Rock - Paper - Scissors

In the Rock, Paper and Scissors game, two players choose one of the following gestures after counting to three:

- **A clenched fist** which represents a rock.
- **A flat hand** representing a piece of paper.
- **Index and middle finger extended** which represents a pair of scissors.

The result of a round is decided this way:

- **Rock defeats scissors**, because a rock will blunt a pair of scissors.
- **Paper defeats rock**, because a paper can wrap up a rock.
- **Scissors defeat paper**, because scissors cut paper.
- **Otherwise**, the players chose the same gesture and it's **a draw**.

1. Moves

- Define a datatype `Move` with three choices `Rock`, `Paper` and `Scissors` that represent the valid moves. Note that the "`deriving (Eq, Show)`" should not be removed from the declaration otherwise the testing framework won't work.
- Write a function `beat :: Move -> Move` such that `beat m` is the move that beats move `m`.
- Write a function `lose :: Move -> Move` such that `lose m` is the move that will lose against move `m`.

2. Playing The Game

- Define a datatype `Result` that represents the outcome of a round of Rock - Paper - Scissors. As we explained above, a player can either `Win`, `Lose` or the game may end up in a `Draw`. Like before, the "`deriving (Eq, Show)`" should not be removed from the declaration otherwise the testing framework won't work.
- Write a function `outcome :: Move -> Move -> Result` that takes as arguments two moves (the first argument is the move of the first player and the second is the move of the second player) and calculates the outcome for the **first player**.

NOTE: Since defining the data types is part of the exercise and their definition is thus not known to the testing framework (E-Systant), the tests for this exercise **only partially check the correctness of your solutions, less accurately than for other exercises.** Hence, be extra careful when defining functions `beat`, `lose` and `outcome`.