# 4. Evaluation

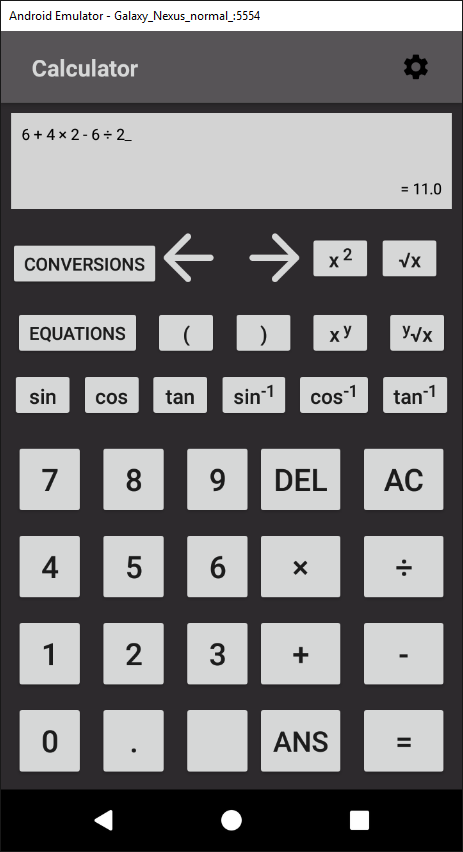
## 4.1 Evaluation of Success Criteria

### 4.1.1 Success Criterion 1

* “Functionality for the four basic operators, +, -, x, ÷.”

This success criterion was the basic building blocks of the entire app. It was designed to ensure that the user could perform basic tasks with the calculator.

My solution fully meets this criterion as shown in the screenshots below:



### 4.1.2 Success Criterion 2

* “The calculator will follow the rules of BODMAS.”

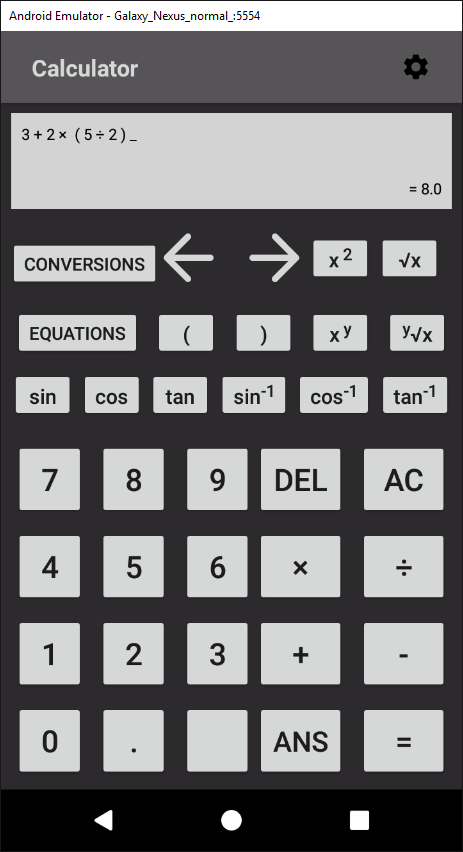
BODMAS is the order of precedence that operators in an expression should be calculated in. From highest priority to lowest, it goes: brackets, orders, division, multiplication, addition, and subtraction.

This was implemented into my solution by calculating the precedence of the operator by retrieving the index of the operator in this string: “-+÷×^√sctzef” (where “sctzef” refer to the trigonometric functions) and dividing it by two.

In the *ShuntingYard* algorithm, when the program is deciding which operator to calculate first, it compares their precedence and completes the operator with the highest precedence first. This means that the trigonometric functions and shows how the order of indexes follow BODMAS.

However, brackets are not included in this method of defining precedence. They are handled differently within the algorithm because the brackets create a sub-expression within the expression which must be completed first.

The success criterion is shown as fully met below:



This screenshot proves the success criterion is fully met because if it didn’t follow BODMAS, the result would be 12.5:

1. 3 + 2 = 5
2. 5 \* 5 = 25
3. 25 / 2 = 12.5

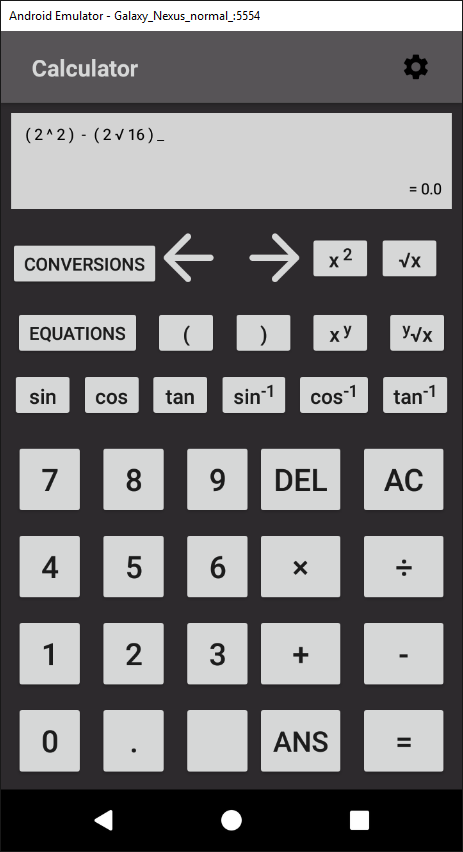
As the correct answer of 8 was displayed to the user, the success criterion 2 is completed.

### 4.1.3 Success Criterion 3

* “Have functionality for using square roots and powers.”

The solution met this success criterion completely. It was implemented by including the root and power operators into the *ShuntingYard* class and adding buttons to the *Calculator* class.

The success criterion is shown as fully met in the screenshot below:



This is because 22 equals 4 and the square root of 16 also equals 4. Therefore, when subtracting them from each other the correct result is 0 which is what the calculator displays.

### 4.1.4 Success Criterion 4

* “Be able to use sin, cos, and tan and the inverses of.”

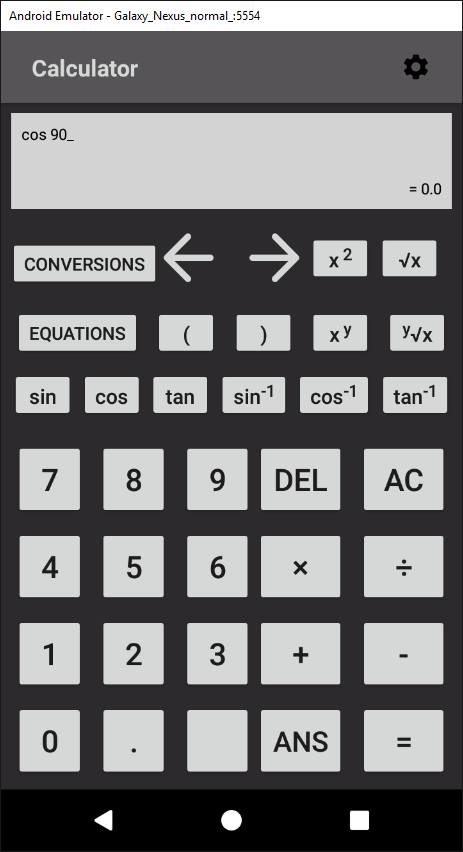
This success criterion was completed by adding operators that corresponding to each of the trigonometric functions. For example, the sine functions operator was “s”.

In the *ShuntingYard* class, each of the operators was added the the *Sign* enum and the operator string used to find precedence. In the enum, the result of the operators could be calculated by using the Java *Math* library’s trigonometry methods.

When using the trigonometry functions, the user inputs a single character operator that represents the trigonometric function. Therefore in the *updateDisplay* method in the *Calculator* activity, the expression is split up into an array and iterated through each character. When the loop encounters a trigonometry operator, it replaces it with the corresponding function. For example, if the loop encountered “c”, it would be replaced with “cos”.

This ensures the quality of the user interface and experience because leaving the operator as a single character would make the user confused and the app would less usable.

The proof for success criterion 4 is shown below:



This shows the criterion as being met as the calculator outputs the correct result for cos(90).

### 4.1.5 Success Criterion 5

* “Cursor controls that will move the cursor around the equation.”

This success criterion was met by adding two buttons (shift left and shift right) which called the *shiftPosition* method in the *Calculator* activity when called. A new variable called *mPosition* was also added.

The method *shiftPosition* altered the *mPosition* variable when by different values depending on which button called it. For example, if the shift right button was clicked, *mPosition* was increased. The value it was increased by depended on the characters it was shifting through. This is because the method had to accommodate for whitespace, multi-digit numbers, etc.

When the user input a value into the expression, a *StringBuilder* was used to insert the input into the expression at the index of *mPosition.* This ensured that the user could input numbers and operators anywhere in the expression.

In the method *updateDisplay*, an underscore was inserted into the expression at the index of *mPosition* to act as a position marker. This made the app much more usable as it enabled the user to know where they were inputting their values.

The proof for this success criterion is shown below:



This shows that success criterion 5 is complete as the position marker is displayed in the middle of the expression, showing that the user can move through the expression and alter it.

### 4.1.6 Success Criterion 6

* “Capability for common forms of notation, such as standard form, fractions, decimals.”

The default display of the calculator is to be outputted as a decimal. This is because the result is calculated as a *double* and displayed to the user by converting the *double* to a *String.*

Scientific notation (or e notation) was also included. This is done by Android without any developer input as it automatically converts a number to scientific notation by default.

Proof for this is shown below:



This increases the usability of the app as it makes results much more readable when they are larger.

However, this success criterion is only partially met because fractions were not implemented. This is because due to time constraints they could not be included. In future iterations of the application, I would add a button to the *Calculator* activity where the user could switch between each notation.

### 4.1.7 Success Criterion 7

* “I will use a survey that asks people that have used the application to rate the usability and intuitiveness of the GUI. If at least 80% of the answers give positive feedback, I will consider this success criteria completed.”

SURVEY

### 4.1.8 Success Criterion 8

* “Displaying tips when the user access more niche and unknown features.”

This success criterion was not implemented. This was because the results of the survey showed that users rated the learnability of the app very highly. Therefore, I decided that creating and implementing tooltips was not a priority and development time could be better spent creating and extending more useful features.

4.1.8 Success Criterion 9

* “Store equations for subjects such as Maths, Physics, Chemistry, etc.”

This success criterion was implemented by including a button in the *Calculator* activity that linked to a *RecyclerView* showing all the possible equations. (As shown in Appendix 5.2.2.).

Each element of the *RecyclerView* is clickable and takes the user to a specific activity where they can calculate the equation. (As show in Appendixes 5.2.3 - 5.2.7).

Each equation activity has multiple input buttons where the user can input their values and a display for the user to see their equation. Depending on which values the user leaves blank, the activity can then calculate the user’s result using the equation’s formula.

For example, if the user enters a=3, b=4 in the Pythagoras’ Theorem equation activity, the app which calculate the result using the Pythagoras’ Theorem formula and return the value for the hypotenuse, 5.

However, this success criterion is not fully complete as I decided not to include any Physics, Chemistry or Biology equations. This is because at GCSE level most required equations for these subjects are different variations of the general formula: .

Therefore, adding different elements for each equation to the *Conversions* activity would result in a lot of duplicate code and clutter the user interface. This would result in the solution becoming much less efficient and usable.

Furthermore, the general formula, is very simple and almost every student wouldn’t require a calculator to solve. Therefore, adding the equation to the calculator would be a waste as the feature wouldn’t be used and would become redundant.

### 4.1.9 Success Criterion 9

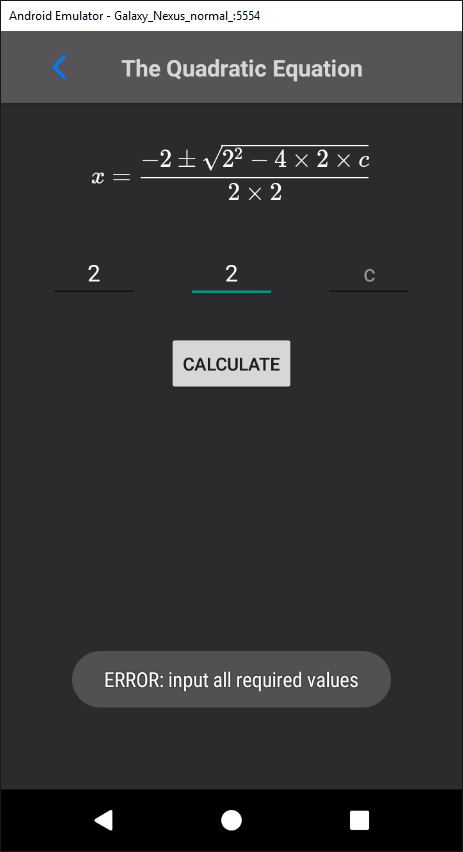
* “Detailed, helpful error messages that inform the user.”

This success criterion was implemented in the form of *Toasts*. *Toasts* are pop ups that appear on the users for a short time before automatically disappearing.

The error messages occur when the user commits an action that would result in a crash. By using a try-catch statement, the exception can be caught and the crash avoided. In the catch statement, a *Toast* is created and displayed which clearly explains the reason for the crash.

For example, if the user uses the Quadratic Equation activity and forgets to input a value for one of the variables, a *Toast* will appear. This toast says: “ERROR: input all required values”.

The proof for this is displayed below:



This screenshot shows how the success criterion has been met as when the user makes an error, a *Toast* pops up with a clear and concise message. This is helpful for the user and increases the usability of the app.

### 4.1.10 Success Criterion 10

* “The capability to change the layout of the app to accommodate users with less common device sizes.”

This success criterion was completed by creating a button in the *Calculator* activity which takes the user to a settings page when clicked. (As shown in Appendix 5.2.10).

This activity has a switch for the devices orientation which goes between portrait and landscape. When the user changes the orientation, a Boolean value is saved in the app’s *SharedPreferences*. The value refers to portrait when true and landscape when false.

In every activity, the Boolean value is retrieved from *SharedPreferences* and is used to lock the device’s orientation to either portrait or landscape.

This success criterion was extended by adding settings which accommodated more users. For example, a switch to allow the user to enable haptic feedback was included. This switch saves a Boolean value representing the value of the switch in the *SharedPreferences* when the state of the switch changes.

This value is then retrieved and if it is true, a *Vibrator*  is used to vibrate the phone for 100 milliseconds when a buton is clicked. The addition of this feature makes the app more usable. This is because it makes the process of entering inputs easier and more efficient for the user as the user always knows that they’ve pressed the button correctly.

Another feature that allows the user to select the amount of decimal points was also created. This feature was designed to increase the usability and efficiency of the solution. It does this because if the user doesn’t require a precise result, then extra decimal points only serve to clutter the interface and reduce the user’s satisfaction.

This feature was implemented by adding a *NumberPicker* where the user could select their amount of decimal points which would be saved to the *SharedPreferences.* Then, in the *Calculator* activity, this value would be used to round the user’s result.

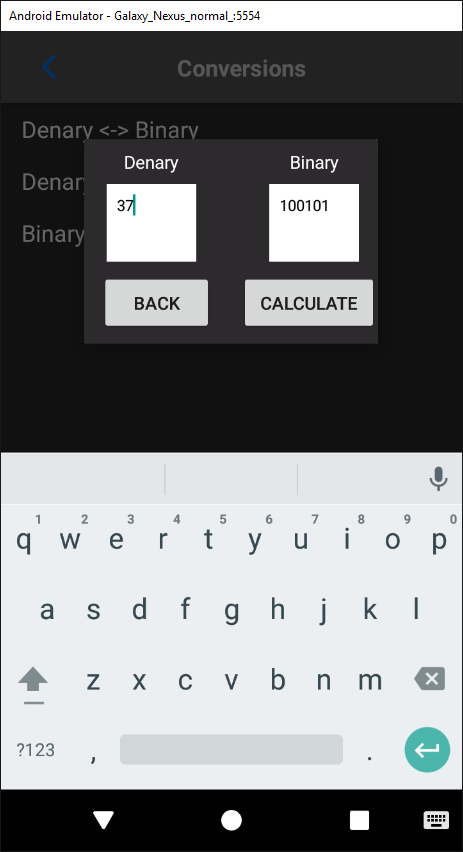
### 4.1.11 Success Criterion 11

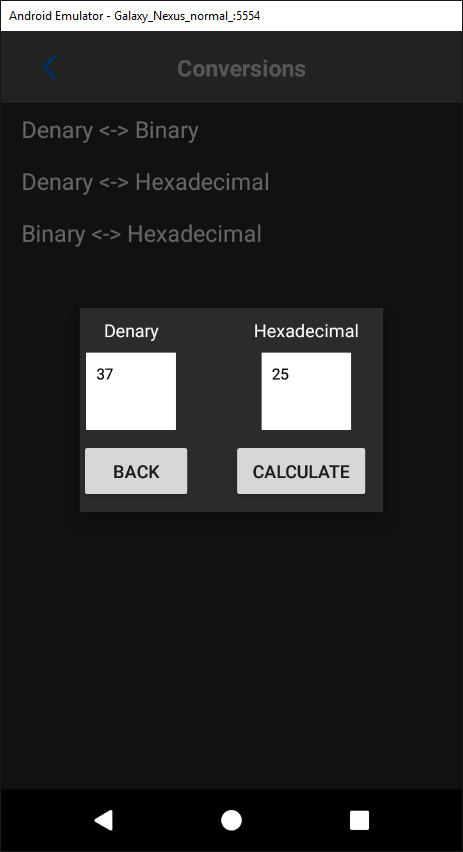
* “Conversions between different base number systems, namely, binary, hexadecimal, and denary.”

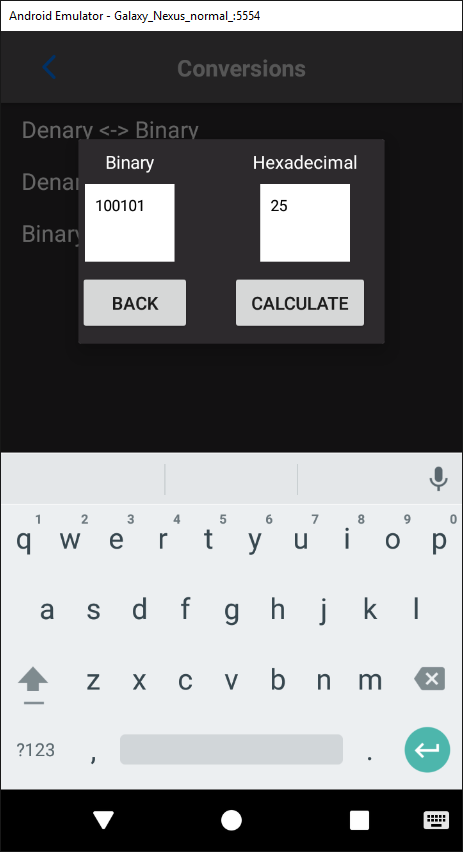
This success criterion was implemented by creating a button in the *Calculator* activity that sent the user to a *RecyclerView* containing each conversion. (As shown in Appendix 5.2.8).

When the user clicks one of these elements, it opens a *CustomDialog*. A *CustomDialog* is a user created XML pop up. (As shown in Appendix 5.2.9). The user can enter their input in the *EditTexts* included in the *CustomDialog* and the result, which is calculated using Java’s inbuilt libraries, is displayed to the user.

Proof for this success criterion is shown below:







This shows that the success criterion has been fully met as the calculator can convert between denary, binary and hexadecimal.

### 4.1.12 Success Criterion 12

* “Conversions between different units, such as miles to kilometres.”

Unfortunately, due to time constraints this criterion was not completed. I decided to remove this criterion to focus on others as it would not be frequently used. This is because students are not often required to convert between different units as the metric is almost always used.

Therefore, removing this criterion from the solution reduce the usability by the smallest amount possible while ensuring the solution remained satisfying.

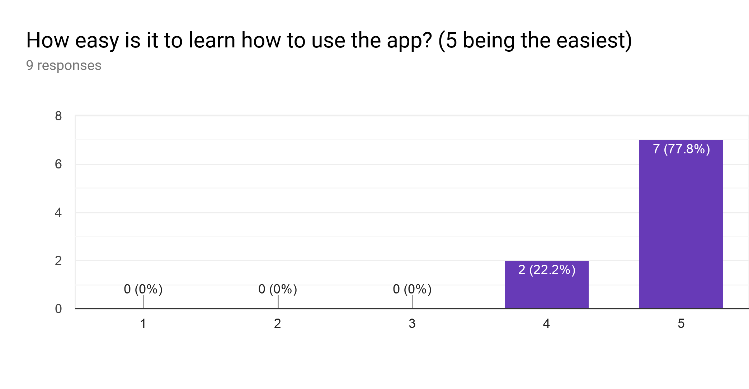
## 4.2 Justification of Usability Features

### 4.2.1 Learnability

The aim of learnability testing is to assess how easy it is for new users to pick up and do basic tasks with the application on their first use.

#### 4.2.1.1 Black Box Testing

The first interaction with the app for Black Box was generally positive. Most of the Black Box users found that the app was presented in a simple and clear interface. This is evidenced by 100% of the testers rating the learnability as positive or overwhelmingly positive.



Some of the black box users were unfamiliar with using a scientific calculator. This led to some confusion when they were faced with the extra features such as the list of equations.

This may have led to some of the responses to the survey concerning learnability to be rated at a 4 rather than a 5.

This could be improved by adding a set of instructions for each equation that included details of what input was required and where. This could be helpful as the different equations required different inputs and therefore could be confusing.

#### 4.2.1.2 White Box Testing

The White Box testing provided further information about the app’s learnability from users experienced with Java and using similar solutions.

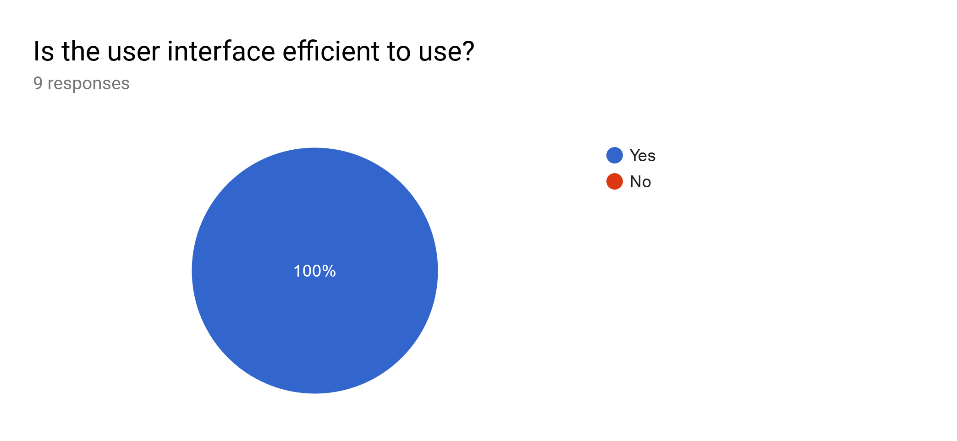
Every White Box tester found that the app was very easy to learn and became capable of using the features quickly. Therefore, I can conclude that the learnability aspect of the solution requires no more improvements.

### 4.2.2 Efficiency

The efficiency tests are used to gauge how quickly and easily the user can perform tasks and interact with the solution once they have become familiar with the software.

#### 4.2.2.1 Black Box Testing

Much like learnability, the efficiency of the solution is also rated very highly, as evidenced below:



100% of the Black Box testers responded positively to the app’s efficiency. This proves that the Black Box users had no problem with interacting with the software.

#### 4.2.2.2 White Box Testing

100% of the White Box testers also responded positively to the efficiency of the app. The White Box testers became familiar with the solution very rapidly and were able to use all the features without any problems.

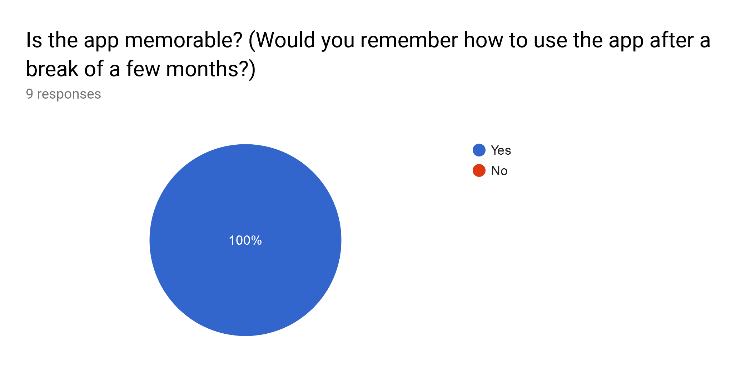
This is likely due to the design of the app being based on other calculators so users did not have to relearn the format and were already familiar with the interface. Because of this, efficiency in the White Box testers was especially high.

### 4.2.3 Memorability

Memorability tests how easy it would be for the user to pick up and use the software after an extended period of downtime.

#### 4.2.3.1 Black Box Testing

All the Black Box testers responded positively the survey concerning the memorability of the app, as evidenced below:



Memorability was rated highly due to the app’s simplistic and efficient design. The nature of the design lends itself to making the app memorable because there are no complicated features or confusing inputs that the user is required to remember.

#### 4.2.3.2 White Box Testing

The White Box testers also rated the memorability of the app highly. These testers also had access to the software during the beta development phase. Therefore, they used the app multiple times with downtime in between each use as the app was being developed.

Each time the beta White Box testers handled the app again, they remembered how to interact with it very easily. Therefore, there are no issues with the memorability of the app.

### 4.2.4 Validation and Errors

The validation and error testing is used to test how severe any errors are that the user makes during the use of the solution and how the solution handles these errors.

#### 4.2.4.1 Black Box Testing

Few of the Black Box testers encountered any errors. Those who did mostly found these errors in the conversions and equations part of the software.

The errors were largely caused by invalid input. However, the user’s input was properly validated and therefore, all the errors were caught and handled properly. Each error was handled by preventing the crash it would cause and then outputting a clear and concise error message.

#### 4.2.4.1 White Box Testing

The White Box testers encountered fewer errors. This is most likely because their insight into the development of the app helped them avoid any mistakes when inputting data that would otherwise cause a crash.

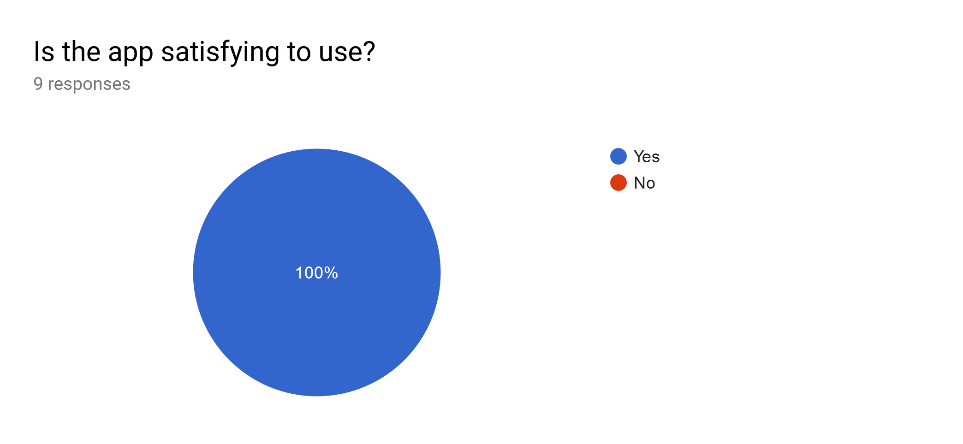
Therefore, I can conclude that there are no fatal errors that would cause the app to completely crash.

### 4.2.5 Satisfaction

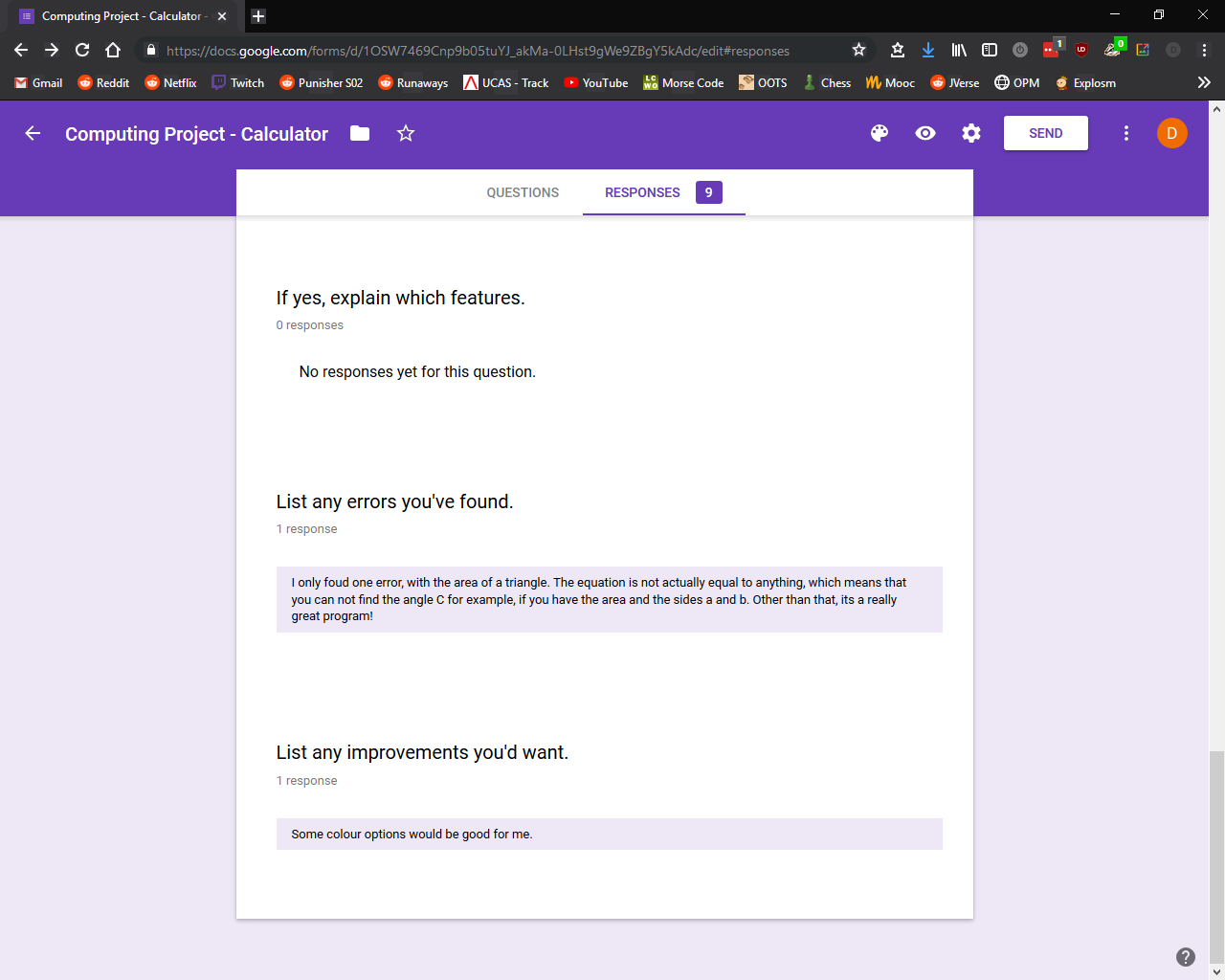
Satisfaction tests how enjoyable the user’s experience of the software is.

#### 4.2.5.1 Black Box Testing

The majority of the Black Box testers found the software satisfying to use, as evidenced below:



However, some Black Box testers found that the colour scheme of the app was undesirable:

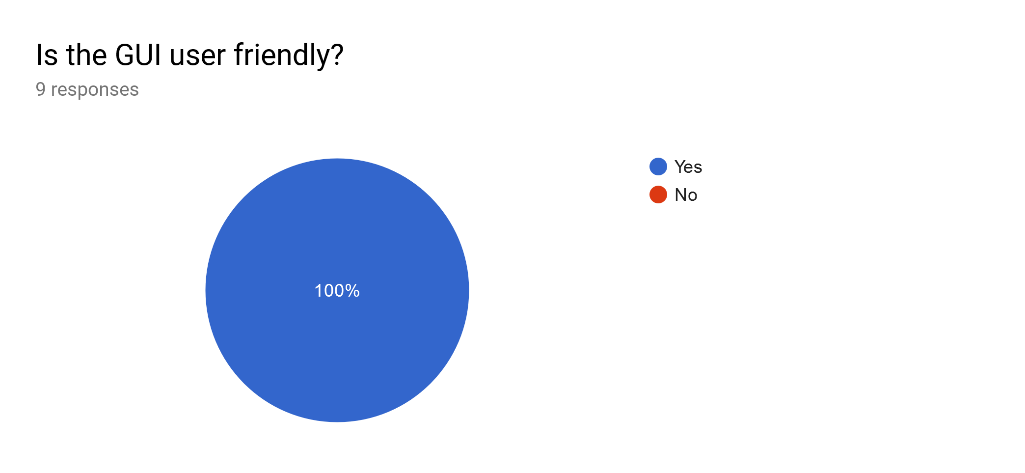


This means that some testers found that aspects of the design were unsatisfactory and detracted from the user experience.

To improve this, I would add options in the settings menu where the user could select their own colour scheme from a colour pallet. This would be done by having selectable primary and secondary colours where the primary colour would affect views such as buttons and the secondary colour would affect parts of the app like the background. When the user changed their colour setting, the colour change would be applied throughout the app to create a custom colour scheme.

#### 4.2.5.2 White Box Testing

The White Box testers also found that the app was satisfying to use, as the GUI was easy and simple to interact with. This is evidenced below:



As the diagram shows, the Black Box testers found that the satisfaction and user experience were high, with 100% of the testers responding positively.

Therefore, I can conclude that the app’s satisfaction is rated well and doesn’t require any improvements.

## 4.3 Maintenance

The software would not have any problems with maintenance. This is because the solution was built iteratively and therefore new features can be easily added into the existing code base. Due to the structure of the code base, it is very simple to create new features and operators and make them useable by linking them to the main *Calculator* activity.

Furthermore, all the libraries such as *MathView*, *RecyclerView,* and *ConstraintLayout* are dynamically linked. This means that in the case of changes to the Android libraries, the software can be updated very easily by updating the version of the library in the app’s Gradle file.

Another factor which increases the solution’s maintainability is that the software was built following the proper industry standards. For example, the code is written with the proper indentation throughout. This increases the maintainability because it makes the code much easier to read. Therefore, in the future, developers will be able to understand the code with very little effort. This will make the process of adding features or debugging simpler.

The code base is also written with comments explaining every line or code block. This can be seen in the Appendix. The use of comments improves the maintainability of the system as new developers can be introduced to the code base with much less hassle. This is because the comments explain the process and functions of the software developers can easily understand the software.

The use of concise but descriptive variable, method and class names also serves a similar purpose. For example, the variable name *mPosition* shows that it is a member variable (the scope of the variable is class wide) and holds the value of the user’s position in the expression. The use of naming schemes like this make the code much more understandable. This makes the code more maintainable as it is easier for the developer to read and therefore edit the code efficiently.

## 4.4 Limitations and Remedial Actions

Most of the limitations of this solution can be found in the success criteria that were either completely unmet or partially unmet.

For example, a one of the limitations of the software is that there is no functionality in place to handle fractions. This negatively impacts the end user as it reduces the amount of operations they can do with the app. The means the app is less helpful for the user and so provides less satisfaction. The remedial actions I would take would be to provide a button on the user interface that would enable the user to switch between decimals and fractions. I would also improve the calculator’s input display to show fractions as rather than . This would improve the user experience as it would increase the readability of the input and make the GUI more satisfying.

Another limitation would be that there are no tips displayed to the user on their first use of the app. This is especially impactful on users who are less familiar with calculators as the app is designed to mimic standard calculator layouts. For unfamiliar users, the design of the app may be confusing at first. Therefore, without tips to help the user, the learnability of the app is severely reduced. In order to improve this, I would create pop ups with descriptive hints which appear whenever the user uses a feature for the first time. This would be done using *CustomDialogs*. For example, if the user clicked on the button for the Quadratic Equation, a pop up would appear explaining what the quadratic equation is and how to use it.

The app also has the limitation of displaying the powers and roots without HTML mark up. For example, 2 to the power 3 is displayed as “2^3” rather than “23”. This detracts from the user experience as the GUI is made less satisfying. It can also make the user input expressions more difficult to read. To remedy this, I would create a method which would iterate through the input String before it was displayed. This method would replace the operators with the proper mark up which could then be displayed to the user.

A further limitation is the lack of conversions between units such as kilometres to miles and vice versa. While this is not the most important limitation as the feature would not be used as much as the others, it could still be helpful in many situations. For example, for Maths GCSE questions where the given variables are in Imperial units. This limitation could be remedied by adding further elements to the *Conversions* activity. There would be a different element for every type of conversion such as gallons to litres, metres to feet, kilograms to pounds, etc. There would also be options for different magnitudes of the same conversion. This would mean that kilometres to miles could be found in the same menu option as metres to feet. Including this feature would improve the user experience by adding more functionality so the software is applicable in more situations.

Improving upon these limitations through the remedial actions specified will greatly improve the solution and it’s capabilities.