



Assignment 1

Hand in this assignment until **Tuesday, 06 May** at the latest.

🤔 Running out of ideas?

Are you hitting a roadblock? Are some of the exercises unclear? Do you just need that one hint to get the ball rolling? Refer to the [#forum](#) channel on our Discord server—maybe you'll find just the help you need.

💡 Rules for this and all future assignments

- In general, the only acceptable file format is plain text (*.txt, *.hs for Haskell code). Files in other formats are not graded, unless explicitly stated differently.
- All code you submit must compile. Code that does not compile (in particular: does not typecheck) might not be graded.
- Please submit code that is nicely and consistently formatted and well-documented¹. Every top-level function definition has to include a type signature and a comment.

📖 Exam-style Exercises

Exercises marked with 📖 are similar in style to those you will find in the exam. You can use these to hone your expectations and gauge your skills.

Task 1: Please answer the following questions about Haskell's type system 📖

A Consider the following types:

- $a \rightarrow b \rightarrow c \rightarrow d$
- $a \rightarrow (b \rightarrow c) \rightarrow d$
- $a \rightarrow b \rightarrow (c \rightarrow d)$

Which pairs of types are equivalent and which are not? Explain.

B Can you give multiple definitions of a function of type $(a, b) \rightarrow a$ that **behave differently**, that is, return different values for the same argument? Explain briefly. Assume that your function actually has to return a value (*i.e.* no crashes, no infinite loops and recursions).

C Consider a function of type $[a] \rightarrow a$. Recall that a represents an *arbitrary* type—the function must thus work correctly for lists over *any* element type. Could it be a function which

- ... returns the largest element of the list?
- ... computes the sum of all list elements?
- ... returns a *constant* value?
- ... performs I/O operations (*e.g.* prints a value to the terminal)?

Explain your answers.

D The following function is supposed to extract the first character from a given string.

```
1 getFirstLetter :: [Char] -> [Char]
2 getFirstLetter s = head s
```

Fix any type errors.

¹To have an idea of “nicely formatted code”, you can find a short style guide here: <https://github.com/tibbe/haskell-style-guide/blob/master/haskell-style.md>.

- E Given the functions `fst :: (a, b) -> a` and `snd :: (a, b) -> b`, derive the type of the following expression:

```
1 |snd . snd . fst
```

Task 2: Finger exercises

- A Define an infix operator that implements logical implication. You can use the Boolean operators `(&&)`, `(||)`, `not`, or a conditional expression `(if e1 then e2 else e3)`. The new operator's precedence should be less than the three Boolean operators' above. Please give some example expressions to show this behavior.

```
1 |(<=>) :: Bool -> Bool -> Bool
```

- B Define a function `distance` to calculate the Euclidean distance between two Points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$.

```
1 |distance :: (Double, Double) -> (Double, Double) -> Double
```

Hint

To avoid repetitive code or extract meaningful code pieces it can be very useful to [introduce local definitions](#) which can be used in the function body. In Haskell such definitions are declared with the keyword `where`. For example if we have:

```
1 |perimeter :: (Double, Double) -> Double
2 |perimeter rectangle = 2 * (fst rectangle) + 2 * (snd rectangle)
```

we can, instead, write

```
1 |perimeter :: (Double, Double) -> Double
2 |perimeter rectangle = double width + double height
3 |   where
4 |       width = fst rectangle
5 |       height = snd rectangle
6 |       double n = 2 * n
```

Try to use `where`-definitions in your solution for `distance`.

- C Write a function `gcdEuclid i j`, such that computes the greatest common divisor of two integers $i, j > 0$ using Euclid's algorithm².

```
1 |gcdEuclid :: Int -> Int -> Int
```

Include a brief comment on how your implementation would behave if parameters `i` or `j` are ≤ 0 .

- D If `gcd(i, j) = 1` for integers `i` and `j`, then `i` and `j` are called *coprime*. Write a Haskell function `coprime` to determine if two integers are coprime.

```
1 |coprime :: Int -> Int -> Bool
```

²https://en.wikipedia.org/wiki/Euclidean_algorithm#Implementations 

Task 3: Safe Head

Consider the Haskell function `head :: [a] -> a`, which returns the first element of a list. `head` is not able to return a value if the list is empty. Haskell would report an error at runtime.

Now imagine the function `headMaybe` with the following type:

```
1 headMaybe :: [a] -> Maybe a
2
3 > headMaybe [1, 2, 3]
4 Just 1
5 > headMaybe "abc"
6 Just 'a'
7 > headMaybe []
8 Nothing
```

`headMaybe` returns `Nothing` on failure and `Just X` on success, where `X` is the head of the argument list.

Define the function `headMaybe` so that it behaves as in the description above.

Hint

You can use the built-in predicate `null :: [a] -> Bool` to test whether a given list is empty.