Prof. Dr. Ralf Hinze
M.Sc. Sebastian Schweizer

**TU Kaiserslautern**
**Fachbereich Informatik**
**AG Programmiersprachen**

# Functional Programming: Exercise 1

Sheet published: Wednesday, April 17th
Submission deadline: Tuesday, April 23rd, 12:00 noon
Registration deadline: Thursday, April 18th, 12:00 noon

**Submission Instructions** After you have registered for the exercise (see task 1), you will get access to your team's Git repository on our GitLab server. Please do the submission for tasks 3 and 4 into that repository by Tuesday, April 23rd, 12:00 noon.

## 1 Register for Exercises

In order to take this course you need to register for the exercises. Please form teams of 3–4 students each. Send an email including your name, contact email address, matrikulation number, as well as the names of your teammates (names only) to Sebastian Schweizer (schweizer@cs.uni-kl.de). The registration deadline is Thursday, April 18th, 12:00 noon. If you cannot find colleagues for a team then we will arrange something during the first exercise session.

We will register an account for you on our GitLab server and create a repository for your team. You have to submit your exercise solutions for this and the following sheets into that repository. We will provide the following exercise sheets directly into your repository. Furthermore, you will get read access to an additional repository containing the lecture slides.

## 2 Tool Setup

In this course, we use the standard lazy functional programming language Haskell. In order to solve the exercises, you need a Haskell compiler. Furthermore, you need Git to submit your solutions into your team's repository and to obtain the lecture material. Please familiarize yourself with these tools as described in this section.

### 2.1 Glasgow Haskell Compiler (GHC)

We will mostly work with the interactive environment of GHC, in which Haskell expressions can be interactively evaluated and programs can be interpreted. The following steps guide you to install GHC and open the interactive environment (GHCi). After you have suceessfully started the interactive environment, see task 3 on how to use it.

#### 2.1.1 macOS

Install GHC from Homebrew: `brew install ghc`.
To start the interactive environment, run `ghci` from a terminal.

### 2.1.2 Linux

Have a look into your distribution's official package repository. You will probably find GHC there, for example:

- Arch Linux: `sudo pacman -S ghc`
- Debian, Ubuntu: `sudo apt-get install ghc`
- Fedora: `sudo dnf install ghc`
- RHEL, CentOS: `sudo yum install ghc`

Otherwise, download and install GHC from `https://www.haskell.org/ghc/`.
To start the interactive environment, run `ghci` from a terminal.

### 2.1.3 Windows

Download and install the Haskell Platform from `https://www.haskell.org/platform/windows.html` (use the "Core" version). To start the interactive environment, open a terminal (press Windows key + `R`, run `cmd`) and run the command `ghci`.

## 2.2 Git and GitLab

Git is a distributed version control system. We use one Git repository per team to manage the exercise submissions. This repository is also a good place for you to collaborate with your teammates. Additionally, there is a separate repository containing the lecture slides and supportive material.

If you have not yet installed Git on your computer, you can find installation instructions in the official Git Book[1]. In case you prefer a GUI over the command line tool, you might want to install SmartGit[2]. It can be used free of charge for non-comercial usage. An introduction to the command line tool can be found in the Git tutorial[3].

We have our own GitLab server, located at `https://pl-git.informatik.uni-kl.de`. A central copy of your team's repository will be stored there. Once you push your commits to that server we will be able to grade your exercise solutions. After you have registered to the exercises (see task 1), we will create an account for you and a repository for your team. You will receive an email with the subject "Account was created for you" directly from our GitLab server. It contains a link to set your password. You can then login using your email address and the password you just have specified.

When using Git (command line or GUI), you can authenticate either using your email address and GitLab password, or with your private ssh key. To use email and password, clone the repository using its https url (starting with `https://pl-git`) instead of the ssh url (starting with `git@pl-git`). To use ssh key authentication, please setup your public ssh key in GitLab[4] and clone the repository using its ssh url. You can find both urls on the "Project → Details" page of the project in GitLab.

[1] `https://git-scm.com/book/en/v2/Getting-Started-Installing-Git`
[2] `https://www.syntevo.com/smartgit/`
[3] `https://git-scm.com/docs/gittutorial`
[4] `https://pl-git.informatik.uni-kl.de/profile/keys`

# 3 Introduction to GHCi

The interactive environment (GHCi) interactively evaluates Haskell expressions and interprets programs. Open the GHCi as described in task 2.1 and wait until you see a prompt `Prelude>`. Now you can type any expression and it is immediately evaluated. Try it out with some mathematical expressions, e.g. `4+5*6-7`. Now close the GHCi session by typing `:quit` (or the shortcut `:q`).

One typically uses a text editor to write or edit a Haskell script, saves that to disk, and loads it into GHCi. Create a new file `example.hs` with the following content:

```
-- compute the square of an integer
square :: Integer -> Integer
square x = x * x

-- smaller of two arguments
smaller :: (Integer, Integer) -> Integer
smaller (x, y) = if x <= y then x else y
```

Open a terminal, `cd` to the directory containg the just created file, and run `ghci example.hs`. This loads and interprets the file, you should see:

```
[1 of 1] Compiling Main             ( example.hs, interpreted )
Ok, one module loaded.
*Main>
```

You can now use the functions from your `example.hs` file. Try out e.g. `square 12` and `smaller (42, 44)`. To load a script, you can either give the script as parameter to `ghci` as just explained, or you can type `:load example.hs` into an active GHCi session (shortcut for `:load` is `:l`). To use the `:load` command, ensure that you run GHCi from the directory containing the script or provide an absolute path to your script. When the script is loaded in GHCi and you made some changes to the script file, type `:reload` (or `:r`) to reload your script.

**Submission for this task** Submit two files into the `ex1` folder in your team's repository:

- the `example.hs` file created above
- Enter the expression `map pred "Ju!xpslt"` in GHCi and save the output to a file named `output.txt`.

## 4 Writing your first Haskell Function

Create a file `shapes.hs` containing the following functions:

```
areaRectangle :: Integer → Integer → Integer
areaRectangle width height =


perimeterRectangle :: Integer → Integer → Integer
perimeterRectangle width height =


areaCircle :: Float → Float
areaCircle radius =


perimeterCircle :: Float → Float
perimeterCircle radius =
```

Fill in the gaps (i.e. after each = symbol) such that the functions calculate the area respectively the perimeter of a rectangle or circle with the given dimensions. Note that you can use the constant *pi* which is already defined. Try out your functions in GHCi and submit the file to the repository.

## Final Remarks

**First Exercise Session** The first exercise session is on Thursday, April 18th, 11:45am, Room 48-453. Please come there if you have not yet found a team or you have any technical issues. Nothing else will be discussed in that session (you can of course ask questions).

**Next Exercise Sheets** The next exercise sheets will be provided directly into your team's repository (on Wednesdays). Use `git pull` to fetch these updates. We will also provide skeleton files for some exercises such that you only need to fill in some parts.