Serie 3 - Haskell

Exercise 1

A triangular number is the number of objects that can be arranged in a triangle as is shown in Fig. 1. This is how bowling pins, pool balls, or snooker balls are arranged. The figure shows the first six triangular numbers: 1, 3, 6, 10, 15, 21. The tetrahedral number represents the number of objects arranged in a pyramid (more precisely, a tetrahedron) built up from triangles as shown in the Figure.

The exercise:

- Write two functions to calculate the n-th triangular number, using classical recursion and tail recursion, respectively.
- Write two functions to calculate the n-th tetrahedral number, using patterns and guards, respectively. Hint: use the function from the preceding exercise.

Exercise 2

The sum of the squares of the first ten natural numbers is,

\[ 1^2 + 2^2 + \ldots + 10^2 = 385 \]

The square of the sum of the first ten natural numbers is,

\[ (1 + 2 + \ldots + 10)^2 = 55^2 = 3025 \]

Hence the difference between the sum of the squares of the first ten natural numbers and the square of the sum is 3025 − 385 = 2640.

Write a Haskell program to find the difference between the sum of the squares of the first one hundred natural numbers and the square of the sum.

OPTIONAL: Implement the same solution in your favorite non functional language and discuss the differences.

Exercise 3

In this exercise, you are going to implement a set of functions which operate on lists. Their semantics are given below.

a. Write a function `insertNode` which adds a new node to the list. The new node should be inserted before the first node with a higher value (we assume that all lists to contain numbers).

b. Write a function `deleteNodes` which deletes all nodes which satisfy a certain predicate p.

c. Write a function `removeDuplicates` which removes duplicates to get a list with nodes having unique values.

---

1This exercise is an adaption of problem 6 from [http://projecteuler.net/](http://projecteuler.net/) You can find a ton of cool problems there.
d. Write a function `sumNodes` which calculates the sum of all nodes of the list.

e. Write a mapping function `mapList` which applies to each node of the list a given function \( f \), e.g., the square function, and returns a list with the resulting values.

f. Write a function `mergeLists` which merges two sorted lists to produce one list which is also sorted.

g. Use the function from[e] to implement a sorting function `sortList` which sorts a list in ascending order. A Mergesort would be adequate in this case.

The idea is to write for each function unit tests. There is a testing framework called HUnit, very similar to JUnit. To implement and use test cases, you have to import HUnit as a module in your source file like this:

```
import HUnit
...

tests = TestList [
  (TestCase (assertEqual "error message" [1,2,2,3] (insertNode 2 [1..3]))),
  (TestCase (assertEqual "error message" [2,4,6] (deleteNodes odd [1..6]))) ]
...

run = do runTestTT tests
```

Now to run your tests, you can just call `run`.

**Hint:** work test-driven, i.e. first write appropriate tests, then code your functions until the tests are “green”.