

- Mark your completed exercises in the OLAT course of the PS.
- You can use a template .hs file that is provided on the proseminar page.
- Upload your modified .hs file in OLAT.
- Your .hs file must be compilable with ghci.

Exercise 1 *Rational Numbers*

5 p.

Implement rational numbers in Haskell. Here, rational numbers are represented by two integers, the numerator and the denominator. For instance the rational number $\frac{-3}{5}$ can be represented as `Rat (-3) 5` when using the following data type definition:

```
data Rat = Rat Integer Integer
```

1. Implement a normalisation function `normaliseRat :: Rat -> Rat` for rational numbers so that all of `Rat 2 4`, `Rat (-1) (-2)` and `Rat 1 2` are transformed into the same internal representation. Furthermore, implement a function `createRat :: Integer -> Integer -> Rat` that, given two `Integer`s, returns a normalised `Rat`. (1 point)
Hint: the Prelude contains a function `gcd` to compute the greatest common divisor of two integers.
2. Make `Rat` an instance of `Eq` and `Ord`. Of course, `Rat 2 4 == Rat 1 2` should evaluate to `True`. (1 point)
3. Make `Rat` an instance of `Show`. Make sure that `show r1 == show r2` whenever `r1 == r2` for two rational numbers `r1` and `r2`. In particular, `show (Rat 1 2) == show (Rat 2 4)` should evaluate to `True`. Moreover, integers should be represented without the `"/"` symbol. (1 point)

Examples:

```
show (Rat (-4) (-1)) == "4"  
show (Rat (-3) 2) == "-3/2"  
show (Rat 3 (-2)) == "-3/2"
```

4. Make `Rat` an instance of `Num`. See <https://hackage.haskell.org/package/base-4.18.0.0/docs/Prelude.html#t:Num> for a detailed description of this type class. (2 points)

Exercise 2 *Monoids*

5 p.

A **monoid** is an algebraic structure that consists of an associative binary operation \circ and a neutral element e where the following laws are satisfied for all x, y, z :

- $x \circ (y \circ z) = (x \circ y) \circ z$
- $x \circ e = e \circ x = x$

We model monoids in Haskell in the following class.

```
class MonoidC a where  
  binop :: a -> a -> a  
  neutral :: a
```

1. One instance of a monoid can be obtained by using addition on numbers as the binary operation. Moreover, lists form a monoid where the binary operation is the append function. In both instances, the choice of the neutral element can be deduced from the monoid laws.
Define the described `MonoidC` instances for `Double`, `Integer` and `[a]`. (1 point)
2. We consider extended tally lists to represent numbers, where a plus sign indicates +1, and a minus sign indicates -1. For instance, `++`, `+++` and `++---` represent the number 2, whereas `---` and `---+` represent the number -3.
Define a function to normalise such tally lists, such that after normalisation the tally lists do not contain any sublist of the form `+--` or `--+`. For instance, all tally lists that represent the number 2 should normalise to `++`. (1 point)
3. We define a datatype `Tally` with constructor `PM :: String -> Tally` to represent extended tally lists. Make `Tally` an instance of `Eq`, `Show`, and `MonoidC` (with tally list addition as the binary operation). In the latter case, make sure that the result of an addition is normalised. Moreover, `PM xs == PM ys` should return `True` whenever `xs` and `ys` represent the same number. The `show` function should just return the internal string representation. (1 point)
4. Define a function `combine` that takes a list of elements x_1, \dots, x_n in a monoid, and computes the combination of these elements, i.e., $x_1 \circ \dots \circ x_n \circ e$. The function definition should include the (most general) type of `combine`. (1 point)
5. After completing the previous tasks,
 - describe informally what the function `combine :: [Integer] -> Integer` computes,
 - describe informally what the function `combine :: [String] -> String` computes,
 - describe why `combine [[1,3,7]]` and `combine [1,3,7]` differ, and
 - explain the difference between the result of `combine [PM "+++", PM "+---+", PM "----+"]` and that of `combine ["++---+", "+----+", "----+"]`. (1 point)