1. For evalDeer, as there are 5 types of DeerLang expressions stated in the handout, we matched with the 5 types separately following the instructions stated in the handout. We treat the type (Builtin String [Expr]) and (Apply Expr [Expr]) differently by calling on different builtinHelper and applyHelper respectively (both the helpers break the expression down).

   For doGetEnvironment, we check if the input list of Definitions is empty and just return the input environment, then we loop through the list of Definitions and call getEnvHelper to insert the current Definition into the environment and return the environment back to doGetEnvironment.

   For buildDataEnvs, we check if the input list of columns is empty and just return the list of input environment, then we get the length of one of the columns and set the return list of environment with the same size. Once getting the environment size built we call buildDataEnvsHelper and loop through the list of input columns using zipWith to insert the values from the column into the respective environment to create our environment of rows.

2. The built-in helpers in Haskell were very helpful, for example zipWith. Also pattern matching was very helpful in completing the project.

   The challenging part for this assignment when I was working on the helper function buildDataEnvs, I tried to see if I could loop through the input list of columns with doing a for loop (as in how we could do it for an imperative language). After struggling with that I asked my partner to see if he had any ideas, he suggested me doing it with list comprehension. I found using list comprehension to loop through a list very useful.

3. I worked with a partner on this project, I completed the helper function buildDataEnvs, completed computeSpreadsheet and computeSpreadsheetHelper with my partner together online with screen sharing. Lastly, I wrote testcases for computeSpreadsheet.