Before beginning to solve the exercises, open a new theory file named Ex01.thy and add the following three lines at the beginning of this file.

```
theory Ex01
imports Main
begin
```

### Exercise 1.1 Calculating with natural numbers

Use the `value` command to turn Isabelle into a fancy calculator and evaluate the following natural number expressions:

- \(2 + (2::nat)\)
- \((2::nat) \times (5 + 3)\)
- \((3::nat) \times 4 - 2 \times (7 + 1)\)

Can you explain the last result?

### Exercise 1.2 Natural number laws

Formulate and prove the well-known laws of commutativity and associativity for addition of natural numbers.

### Exercise 1.3 Counting elements of a list

Define a function which counts the number of occurrences of a particular element in a list.

```
fun count :: "'a list ⇒ 'a ⇒ nat"
```

Test your definition of `count` on some examples and prove that the results are indeed correct.

Prove the following inequality (and additional lemmas if necessary) about the relation between `count` and `length`, the function returning the length of a list.

```
theorem "count xs x ≤ length xs"
```
Exercise 1.4  Adding elements to the end of a list

Recall the definition of lists from the lecture. Define a function \textit{snoc} that appends an element at the right end of a list. Do not use the existing append operator @ for lists.

\textbf{fun} \textit{snoc} :: “′a list ⇒ ′a ⇒ ′a list”

Convince yourself on some test cases that your definition of \textit{snoc} behaves as expected, for example run:

\textbf{value} “\textit{snoc} [] c”

Also prove that your test cases are indeed correct, for instance show:

\textbf{lemma} “\textit{snoc} [] c = [c]”

Next define a function \textit{reverse} that reverses the order of elements in a list. (Do not use the existing function \textit{rev} from the library.) Hint: Define the reverse of \( x \neq xs \) using the \textit{snoc} function.

\textbf{fun} \textit{reverse} :: “′a list ⇒ ′a list”

Demonstrate that your definition is correct by running some test cases, and proving that those test cases are correct. For example:

\textbf{value} “\textit{reverse} [a, b, c]”
\textbf{lemma} “\textit{reverse} [a, b, c] = [c, b, a]”

Prove the following theorem. Hint: You need to find an additional lemma relating \textit{reverse} and \textit{snoc} to prove it.

\textbf{theorem} “\textit{reverse} (\textit{reverse} xs) = xs”

Homework 1.1  More Finger Exercise with Lists

\textit{Submission until Tuesday, October 23, 10:00am.}

\textbf{Submission Instructions}

Submissions are handled via \url{https://competition.isabelle.systems/}.

- Register an account in the system and send the tutor an e-mail with your username.
- Select the competition “Semantics” and submit your solution following the instructions on the website.
- The system will check that your solution can be loaded in Isabelle2018 without any errors and reports how many of the main theorems you were able to prove.
- You can upload multiple times; the last upload before the deadline is the one that will be graded.
• If you have any problems uploading, or if the submission seems to be rejected for reasons you cannot understand, please contact the tutor.

General hints:

• If you cannot prove a lemma, that you need for a subsequent proof, assume this lemma by using sorry.

• Define the functions as simply as possible. In particular, do not try to make them tail recursive by introducing extra accumulator parameters — this will complicate the proofs!

• All proofs should be straightforward, and take only a few lines.

Define a function fold_right that iteratively applies a function to the elements of a list. More precisely fold_right \( f \ [x_1, x_2, \ldots, x_n] \ a \) should compute \( f x_1 (f x_2 (\ldots (f x_n a))) \). The following evaluate to true, for instance:

\[ \text{value} \quad \text{"fold_right (+) [1,2,3] (4 :: nat) = 10"} \]
\[ \text{value} \quad \text{"fold_right (#) [a,b,c] [] = [a,b,c]"} \]

Prove that fold_right applied to the result of map can be contracted into a single fold_right:

\[ \text{lemma} \quad \text{"fold_right f (map g xs) a = fold_right (f o g) xs a"} \]

Here \( o \) is the regular composition operator on functions, i.e. \( f o g = (\lambda x. f (g x)) \).

Prove the following lemma on fold_right and append:

\[ \text{lemma} \quad \text{"fold_right f (xs @ ys) a = fold_right f xs (fold_right f ys a)"} \]

For the remainder of the homework we will consider the special case where \( f \) is the addition operation on natural numbers. Prove that sums over natural numbers can be “pulled apart”:

\[ \text{lemma} \quad \text{"fold_right (+) (xs @ ys) (0 :: nat) = fold_right (+) xs 0 + fold_right (+) ys 0"} \]

The notation \( (+) \) is just a shorthand for \( \lambda x y. x + y \).

Finally prove that calculating the sum from the right and from the left yields the same result:

\[ \text{lemma} \quad \text{"fold_right (+) (reverse xs) (x :: nat) = fold_right (+) xs x"} \]

You may need a lemma about snoc and fold_right.