Isabelle/HOL Exercises
Arithmetic

Power, Sum

Power

Define a primitive recursive function *pow x n* that computes x^n on natural numbers.

```
primrec pow :: "nat => nat => nat" where
  "pow x 0       = Suc 0"
| "pow x (Suc n) = x * pow x n"
```

Prove the well known equation $x^{m \cdot n} = (x^m)^n$:

```
theorem pow_mult: "pow x (m * n) = pow (pow x m) n"
```

Hint: prove a suitable lemma first. If you need to appeal to associativity and commutativity of multiplication: the corresponding simplification rules are named `mult_ac`.

```
lemma pow_add: "pow x (m + n) = pow x m * pow x n"
  apply (induct n)
  apply auto
done

theorem pow_mult: "pow x (m * n) = pow (pow x m) n"
  apply (induct n)
  apply (auto simp add: pow_add)
done
```

Summation

Define a (primitive recursive) function *sum ns* that sums a list of natural numbers: $sum[n_1, \ldots, n_k] = n_1 + \cdots + n_k$.

```
primrec sum :: "nat list => nat" where
  "sum []     = 0"
| "sum (x#xs) = x + sum xs"
```

Show that *sum* is compatible with *rev*. You may need a lemma.

```
lemma sum_append: "sum (xs @ ys) = sum xs + sum ys"
  apply (induct xs)
```

```
  apply auto
done
```

```
theorem sum_rev: "sum (rev ns) = sum ns"
  apply (induct ns)
  apply (auto simp add: sum_append)
done
```

Define a function $Sum\ f\ k$ that sums f from 0 up to $k-1$: $Sum\ f\ k = f\ 0 + \cdots + f(k-1)$.

```
primrec Sum :: "(nat => nat) => nat => nat" where
  "Sum f 0       = 0"
| "Sum f (Suc n) = Sum f n + f n"
```

Show the following equations for the pointwise summation of functions. Determine first what the expression `whatever` should be.

```
theorem "Sum (%i. f i + g i) k = Sum f k + Sum g k"
  apply (induct k)
  apply auto
done
```

```
theorem "Sum f (k + l) = Sum f k + Sum (%i. f (k + i)) l"
  apply (induct l)
  apply auto
done
```

What is the relationship between `sum` and `Sum`? Prove the following equation, suitably instantiated.

```
theorem "Sum f k = sum whatever"
```

Hint: familiarize yourself with the predefined functions `map` and `[i..<j]` on lists in theory List.

```
theorem "Sum f k = sum (map f [0..<k])"
  apply (induct k)
  apply (auto simp add: sum_append)
done
```