Lab Sheet 4

List Comprehensions

Functional Programming - Winter 2022/2023 - November 16, 2022 - Schupp/Lübke

How to succeed with the labs and exercises?

Labs and exercise sheets are published every week on the course homepage at StudIP. As described in the first lecture, each successfully completed lab and exercise earns you bonus points towards your final score in this semester's exam. Keep in mind that you only get bonus points if you would pass the exam without them. **Cheating does not help you - but we will!**

How to complete a lab successfully?

In the lab, you will solve the tasks on lab sheet. You are encouraged to talk to your neighbors and find solutions together, as well to ask the tutor for help. **Towards the end of the session, the tutor will briefly discuss your solutions with you. To pass the lab, you should complete two thirds of the tasks (rounding half up).**

How to complete an exercise successfully?

In order to complete an exercise sheet successfully, you must upload your answers using INGInious **before the deadline** printed on the exercise sheet. We will not consider any solutions handed in after the deadline! Furthermore, you must solve and hand in the exercises **individually** and your Haskell code **must compile** and **pass certain amounts of tests** as specified. During the exercise session, we develop possible solutions together. Please participate! We encourage you to ask and answer questions from fellow students.

Technically, Haskell files you submit using INGInious must have the format as specified in the task sheets (usually ".hs", ".lhs", or ".txt"). Furthermore, INGInious will only consider your last submission. Therefore, if you first submit successfully (your code compiles and tests are passed) and afterwards unsuccessfully (your code does not compile or certain tests fail again), your last submission counts, and - if it does not compile - will therefore be ignored. Make sure your last submission was successful!

How to get additional information?

We encourage you to discuss past and present exercise sheets with us. Either approach us during the exercise session, or visit us during the weekly office hours. We are also available via e-mail or on the StudIP forum. We try to reply as quick as possible and in general, you should get a reply the next weekday, but we cannot guarantee this.

Task 1 Write the following function definitions to a file (or use the enclosed *Errors.hs*).

```
add :: Int -> (Int -> Int)
add = \x -> y -> x + y

myLength :: [a] -> Int
myLength xs = sum [1 : x <- xs]

odds :: Int -> [Int]
odds n = [i | i `mod` 2 == 1, i <- [1..n]]

quadruple :: Int -> Int
quadruple x = (mult 2 x) * 2
  where mult 0 _ = 0
mult n m = m + mult (n-1) m
```

Load the file using GHCi (`:load`), look at the error messages, understand the problem and fix all errors. Repeat until the file loads with no errors. One error is **not** caught by GHCi. The function `quadruple` will fail when you use it. Why is that?

Task 2 Are the definitions `repeatc` and `repeatn` legal. If so, are they equivalent?

```
repeatn :: Int -> [a] -> [a]
repeatn n xs = [x | x <- xs, i <- [1..n]]

repeatc :: Int -> [a] -> [a]
repeatc n xs = [x | i <- [1..n], x <- xs]
```

Task 3 Using the `prime` function from the lecture, implement the function `numprimes` using list comprehension. The purpose of `numprimes` is to determine the number of primes smaller than or equal to its argument.

Task 4 Define the function `scalarProduct`, which takes two integer lists of the same length and calculates the sum of the products.

Task 5 Assuming we have the following nested list of integers, and we want to *remove* every *even* integer from the list. Implement a function `nestedRemoveEven` to do exactly this.

```
Prelude> nestedRemoveEven [ [1,2,3], [4,5,6], [7,8,9] ]
[[1,3],[5],[7,9]]
```