

- 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 .hs-file(s) of Exercises 2 and 3 in OLAT.
- Your .hs-file should be compilable with ghci.

Exercise 1 *Pattern Matching***2 p.**

Consider the following datatype definitions:

```
data Subject = CS | Math | Physics | Biology
data Programme =
  Bachelor Subject
  | Master Subject
  | Teaching Subject Subject -- teachers need two subjects
data Student = Student
  String -- name
  Integer -- matriculation number
  Bool -- active inscription
  Programme
```

Determine which of the expressions 1.-3. match the patterns in (i) and (ii). For each match give the corresponding substitution. (2 points)

1. `Student "Jane Doe" 243781 True (Teaching Math Physics)`
2. `Student "Max Meyer" 221341 False (Teaching CS Math)`
3. `Student "Mary Smith" 234145 False (Master CS)`

(i) `Student name n _ (Teaching Math _)`

(ii) `Student name n False p@(Master _)`

Exercise 2 *Function Definitions***3 p.**

1. Define a function `disj :: Bool -> Bool -> Bool` for computing the disjunction of two Booleans. (1 point)
2. Define a function `sumList :: List -> Integer` that takes a list of integers (as defined in the lecture) and returns the sum of its elements. The sum over an empty list should be 0. (1 point)
3. Define a function `double2nd :: List -> List` that doubles every second element in a given list of integers, i.e., $[1, 7, 9, 3] \rightsquigarrow [1, 14, 9, 6]$. (1 point)

Exercise 3 *A Recursive Function*

5 p.

In this exercise, we will extend the `Expr` datatype from the lecture with variables:

```
data Expr =  
  Number Integer  
| Var String  
| Plus Expr Expr  
| Negate Expr
```

We will also need the following datatype to store variable assignments:

```
data Assignment = EmptyA | Assign String Integer Assignment
```

Here, the `EmptyA` constructor corresponds to an empty assignment in which all variables have value 0. The `Assign` constructor takes an assignment and changes the value of one variable to the given integer (see examples below).

1. Write a function `ite :: Bool -> Integer -> Integer -> Integer` such that `ite b x y` returns `x` if `b` is true and `y` otherwise. Use pattern matching on Booleans only. (“ite” stands for “if-then-else”)(1 point)
2. Write a function `lookupA :: Assignment -> String -> Integer` that returns the value corresponding to the given variable in the given assignment. (1 point)

Example: Let `myAssn = Assign "x" 1 (Assign "x" 2 (Assign "y" 3 EmptyA))`. Then:

```
lookupA myAssn "x" == 1  
lookupA myAssn "y" == 3  
lookupA myAssn "z" == 0
```

3. Write a function `eval :: Assignment -> Expr -> Integer` that evaluates the given arithmetic expression under the given variable assignment. (2 points)

Example: Let `myAssn` be as before. Then:

```
eval myAssn (Plus (Negate (Var "y")) (Number 45)) == 42 -- corresponds to (-y) + 45
```

4. In order to store auxiliary results and avoid computing the same things twice, extend the `Expr` type with a “`let x = e1 in e2`” construct, i.e. the result of `e1` is assigned to the variable `x` when evaluating `e2`, the result of which is then returned. Extend your “eval” function accordingly as well. (1 point)

Example:

You should be able to write an expression that encapsulates something like this:

$$\text{let } x = 2 + 3 \text{ in } x + x$$

How you represent this in your datatype is up to you.