## 2.4 Validation

### Validation Features

Many of the features that will be included in my project require very specific inputs. Therefore, I will have to implement many validation rules for each feature.

For example, I have many features that require an input between a specific range. An input that is outside the range could cause an error which could cause the program to have an unrecoverable crash. Therefore, I will include code that validates the user’s input on a case by case basis when each feature is used.

This validation will include things such as checking that the user’s input is within the correct range. For example, with the inverse sine and cosine functions, the input must be within -1 and 1. If the input is outside of this range, a message will be shown to the user detailing the message and how to fix it.

The message to the user will be shown instead of running the function with the invalid input. Therefore, the crash will be prevented and the user can fix their error.

Another potential error is from the user inputting an incorrect expression. For example, having multiple operators that don’t apply to a variable. E.g.: “3+\*2”. This will cause an error because the “+” operator is trying to perform an operation on “3” and “\*”. The data types are incompatible and therefore it will cause an error.

This can be prevented using validation rules that will not allow operators adjacent to each other in the expression. However, exceptions must be made to the minus operator to allow for negative numbers to be used.

When this error is caught by the validation rules, a message will be shown to the user that explains the error. Also, the cursor will be placed at the point where the error occurred to make it easier for the user to find their mistake and correct it.