Exercise 1

NTNU

TDT4165 fall 2018

1 Theory solutions

- What is the difference between a null pointer reference (e.g. null in java) and **None :: Maybe a**?
  None is not a member of any other types than Maybe a, while Java's null is a member of every object type (but not primitive data types). This means that our compiler will react if we try to use e.g. a None :: Maybe Int as an Int.

- Is $ a keyword or a function? What does it do?
  (hint: try typing **:t ($)** in GHCi)
  $ is a function (a -> b) -> a -> b. It is used as an operator, has low, right-associative precedence and allows us to avoid using parentheses. **show (5*2)** can be rewritten as **show $ 5*2**. The low precedence of $, ensures that 5*2 is calculated first.

- Explore (.) the same way. What is the difference between (.) and ($)?
  (.) :: (b -> c) -> (a -> b) -> a -> c is also a function, but serves a different purpose. It allows us to compose functions.

- Try running **take 50 $ map fib [1..]** where fib is the unoptimized fibonacci function from exercise 1.
  - What does this code snippet do?
    The right hand side gets evaluated first, creating fibonacci numbers from an infinite list. On the left hand side, we are specifying that we want 50 numbers from this list.
  - Do you get the whole result at once? Why or why not?
    Running this in GHCi, means that we are also asking for the result to be printed. Because Haskell is lazy, evaluation is done when the values are needed. A number runs through the pipeline through map, then take and is printed when it's fully evaluated.
  - What happens if you run the Python code below? What changed and why?
    Python has eager evaluation, our code is executed where it is defined.

The print statement is not evaluated before **res** is fully populated. Therefore, we get the whole list at once.

- In what situations can infinite lists be useful?
  Infinite lists are particularly useful in situations where we do not know how many elements of a list we are going to need, for example when iterating towards a value.

- Try typing **:t (,)** into GHCi. What is the type of (,) and what does this mean?
  (,) :: a -> b -> (a,b). (,) is a tuple constructor.

- How would you go about creating a curried function in Python? Is it possible? It is possible to create curried functions in any language with first class functions. In Python, you can curry a two-argument function by returning a one-argument lambda expression from it.

- Point-free style is a way of programming without mention of the arguments we are working with. Why is this possible in Haskell? Point-free style is a property made possible by the desugaring to one-argument lambda expressions, in combination with functions being first-class.