CS_205 Declarative Programming 9 October 2020
Monika Seisenberger, Ulrich Berger

Lab Task 2: Types and Type classes

1) What are the types for the following expressions? Write the answer on this sheet and confirm your answer using Haskell [Hint: use the command `:type`.]

```
('a','b','c')
['a','b','c']
[('1', False), ('0',True)]
([True,False], ['0','1'])
```

2) Types: Char, Int and Bool

a) Write a function that takes as input two lists of type `Char`, and checks whether the two lists are equal.

b) Modify the function so that it checks whether the two lists are equal at a given index n. You may assume that the user enteres a "sensible" index, i.e does not produce an error here.)

c) Now improve your solution to b) so that it works or all inputs. [i.e. checks first whether the index n given is less than the lengths of the two lists.] (If this is not the case, the function should return False.)

3) What are the (most general) types for the following functions: Write and test the completed functions in Haskell.

```
a) middle (x,y,z) = y
b) swap (x,y) = (y,x)
```

Example: the most general type for the head of a list is: `head :: [a] -> a` (where a is a variable for an arbitrary type).

For the functions below you also need to check which of the constraints: `Num a, Ord a, Eq a` you need to add.

Example: The most general type of the function `double x = x+x`: is

`double :: (Num a) => a -> a`.

Explanation: double works for an arbitrary type a, as long as a is in class Num (i.e., is a numerical type such as Int, Float, etc...). We write this constraint by adding the ``(Num a) =>'' at the beginning of its type.

```
c) add (x, y, z) = x + y + z
d) ordered x y z = x<=y && y<=z
e) palinedrome xs =  (xs == reverse xs)
```

Optional task: **safetail** Consider the function `safetail` that behaves in the same way as `tail`, except that `safetail` maps the empty list to the empty list, whereas `tail` gives an error in this case. Give variants `safetail1, safetail2` using

i) a conditional expression

ii) guarded equations

Hint for question 2 and challenge task: Syntax of an if-statement:

```
isZero :: Int -> Bool
isZero n = if n==0
           then True
           else False
```

Of course this function can be written more concise without if-statement:

```
isZero :: Int -> Bool
isZero n = n==0
```

It is intended that you complete this sheet in week 2. [At least you need to get it started in week 2, and get some questions signed off.]