**COMP2212 Coursework Report**

**Language features**

* What/why the language does what it does
* Error messages
* Own instances of EQ, ORD, SHOW

**Syntax**

The syntax for our language is as follows:

*(NB:* n *is any integer value and* x *is any string value)*

**Let ::=** let x := Expr in Let | Expr

**Expr ::=** Expr = Call | Expr != Call | Expr < Call |

Expr <= Call | Expr > Call | Expr >= Call |

Expr ++ Call | Call

**Call ::=** Call Literal | Literal

**Literal ::=** ( Expr ) | [ List ] | \\ x -> Let | n | x

**List ::= *empty*** | Literal List

Much of the syntax is self-explanatory and similar to that of the Toy language. The scope for values instantiated by let blocks is all of the code that follows the let block. Expr mostly consists of comparison operations which were not required for any of the problems but we added for some extra flexibility in our language. The append operator (++) however was useful for our solution to problem 5. Literal allows for bracketed expressions, lambda expressions (\\), and also lists ([]) which can either be empty or can contain Literals. A list would model a row in the CSV file, hence we could use it for example with the append operation to add to the output.

* Syntactic sugar?
* Lexing rules

**Interpreter Execution Model**