CS-205 Declarative Programming October 3rd 2023
Cécilia Pradic, Monika Seisenberger

Lab 1: Getting started with Haskell

The purpose of this lab is to get you started with haskell. Unfortunately, there are a couple of issues with the current setup in the labs this year. As a result, we would like to encourage you if possible to use haskell on your machine. Installation advice may be found on canvas.

If you cannot, you can use the lab machine but as of time of writing, the new haskell bundle (labeled with the modern **purple** haskell logo) is not working. You want to use **Hugs** instead, which can also be found in the specialist apps folder for computer science[1]

- Feel free to ask questions about setting up haskell on your machine
- Note that signing off is also an opportunity for you to ask questions if anything still feels unclear. We are here to help.
- Before calling a lab demonstrator for signing-off, do load your code through `ghci`/`hugs` to check for errors!
- **Please**, to edit code, please use a text editor that has at least syntax highlighting for Haskell, both for your own sake and the demonstrators'. On the lab machines, as of now, you can use `Notepad++` to edit plaintext/Haskell files. You can use fancier editors, but please try to avoid `Notepad`.
- Attempt to do all lab tasks that are not explicitly labelled as **challenge**.
- You can check the file `lecture1.hs` on canvas for inspiration
- The labs are meant to be signed off in the week they are given, but if you do not have time to get it signed off at the end of a given session, we will sign it off at the begining of the subsequent week without asking for justifications.
- Please do not share any solutions, but give every student the opportunity to solve the tasks themselves.

Task 1.1 evaluating expressions Open `ghci` or `hugs` and let us try to evaluate some expressions as we did during the lecture. What are the values of

a) `5 + 4`, `4 / 3`, `4 == 3`, ``15 `div` 6``, `div 15 6` (arithmetic operations)

b) `tail [1,3,5]`, `[1..3] ++ [-1,2] ++ [2,5]` (list operations)

c) `"abc" ++ "77"`, `'c' : "ab"` (strings are lists of characters)

Reflect and try to guess what the different operators are doing.

[1]It is an older interpreter and does not have as advanced features as `ghci`, but it will work for our purpose.

Tip. *Regular functions that take at least two arguments can be turned into binary operators, like we did for* `mod` *above, by surrounding the function name by backticks* `` ` ``*.*
In the other direction, surrounding an operator like `+` *by parentheses to obtain* `(+)` *turns it into a function, i.e.,* `(+) 1 2` *is just another notation for* `1 + 2`*.*

Task 1.2. Load your first program Next, to be able to write programs, you need to create a file. We would recommend you start organizing a folder for CS-205 that would contain a file per lab.

Create a file called e.g. `lab1.hs` using your favorite editor (`Notepad++` if you don't have one), double-check that you have not called that file `lab1.hs.txt` and type the following function

```haskell
square :: Float -> Float
square x = x * x
```

To load an existing Haskell file, there are a couple of options that depends on what you are using

- If using `hugs` on the lab machines, you may simply use the `File` menu and `Open`; that will load your file
- in the prompt (of either `ghci` or `hugs`) you may type `:l <full path>\lab1.hs` to load your file
- in the prompt you may alternatively use `:cd` to change the current path to the folder where `lab1.hs` is located, and then type `:l lab1.hs`
- if launching `ghci` from the command line, you can pass the file as an argument and `ghci` will launch it

Tip. *Some useful special commands your prompt has:*

- `:?` *lists all the special commands*
- `:r` *reloads the files that you had loaded*
- `:t` *checks the type of an expression*
- *(*`ghci` *only, advanced users)* `:cmd bla` *execute* `bla` *in your shell*

1. What is the result of `square yourStudentNumber`? Include the Haskell call and response as a comment in your file.

   Tip. *There are two types of comments in Haskell:*

   - *if you write* `--`*, the rest of the line is interpreted as a comment*
   - *you can open and close a multiline comment with* `{-` *and* `-}` *respectively*

2. Complete the following snippet

   ```
   average :: Float -> Float -> Float
   average x y = {- complete the body of this declaration -}
   ```

   so that `average` computes the average of the two numbers `x` and `y`.

3. **Challenge task:** Complete the following function definition

   ```
   variance :: Float -> Float -> Float
   variance x y = {- complete that -}
           where mean = {- complete that -}
                 xd   = x - mean
                 yd   = y - mean
   ```

   that computes the variance[2] of the distribution given by the two numbers.

Task 1.3 Write some list functions

Implement the following functions in your file

1. `second :: [Int] -> Int` which returns the second element of a list
2. `repeat3 :: [Int] -> [Int]` which repeat its input three times. For instance, `repeat3 [1,2]` should be `[1,2,1,2,1,2]`
3. `shiftRightByOne :: [Int] -> [Int]` which takes out the first element and append it at the end of its input.
4. `firstHalf :: [Int] -> [Int]` and `secondHalf :: [Int] -> [Int]` that map a list to its halves. For instance, we would have `firstHalf [1..4] == [1,2]` and `secondHalf [1..4] == [3,4]`

To do so, you may use any of the functions of the standard library which are listed below with their types[3].

```
head :: [Int] -> Int              -- extract the first element
tail :: [Int] -> [Int]            -- drop the first element
length :: [Int] -> Int            -- length of the list
reverse :: [Int] -> [Int]         -- reverses the list
(!!) :: [Int] -> Int -> Int       -- access an element at a given index
(++) :: [Int] -> [Int] -> [Int]   -- concatenate two lists
take :: Int -> [Int] -> [Int]     -- keep an initial segment of the list
drop :: Int -> [Int] -> [Int]     -- drop an initial segment of the list
```

Tip. *Don't hesitate to use the interactive prompt to check that your solutions work as you expect them to!*

[2] `https://en.wikipedia.org/wiki/Variance`; essentially take the distance to the average, square them, and take the average.

[3] This is not their actual types, as you can check using your interpreter - but you can pretend it is for this exercise. The actual types are *more general*.