Lab Session Software Testing 2014, Week 2 With each deliverable, indicate the time spent.

```
module Lab2 where

import Data.List
import System.Random
```

1. Write a program (in Haskell) that takes a triple of integer values as arguments and gives as output one of the following statements:

   - 'Not a triangle' ('Geen driehoek') if the three numbers cannot occur as the lengths of the sides of triangle,
   - 'Equilateral' ('Gelijkzijdig') if the three numbers are the lengths of the sides of an equilateral triangle,
   - 'Rectangular' ('Rechthoekig') if the three numbers are the lengths of the sides of a rectangular triangle,
   - 'Isosceles' ('Gelijkbenig') if the three numbers are the lengths of the sides of an isosceles (but not equilateral) triangle,
   - 'Other' ('Anders') if the three numbers are the lengths of the sides of a triangle that is not equilateral, not rectangular, and not isosceles.

   Here is a useful datatype definition:

   ```
   data Shape = NoTriangle | Equilateral
              | Isosceles  | Rectangular | Other deriving (Eq,Show)
   ```

   Now define a function `triangle :: Integer -> Integer -> Integer -> Shape` with the right properties.

   You may wish to consult `http://en.wikipedia.org/wiki/Triangle`. Indicate how you *tested* or *checked* the correctness of the program.

   Deliverables: Haskell program, test report of 1/2 A4, indication of time spent.

2. A permutation of a finite list is another finite list with the same elements, but possibly in a different order. For example, [3,2,1] is a permutation of [1,2,3], but [2,2,0] is not. Write a function

   ```
   isPermutation :: Eq a => [a] -> [a] -> Bool
   ```

   that returns `True` if its arguments are permutations of each other.

3. Define some testable properties for this function, and use a number of well-chosen lists to test `isPermutation`. You may assume that your input lists do not contain duplicates. What does this mean for your testing procedure?

4. Write a Haskell function `perms` that generates a list of all permutations of a given list without duplicates. (Compare Exercise 6.28 in *The Haskell Road*.) What do you know about the number of permutations of a list without duplicates? How can you use this knowledge to test your implementation?

5. A derangement of the list $[0..n-1]$ of natural numbers is a permutation π of the list with the property that for no x in the list $\pi(x) = x$. This is what you need if you prepare for 'Sinterklaas' with a group of friends, where you want to avoid the situation that someone has to buy a surprise gift for him- or herself.

   Give a Haskell implementation of a property `isDerangement` that checks whether one list is a derangement of another one.

6. Give a Haskell implementation of a function `deran` that generates a list of all derangements of the list $[0..n-1]$.

7. Define some testable properties for the `isDerangement` function, and use some well-chosen integer lists to test `isDerangement`.

8. Bonus question. Find out how to use the System.Random library, and write a Haskell function that produces an arbitrary derangement for a given list. You can use this for your next Sinterklaas surprise party. Does it follow from your previous tests that your function performs as expected? If your answer is 'yes', explain your reasoning. If your answer is 'no', write some appropriate tests and carry them out.

9. Bonus question. Can you find a recurrence (a function f defined by recursion) for the *number of derangements* of a list of n distinct elements?

   The function f can be tested as follows: `map f [0..n]` should look the same as `[ length $ deran k | k <- [0..n] ]`.