** Define a primitive recursive function *ListSum* that computes the sum of all elements of a list of natural numbers.

Prove the following equations. Note that *[0::'a..n]* und *replicate n a* are already defined in a theory `List.thy`.

```
consts ListSum :: "nat list ⇒ nat"

primrec
  "ListSum []     = 0"
  "ListSum (x#xs) = x + ListSum xs"

theorem ListSum_append[simp]: "ListSum (xs @ ys) = ListSum xs + ListSum ys"
  apply (induct xs)
  apply auto
done

theorem "2 * ListSum [0..<n+1] = n * (n + 1)"
  apply (induct n)
  apply auto
done

theorem "ListSum (replicate n a) = n * a"
  apply (induct n)
  apply auto
done
```

**(b)** Define an equivalent function *ListSumT* using a tail-recursive function *ListSumTAux*. Prove that *ListSum* and *ListSumT* are in fact equivalent.

```
consts ListSumTAux :: "nat list ⇒ nat ⇒ nat"

primrec
  "ListSumTAux []     n = n"
```

```
    "ListSumTAux (x#xs) n = ListSumTAux xs (x + n)"

  constdefs ListSumT :: "nat list ⇒ nat"
    ListSumT_def: "ListSumT xs == ListSumTAux xs 0"

  lemma ListSumTAux_add [rule_format]: "∀ a b. ListSumTAux xs (a+b) = a +
ListSumTAux xs b"
    apply (induct xs)
    apply auto
  done

  lemma [simp]: "ListSumT [] = 0"
    by (auto simp add: ListSumT_def)

  lemma [simp]: "ListSumT (x#xs) = x + ListSumT xs"
    by (auto simp add: ListSumT_def ListSumTAux_add[THEN sym])

  theorem "ListSumT xs = ListSum xs"
    apply (induct xs)
    apply auto
  done
```