Computer Science: Developing a Scientific Calculator

David Denny {Ma} [012071]

A level H446

Contents

0. Introduction

In 2017, there were approximately 600,000 students taking maths and science GCSE exams. In maths, 31% of these got the equivalent of a 3 (equivalent to a D) or below. There are many programs and apps that solve the problem of having a portable calculator, however, I believe that they could be improved upon to better help students.

In 2014, there were a total of 288,000 students entered for STEM A Levels. Approximately 25% of those students exited Sixth Form with a D grade or below. I think that the difficult mathematics that are included in STEM subjects are a large contributing factor to those who get lower grades. Therefore, I think developing an app that will cater specifically to students will help them improve.

My project aims to solve this problem by creating a scientific calculator Android app that is specifically designed for GCSE students, so they can have access to better tools to enable them to achieve their best possible grades.

1. Analysis

1.1 Features and Computational Amenability

The scientific calculator I aim to make will have a multitude of features that will be well suited to being created using a computational approach.

One such feature is that the calculator must be able to perform basic arithmetic operations following the rules of BODMAS. This feature is amenable to a computational approach because using the mathematical operations can be calculated much quicker and with much more reliability compared to the traditional methods of using a pen and paper which can be very prone to mistakes.

Another feature important to this project is to have the calculator in an Android app because it provides a degree of portability and ease of use that is very important for the app to be successful. This feature lends itself well to computational methods of solving because using a computer is the only way to provide an Android app to the users.

This project will also have the capability of having some common maths and science equations loaded where the students can input their variables into the calculator to solve the equation to help students as they are revising to check their answers. These inbuilt equations will also contain walkthroughs to manually solving the equation and revision tips.

This is amenable to being solved with a computer because the average phone has plenty of storage to hold all the information required to store different equations. It's suited to a computer because it is compact and should be intuitive and easy to use, unlike traditional methods such as keeping the information stored in, for example, a revision book because they are heavy and sometimes difficult to navigate. This makes my approach of using an app beneficial to the user because it streamlines their process of work and revision.

I also hope, if possible to have some basic graph functionality where the user can enter linear, quadratic and cubic equations into the app and the user will be provided with a graphical representation of that equation. This feature is important and will be solved easier with a computer because drawing one with a pencil will take much longer and is more vulnerable to errors.

1.2 Stakeholders

This project has multiple stakeholders which are listed below with their requirements:

1. **Students;** specifically, they will be students in Year 10, 11, 12, and 13 currently taking the GCSEs and A Levels. They will make up the majority of my potential user base. This stakeholder will have a high interest in this product because it will be designed specifically for them.

   Requirements:

   i. This stakeholder will have the main requirement of an Android phone with at least 2GB of storage and at least 3GB of RAM so they can store and run the program.

   ii. Another requirement for the students is the compatible Android version so the app will run on their phones. I will be developing this program to be compatible with either Marshmallow or Lollipop as these are the most popular versions with 32% and 27.7% usage share respectively as of October 2017. This will give the project the largest potential user base possible

   iii. Another important requirement is the correct aspect ratio so that the program will be presented correctly and be properly functional.

2. **Teachers;** these are another important stakeholder because the teachers must be aware and knowledgeable about the program so that they can introduce it to the students.

Requirements:

i. The main requirement for teachers to have is a phone with the correct version, either 5.X or 6.X. This will enable them to access the program as it will not be compatible with less popular versions as that would take too much time to program.

ii. Another requirement for the teachers would be for their phone to have enough RAM and storage space to be able to run. However, it is unlikely for the program to take up a lot therefore, an average phone should have more than the capacity to run the program.

iii. Also, teachers will need their phone to have the correct screen resolution, so the app is displayed properly on the device.

iv. Another important requirement is an internet connection to enable the stakeholders to download the app from where it is hosted, for example, the Google Play Store.

3. **Parents;** this is potentially the least sizable stakeholder but are still important because the parents will need to have access to the app because it will enable them to help their children with their schoolwork where they may not be able to otherwise.

Requirements:

i. A phone with a minimum of 2GB RAM and 4GB storage to enable the stakeholder to be able to use the app properly.

ii. Their phone would also need to have a compatible screen resolution that will provide the display for the user that makes the app easily readable.

iii. Another requirement is an internet connection, so parents can download the app from where it's hosted, and also download updates if they are made to ensure that their app is the most recent version possible.

iv. The parents will also need their phone to have the correct version, so the app is compatible with their phone.

1.3 Research of Existing Solutions

During my research of this problem, I have found many different solutions that solve the problem, or at least partially solve it. The various solutions achieve different levels of success and therefore, I will be analysing different apps.

1.3.1 HiPER Scientific Calculator

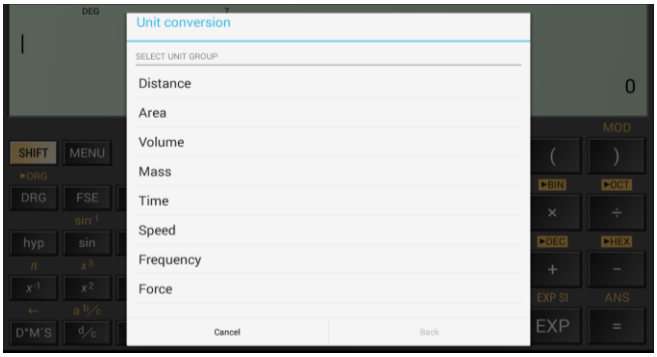
This is the first example of a very successful solution that I came upon. The HiPER Calculator is presented in the traditional scientific calculator form and is very easy to interpret.

The basic layout of the calculator

This app has many positive points about it. For example, it's usage of the traditional layout means that the users will instantly be able to interact easily with the app.

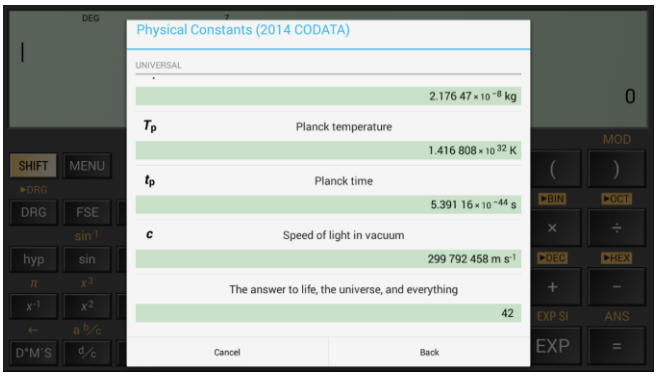
The main positive point about this app is that it performs very well and is responsive to user inputs which means that it's quick and easy to be used. This app covers all the basic features and functions such as conforming to BODMASS, basic trigonometry, surds, powers, etc. However, the calculator also contains inordinate amounts of other, more niche features that make this calculator much better than the rest on the market.

An example of this is the large amount of hidden functionality that it contains behind different menus. For example, it has the ability to convert between almost any units:



The different possible conversions

Other features like this include the storage of common constants that are used in scientific fields.



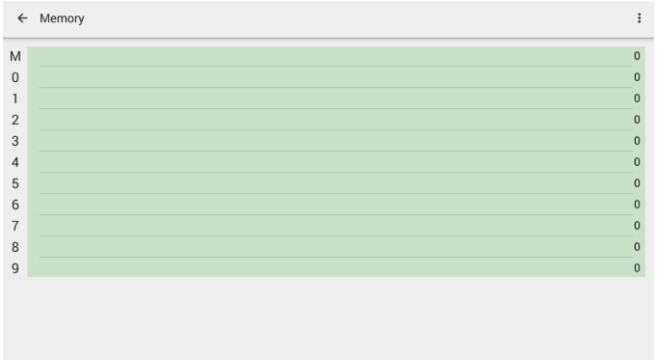
Different possible constants – there are many more

This feature is incredibly useful for students as it reduces the amount of time it takes to complete calculations which can be especially useful in certain situations, for example tests. I will be almost certainly adopting this feature in my own project because I believe that this improves the usability of the calculator a lot.

However, I will make my own changes to this because as my own calculator will be specialised towards students in the years. Because of this, I will remove some of the constants as they will not be used up to GCSE and therefore will be useless.

Also, I will be altering the constants to fit in line with the exam's own values. For example, the speed of light in vacuum is 299,792,458ms^{-1}, but in exams the value is taken to be 3.0x10^{8}. Therefore, it is counterproductive to GCSE students because they will receive a slightly incorrect value which could result in rounding errors in their calculation, leading to a loss of marks.

However, this app does also come with a relevant flaw in that the user cannot store numbers as variables for ease of use. The memory storage in this app is reduced to M, and 0-9:



The calculator's storage capacity

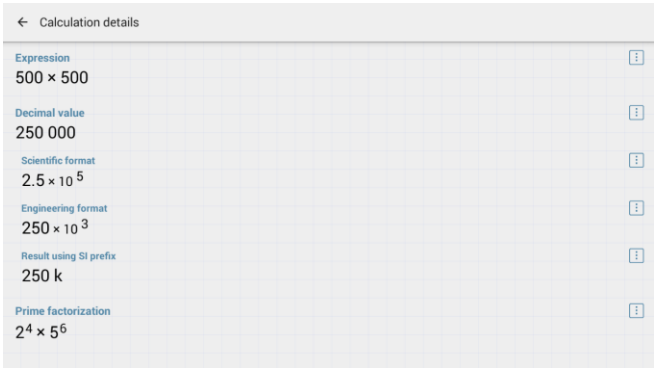
I believe this is a major flaw in the calculator because the ability for the user to store their own preferred values in variables is very useful to make the use of the calculator more efficient. Also, the added customisability it provides means using the app becomes more intuitive.

The HiPER Calculator also has the functionality to convert numbers between different representations, such as binary, hex, and denary. I think this feature can be very useful, especially to particular students, such as ones taking Computer Science. This feature is further extended past just simple conversions by having modes for different base number systems.

The calculator in the binary mode

As the screenshot shows, 255 has been converted to 1111111, and the app has entered the 'BIN' mode where the features are different. Prominent examples of this are all the keys on the number pad except '1' and '0' are disabled. The letters from A – F are also enabled when 'HEX' mode is enabled. Further features include the inclusion of logical operators such as 'AND', 'OR', etc. which would be very useful for the user when performing binary calculations.

Other helpful features include things like the ability to copy the output to the clipboard by tapping on the number. The app also has the functionality to display in depth details about the calculations the user is performing.



The output of the 'More' button

As shown in the above screenshot, by clicking on 'more', the user is presented with further details about the calculation which involves things like displaying the result in standard form or with different prefixes.

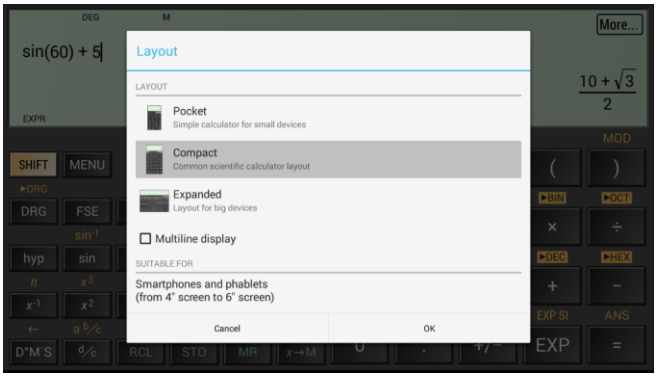
These features and other ones like them are things I would like to include in my final project, however, they are considerably less important than other aspects of the app because while they are helpful, I feel that they are not as necessary to the calculators overall functionality. There are also some features that are useful, but not necessary in my own project, such as the functionality for complex numbers because they are not introduced until A Level and therefore would be pointless in my project.

This calculator also has many benefits other than it's many features, for example, the app itself is presented in a very clean and uniform way that makes it enjoyable to use as opposed to others on the market which can tend to be gaudy and unprofessional. A good example of this is the app's menus:

The calculator displaying different formats for the result.

In my opinion, the presentation of the app and its different menus is very sleek and minimalist which makes using the app easy and intuitive.

There are also other benefits to using this app as opposed to others relating to it's presentation. For example, there are choices in the menu for different layouts. This is an important feature because there is a risk for users with uncommon device sizes to have the app be presented with a distorted view that can block off some buttons and features that limit the apps usability.

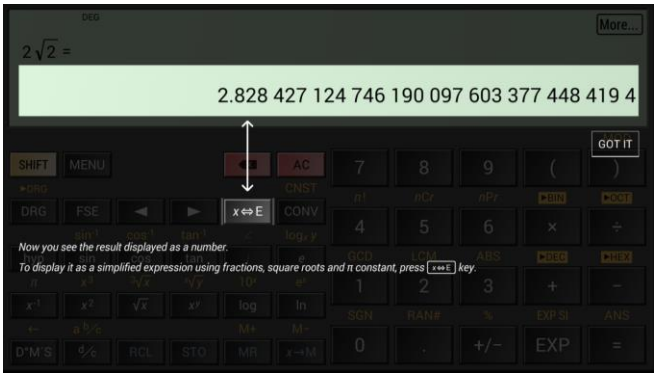


The different layout options

The layout mode for tablets

I will attempt to include this feature in my own project because it allows a larger percentage of users to access the program and therefore become more useful to a larger number of people.

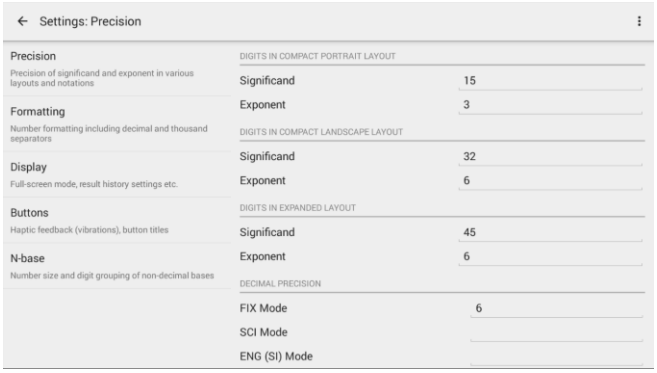
Another helpful feature that this app includes are the tips that come up when the user accesses certain, more nice or complicated features.



A tip introducing the user to converting between different displays

This feature makes the calculator more accessible to users with different levels of knowledge and experience with using scientific calculators. This will be especially important in my own project because increasing accessibility to a higher proportion of users means that more students will be able to benefit from the app. Therefore, this will certainly be a feature that I will be implementing into my own project. I will focus the use of these tips on underutilised features such as the memory storage because it's important not to bombard the user with notifications as it can annoy the user and so make the app undesirable.

This app also has a very large range of settings for different parts of the program, such as formatting and the display.



The different settings for 'precision'.

The range of settings for the calculator provide a good benefit for using the HiPER Calculator over other apps because it offers a large amount of customisability, making the app easier for the user to interact with and understand.

This app has other features designed to heighten its level of understandability such as the larger range of error messages for different situations.

The error that appears when an expression is unfinished

This feature is designed to help the user and make the functionality of the app easier to use and understand because the error messages are more descriptive than tradition calculators which would simply output 'SYNTAX ERROR'. The traditional calculators are less detailed therefore require the user to take longer to fix their expression, whereas the HiPER Calculator has a detailed error message and even highlights the area where the error occurred.

This, and features similar to it such as the calculator auto-completing the expressions in some situations such as a missing bracket or unused decimal point improve the calculators usability by streamlining the process for the user.

1.3.2 ColorFul Calculator

This app is an example of a less successful, while still functional calculator. The presentation of this calculator is lacking, especially in terms of a professional setting.

The base view of the calculator

While this app does cover all of the very basic functionality, such as the four basic operations, capability for percentages, and memory storage. However, the app does not extend much beyond that. For example, although the app does contain the ability to store values in the memory, there is only space for a single number.

The value '42' is stored in the memory

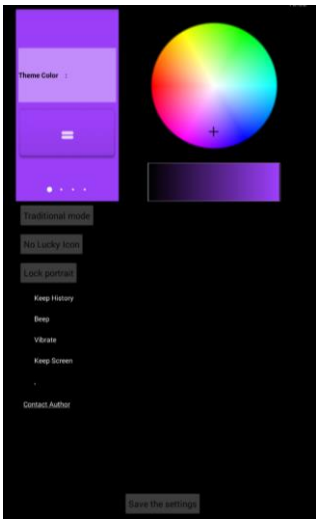
Due to this lack of any major functionality, the app is much less successful when it comes to the user performing calculations that are more involved than operations on a couple of numbers. Another disadvantage to this app is that when inputting the expression, the calculator does not display the user's input. This means, if the expression is longer than a few separate numbers, it can be awkward to remember what the user has entered to the calculator, meaning it is easy to make mistakes.

Another disadvantage due to this flaw is that if the user gets an incorrect output, they cannot de-bug their own expression to find where the error occurred because it is not displayed to them.

Furthermore, the app contains advertisements which can be very intrusive based on their location. As shown in the screenshots above, the adverts are placed below the calculator, just below the 'equals' sign. This placement could lead to the user accidentally clicking on the advert instead of the equals sign if using the app quickly. This could be very frustrating for the user, leading to less usage of the calculator.

In light of this, I will not include adverts in my own project because I think they can be very detrimental to the user's experience. Also, if it was made necessary to have adverts, due to a need for funding, for example, I would make sure that they were in the most unobtrusive location possible because the user's experience should be the utmost important.

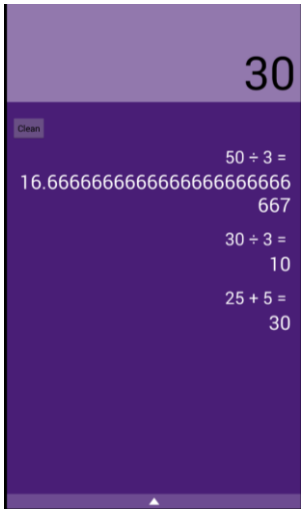
However, this app does have some redeeming features, for example, it has some semblance of customisability.



The range of different colours that can be selected

The app can be customised using a colour wheel that changes the general theme of the rest of the app. Although this feature does add customisability to the app which is a positive, the potential customisation is largely unprofessional and gaudy making it ill-suited for an app that will be used in a work environment. Therefore, in my opinion, this feature would be largely pointless in my project.

This app does contain other advantages, such as the feature that includes a history of past calculations.



The past calculations are shown

This feature can be very useful because it allows the user to access and help remember what their past calculations are, and so reduces the chance of the user losing track of what they're doing.

1.4 Requirements for the Solution

Requirements for my solution can be split into two sections, the requirements for myself as the developer and for the user.

1.4.1 Developer's Requirements

The resources required for the solution to be developed will include things like:

- A computer with the capability to run Java. This is because I will be programming the project in Java due to the language's versatility and because I am very familiar with it. This computer will need to have no limitations or restrictions on the internet usage and will require full admin access.
- An Android virtual machine. This will be used to test my program without having to use a physical phone which will make the process quicker and more efficient.
- An Android phone with the required minimum specifications. This will be used for testing of the project, so I can find out how the program runs on actual phones and use it to bug fix my program.
- Testers. I will require GCSE students to test my program on because they are the primary stakeholder and, so it will be important to test my program on them to take their criticisms into account to make the program as useful and successful as possible.

- Access to wherever the app is hosted, for example the Google Play Store so the app can be accessed easily by the users.

1.4.2 User's Requirements

The user's requirements will differ greatly from the developer's. Their requirements will contain:

- An Android phone with the minimum specifications requirements, including the correct version of Android. This will mean that they will be capable of running the program without crashing or otherwise harming their phone. Also, their phone will need to have the correct resolution so that the program can be displayed clearly, and the UI is properly presented and is easily used and intuitive.
- A stable internet connection. This will be required so that the user can download the app from wherever it is hosted and also to potentially download updates.

1.5 Limitations of the Proposed Solution

My proposed solution will, unfortunately, have some limitations. For example, my project will likely have the limitation of not having functionality for graphs. While having the ability of drawing graphs in the app would be very useful, I think it will be very difficult to be able to include this feature.

This is because implementing the feature will have more difficulties to overcome than other, more simple features. Also, graph functionality is less of a priority compared to other features that are simply more important.

This can be seen in that there are dedicated apps to just graphs, showing that it is very complex to make well. Because of this, I think that if I try to implement graph functionality, other parts of my project will suffer.

Another limitation that is likely to occur is that the app itself may end up with an ugly user interface. This is because I am lacking when it comes to making artful and attractive design. Therefore, to overcome this potential limitation, I will certainly make often checks and tests with other people such as potential users. This will ensure that the final product will have an attractive design that is easy to use and intuitive.

1.6 Success Criteria

The success criteria for my project can be broken down into basic features that are critical to the app's success and more advanced ones that, while still important, the app can function without.

1.6.1 Basic Success Criteria

- Functionality for the four basic operators, +, -, x, ÷. This will be of the utmost importance because without this, the calculator will not work at even the most basic level.
- The calculator will follow the rules of BODMASS. This feature is very important because it makes the experience more streamlined for the user because they don't have to concern themselves about the formatting. Also, it will allow users to be more familiar with the app as they will be used to calculators following BODMASS.
- Have functionality for using square roots and powers. This feature will increase the amount of calculations the user is able to perform.
- Be able to use sin, cos, and tan and the inverses of. This is feature is required because it will enable the user to do trigonometry related functions which are a very large part of the maths GCSE.
- Cursor controls that will move the cursor around the equation. This is required because it allows the user more freedom and enables the user to edit equations.
- Capability for common forms of notation, such as standard form, fractions, decimals, etc. This is essential to the app because it will allow the calculator to be more understandable for the user.
- 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. I can also use this feedback to improve upon the design in later development iterations.

1.6.2 Advanced Success Criteria and Essential Features

- Displaying tips when the user access more niche and unknown features. This feature will be very beneficial because it will help users who are less computer literate and therefore increases the accessibility to a higher proportion of the potential user base.
- Store equations for subjects such as Maths, Physics, Chemistry, etc. that the user can select and enter their variables to complete the equations. These equations will be stored in a list that the user can access by tapping a button on the main activity.
- Detailed, helpful error messages that inform the user. This is important because it enables the user

to use the app in a more productive way and therefore improves the apps usefulness.

- The capability to change the layout of the app to accommodate users with less common device sizes. This feature is important because it will allow a larger percentage of users to be able to interact properly with the app.
- Conversions between different base number systems, namely, binary, hexadecimal, and denary. This will be a useful feature as conversions between number systems feature heavily in the Computer Science GCSE and therefore will be helpful to GCSE students.
- Conversions between different units, such as miles to kilometres. This feature will be very beneficial to the app because it will streamline the user's process when doing calculations because less time will be needed to be spent doing conversions.

2. Design

2.1 Method Table

Method	Parameters	Variables	Class	Description
BODMAS	The mathematical expression that the user enters that will be calculated. This is called the infix.	• infix • postfix	ShuntingYard	This method is the basis of the entire app. This will take a string that the user provides, called the "infix". I will use the Shunting Yard algorithm to apply BODMAS. This algorithm is stack based and works by taking the infix (which is the user's mathematical expression) and converts it into the postfix. The postfix is created by manipulating the infix with stacks. The postfix is also known as Reverse Polish Notation or RPN. The postfix can then be evaluated to compute an answer to the user's original expression. This will occur in a different method. For example, if the infix is "3 + 4 × 2 ÷ (1 − 5) ^ 2 ^ 3" the postfix will become "3 4 2 × 1 5 − 2 3 ^ ^ ÷ +".

EvaluateRPN	The postfix that has been created by the BODMAS function.	• postfix • result	ShuntingYard	This method will evaluate the postfix and return the answer to the user's original expression. This is done by looping through the string of the expression and when the specific element is an operator, pop two numbers off the stack and manipulate them according to the type of operator that is given. For example, if a "+" operator is the current element in the loop, two numbers will be popped off the stack and added together. This works similarly for other basic operations such as powers, division, subtraction, etc. Because the string is in Reverse Polish Notation, the order of the numbers and operators means that when the numbers are popped off the stack, they will be calculated according to BODMAS.
Roots	A float as the base and an integer as the root.	• base • root • result	MainActivity	The user can enter their root number and their base number. The base number will be rooted by the root using the Java Math class. The result will then be stored in the result variable. For example, the user entering "2" and "4" will perform a square root of 4.
Trigonometry	Which function is to be used (sin/cos/tan).	• trigFunction • input • result	MainActivity	The method will evaluate the trigFunction variable to decide the

	The number that it should be performed on.			trigonometry function that will be used. Depending on the trigonometry function, the user's input will be calculated using Java's Math class. The answer from this will be stored in the result variable.
Inverse Trigonometry (sin^{-1}, cos^{-1}, tan^{-1})	Which trigonometric function is to be used (sin^{-1}/ cos^{-1}/tan^{-1}). The number that it should be performed on.	• trigFunction • input • result	MainActivity	The method will decide which inverse trigonometric function to use on the user's input depending on the parameters given to the method. When the calculation is done, the result will be stored in the result variable. This method will require validation so the user's input are within the appropriate bounds. For example, doing the calculation $sin^{-1}(x)$ where $-1 \leq x \leq 1$ will result in an error. Therefore, the method will require validation to prevent the user performing these actions and crashing the app.
convertNotation	Which notation form the user wants to convert their answer to. The notation that the user's number is currently in. The number that the conversion will be performed on.	• currentNotation • requiredNotation • input • result	MainActivity	This method will take the user's input and convert it to all the other notation forms. This will be displayed to the user. For example, if the user has a decimal number, it will be shown in fractions, standard form, and surds. The user then has the option to select a specific type of notation for their input to displayed in.

convertUnits	A string containing the current unit and a double containing the value of that unit.	• currentUnit • input • resultNum • resultUnit	MainActivity	This method will be called on the user's click of a button. It will bring up a menu where the user can select the conversion they want. For example, they can select "cm -> m". The method will either take the number currently stored in the calculator's answer variable or the user can enter their own. In this example, where the user has just entered a calculation with the result of "120.0", the currentUnit would be "cm". The input is "120.0", the resultUnit is "m". The calculator will then perform the conversion of centimetres to metres and store the result in the variable "resultNum". In this example, the resultNum would be "1.2". The method will have the functionality to convert: • distance • time • area • mass • volume • speed
Display Error	The error code	• errorCode • errorDetail	MainActivity	This method will be called for different types of errors. For example, it will be called when the calculator encounters a failed verification check. When this occurs, the error code

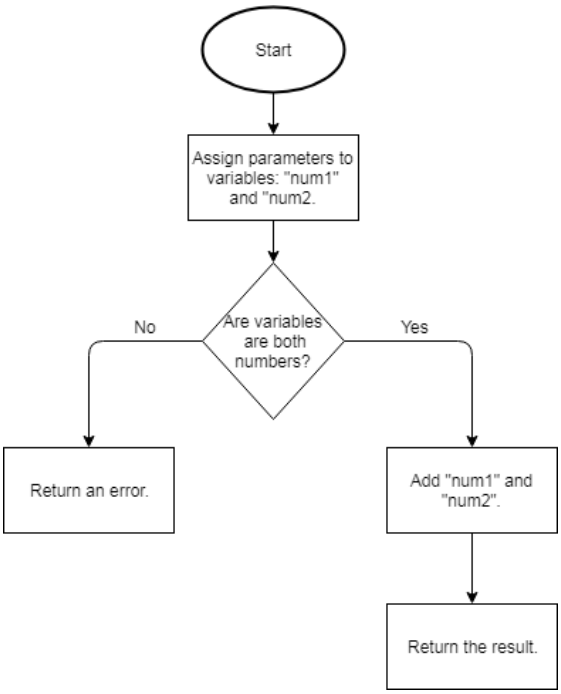
				will be a parameter in the method call. This error code will be linked to a description of the error and how to fix it. This description will be a string stored in the variable “errorDetail” and will be displayed to the user. Another example of when this method might be called is when a method catches an exception. Each type of exception will have its own unique error code with a corresponding description of the error that will be displayed to the user.
showEquations		• selectedEquation	Equations	This method will display a list of the stored mathematical equations in the calculator. When the user selects one, this method will call “useEquation(x)” where x is the user’s equation.
useEquation	The equation that the user has selected	• Variables that are used in the equation, such as frequency, distance, speed, time, etc. These will be provided by the user. • result	Equations	Upon being called, this method will query the user for to provide the known variables and select the unknown variable. For example, with the equation to calculate speed/time/distance, the user would provide two variables – “30 metres” and “5 seconds”. Then the user would select solve for speed and the calculator would output “6 metres per second”. All equations require the values to be entered in SI units.

				This is because the equations in GCSE and A Level papers are to be used with SI units. Therefore, this will help the user become used to doing calculations and become better prepared for their exams.
changeLayout	The new layout that the user has selected.	• currentLayout • tablet • phone • portrait • landscape	MainActivity	This method will change the xml layout of the GUI. This will accommodate different screen sizes, so all users can use the app easily without elements being cut off or inaccessible. This method will also enable the users to switch between portrait and landscape layouts. The different orientations will require separate xml files because each one will need to be designed for the all the features to be intuitively accessible.
showTip	A variable that identifies which function the tip should be shown for.	• tipDetail	MainActivity	When a method is accessed for the first time, it will call this method. This method will display a pop up to the user that explains the method that the user is accessing does and anything else that will help the user to use the app. The content of this will be stored in the variable "tipDetail". For example, when the user first accesses the convertNotation() method, a pop up will be shown explaining that the method shows different notations for the user's results. It will

				also outline how the user can select a notation to be used in the calculator.
moveCursor	The directional input (this will be decided depending if the user clicks the left or right arrow).	• direction	MainActivity	Moves the cursor in the GUI different direction depending on the user's input.

2.2 Algorithms

2.2.1 Basic Operations, e.g. Addition

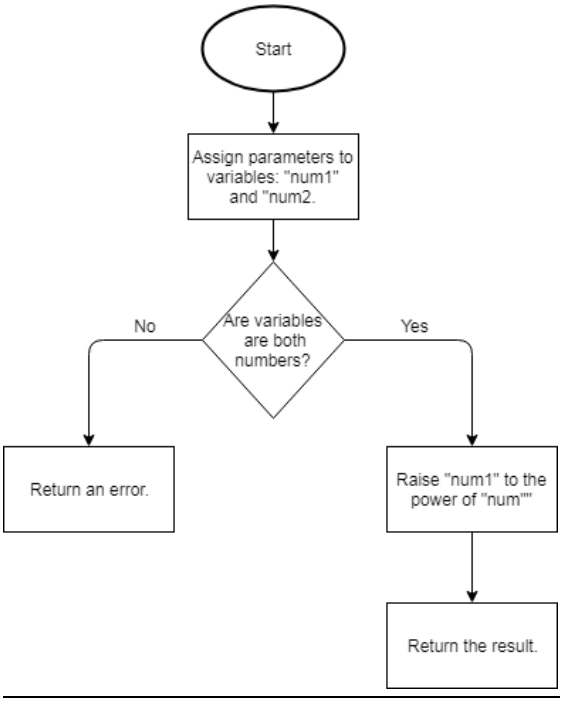


This flowchart shows that two variables are created and added together, then the method is returned.

A similar algorithm will be used for the other basic mathematical operations such as subtraction, division, and multiplication. This is done by changing the operation that is done between the two numbers.

This method is required because basic mathematical operations are the minimum requirement for a calculator to be of any use. Even if the calculator had more advanced features such as converting numbers to different number bases, the calculator would still require basic functionality for it to be helpful to any user.

2.2.2 Powers

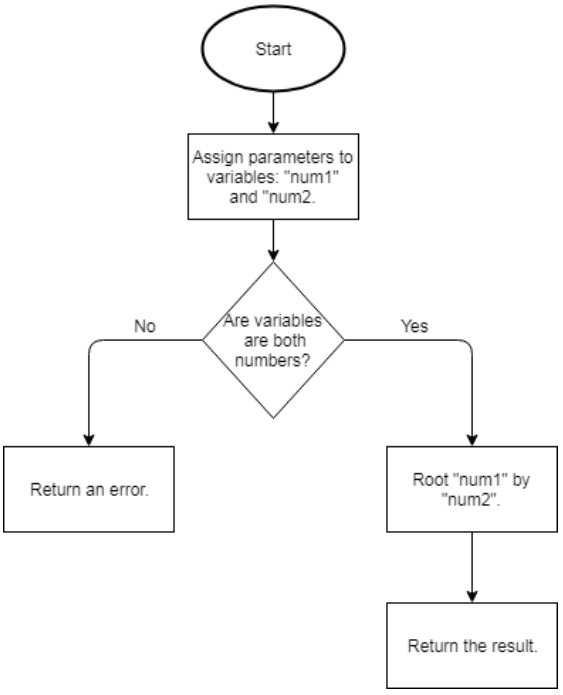


This flowchart shows how the initial parameters that are passed to the method are assigned to the variables, “base” and “power”. After validation checks are performed on the two input numbers, the calculator will raise the base to the order of the power.

This method is required for the calculator app because the ability for the user to user more advanced maths (compared to basic functionality) is very important to for students in their GCSEs or A Levels.

Powers are used in almost every aspect of a STEM subject in GCSE and therefore this functionality is required for the calculator to be of use in more situations.

2.2.3 Roots

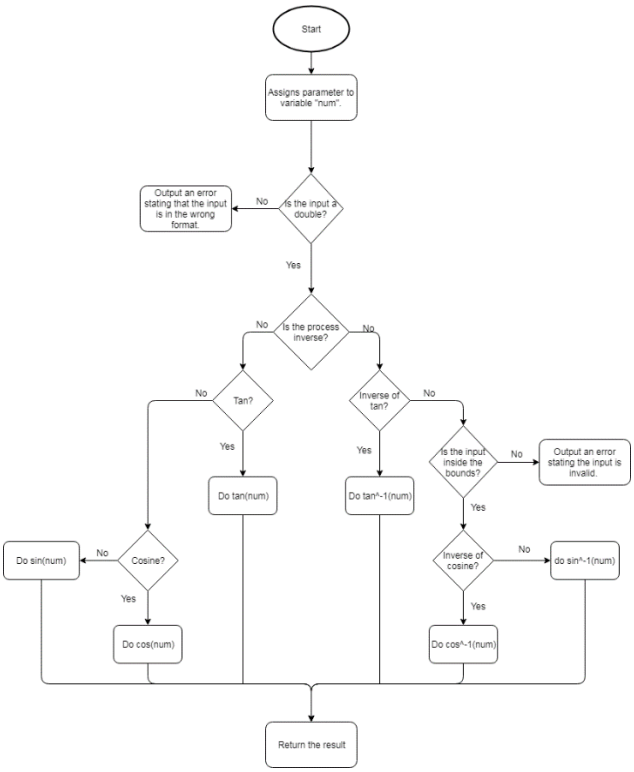


This method takes the parameter and assigns them to the variables ‘base’ and ‘root’. The two variables under go a validation check to ensure that using them won’t result in an error that could lead to a crash.

After the validation is confirmed, the the “base “variable will be rooted by the “root” variable.

This method is required so the calculator can be used and be more helpful in more situations. This is because the functionality to root numbers is often needed during school lessons and for schoolwork. Therefore, without this feature, the calculator would be useless in many situations.

2.2.4 Trigonometry



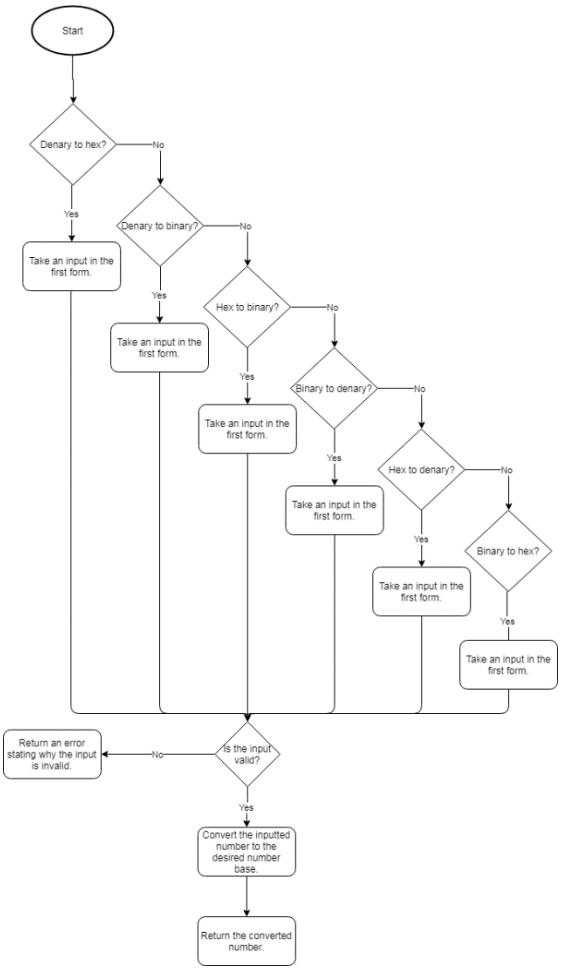
This flowchart shows how the function for calculating trigonometric expressions will be carried out.

This includes two forms of validation. The intial validation ensures that the input is of the correct data type so the function does not attempt to perform a calculation on a string.

The second form of validation is for the sine and cosine calculations only. This validation ensures that the input is between 0 and 1 for inverse calculations of sine and cosine. The validation is only required for the inverse versions of sine and cosine as they must be within the maximum and minimum bounds of their waves whereas the tan wave increases to infinity.

This method is required because trigonometric functions are often used in GCSE and A Level subjects. For example, trigonometry is a large of the Maths GCSE specification and every student will need to learn how to use it. Having this feature in the calculator will help the user understand trigonometry and so help them in their exams.

2.2.5 Number Base Conversions



This flowchart shows how the process of the user selecting a type of conversion, then entering a number which then gets converted.

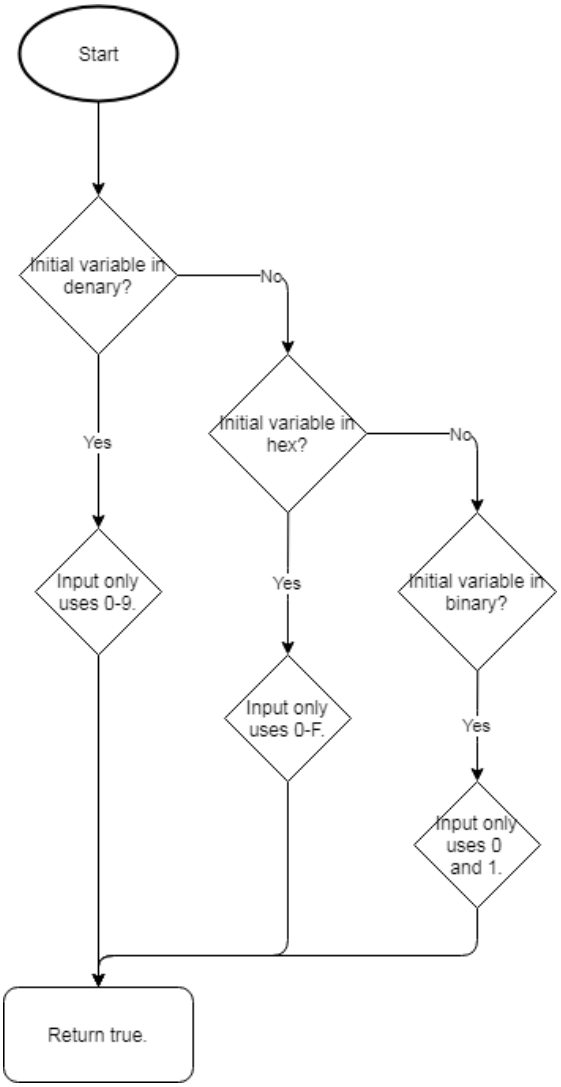
This algorithm enables the user to perform conversions to different bases such as binary to hex.

There is one instance of validation in this algorithm. After the user inputs their number, it will be run through the validity check. This checks that the input is valid for the different types of conversion, for example, binary to denary will only accept the numbers ‘1’ and ‘0’.

This algorithm is especially required for Computer Science GCSE and A Level students. This is because the Computer Science specification requires students to convert denary to hex, binary to denary etc. The specification also includes basic binary operations.

Therefore this feature will be very helpful to those students as it will allow them to check their working so they can see their mistakes and improve upon them.

2.2.6 Number Base Conversions Validity Check

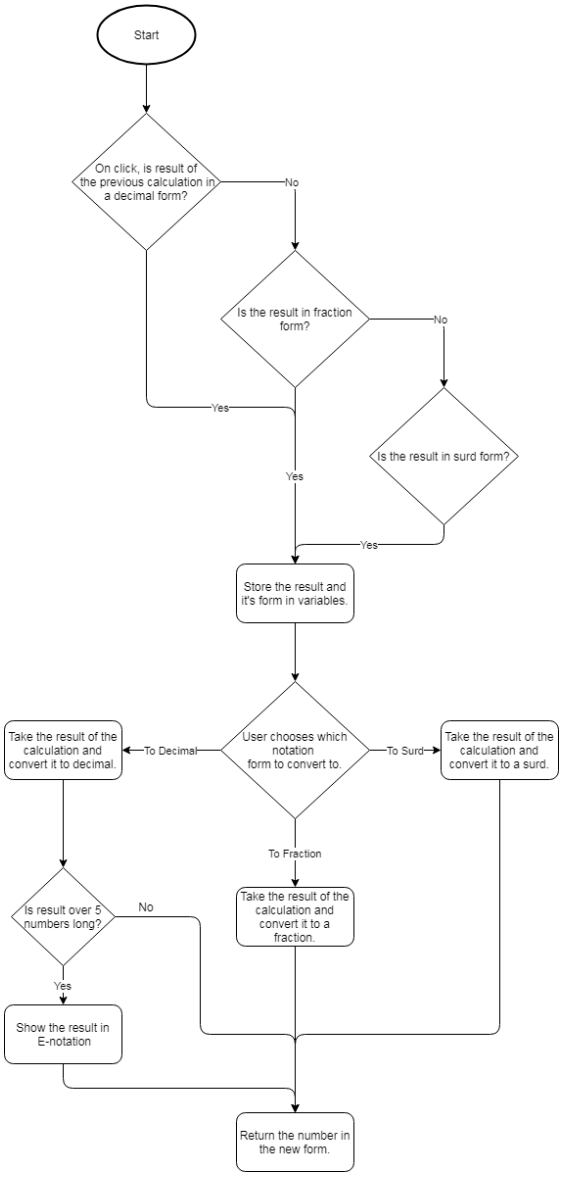


This flowchart shows how the input for the base number conversion will be validated. The algorithm will employ different validation methods depending on the resulting number base that the user wants.

For example, if user selects binary to denary conversion, then the input number must only contain the digits “1” and “0”. This is because binary is a base 2 number system.

This function works in conjunction with the number base conversions. Therefore, this algorithm is required so that the number base conversions can be used without any errors occurring. This potentially prevents errors that could crash the app or otherwise influence the user’s experience.

2.2.7 Notation Conversion



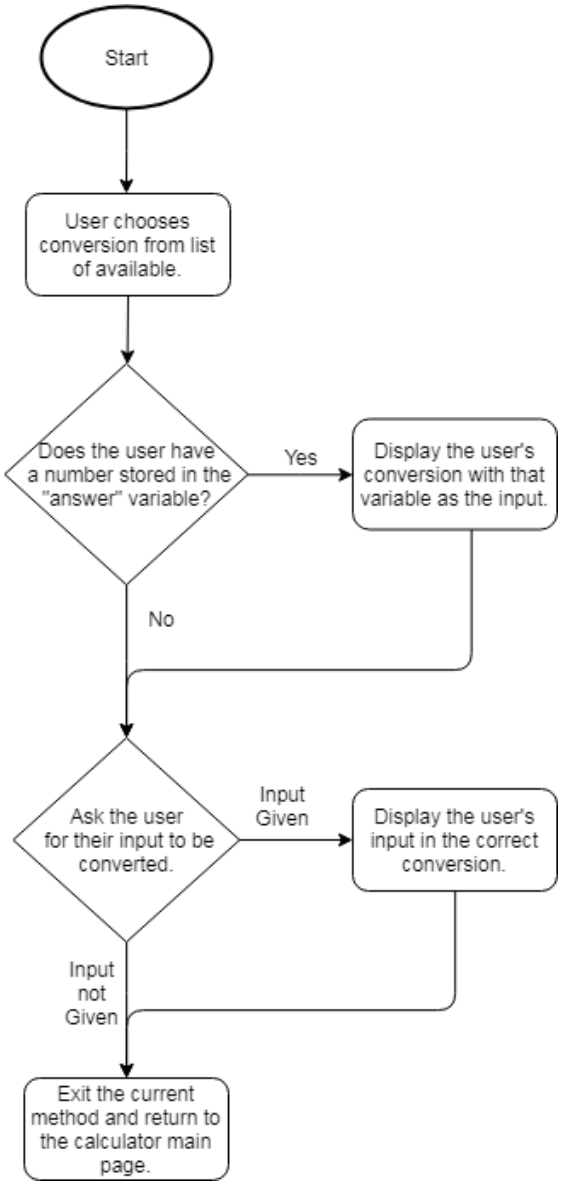
This flowchart shows the process of converting values between different types of notation.

For example, if the user chooses to convert a fraction to a decimal, the first decision (is the input decimal) will be a no so the flowchart carries on to the next one. This one is positive because the decision is whether the input is fractional. The input and which notational form it's in is stored in variables. Then the user chooses what notation to convert it to.

In this example, the user chooses decimal notation. The input is converted to decimal and another decision occurs, if the decimal result is over 9 digits long, it is converted and returned in scientific notation (used when the decimal number is too big to be conveniently written).

This method is required because throughout school, the students are required to be able to convert fractions to decimals and vice versa and to be able to use surds, etc. Therefore, having a method which will display the other forms of their calculations will help the students become familiar with all the different forms of notation.

2.2.8 Unit Conversion



This flowchart shows how the method for converting units will be carried out.

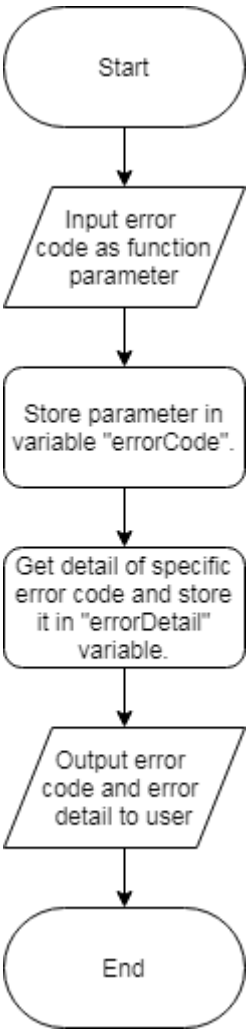
For example, the user will choose their conversion such as millilitres to litres. The method will check if there is currently a number stored in the answer variable. If so, the method will display the conversion using the answer variable as the input number.

After displaying it (or straight away if the variable is empty), the method will then ask the user for input.

If this is denied, the method will exit and the user will be returned to the main calculator screen.

This method is required because in STEM subjects, especially ones like Physics, equations are often used with different units. The specification requires the student to understand how to convert units and so this calculator will help them understand the process and let them focus on harder parts of the problem.

2.2.9 Display Error



This flowchart shows the process of an error occurring and it being represented to the user.

For example, this algorithm will be a method that is called when an error occurs. The method call will have a parameter which will be an integer error code.

Each unique error code will have a corresponding string that can be displayed to the user. This will explain the reason of the error and how the user can fix it.

This algorithm is required because having a clear and understandable error messages will be very important when it comes to the app's usability and the user's experience.

For example, the user makes a mistake when they write out their expression and it doesn't work. If the app doesn't have any intuitive error messages the user will become frustrated and annoyed. This ruins the user's experience and makes it less likely that the user will continue to use my app.

Therefore, having clear, concise and helpful error messages is an important requirement of a successful solution because it greatly improves the UX and makes it more likely that the app will be used.

2.3 Usability Features

2.3.1 Learnability

To help the users learn how to interact with the app quicker, hints will be shown when the user clicks on different buttons. For example, when the user enters an equation that outputs a surd, the app will notify the user of the feature that converts values from surds to fractions or decimals. It also shows the user how to use the feature.

Another way the app will be made easy to learn is through the design of the user interface. The UI will take design features from other well-known calculator apps and real life calculators. This will increase the learnability because the user will find the design intuitive to use.

For example, it will use a "shift" button. This is a commonly used feature on calculators that enable buttons to double up on uses. The buttons can have a default function, and a shift function. Using this makes the app easy to learn because the "shift" button is often used in calculators and so it will be intuitive and not require much time for the user to learn.

2.3.2 Efficiency

To help the users interact with the app as efficiently as possible, the majority of the actions the user can take are located in the first activity and layout of the app. Therefore for a large percent of the time that the app is being used, all the usage will remain in the first activity and therefore it will be efficient. It will be efficient because the user will not have to trawl through menus to perform their desired action.

Another way the efficiency is improved is by multiple layout options for different device dimensions. For example, a tablet layout will take advantage of the larger amount of space to include more buttons and functionality into the main layout. This means the space on the screen will be used very efficiently to enable the user to perform functions as quick and efficiently as possible

2.3.3 Memorability

Using the app will be very memorable because it will closely mirror the design of real life scientific calculators. This means that most features will be labelled and have the same annotations as a real calculator. This will help with memorability because the userbase will likely already have experience with scientific calculators. Therefore they will be used to interacting with a similar design and so they will remember how to use it.

2.3.4 Errors

My app's layout and functionality will be designed to maximise the ease of use for the user. The most common error would likely be the ones caused by the user entering invalid data that could cause crashes. To prevent these types of errors, I will include a lot of validation for every method that the calculator uses. For example, when the trigonometry method is used, the input will be checked that it is between -1 and 1 for cosine and sine.

If the input is invalid, using it in the method would cause an error that could cause the application to crash or otherwise stop working. Instead, the validation rules can catch it and instead inform the user of the invalid input and prevent it from being used.

Other common errors that the application will encounter will be calculations have a result that is outside the possible range for the application to process. This will occur when the user's expression results in an output that is too large to either display on the screen or the processor cannot handle the process.

The error of having a result that is too long can be solved by detecting when the error occurs by testing the length of the result and then changing the notation of the result if it's too long. For example, converting the result from decimal to scientific notation will conserve space and enable the user to read the result with more ease.

The error of the user entering an expression that uses all of the CPU's processing power which will cause the device to freeze or crash is more likely to occur on older devices, such as the one's that run on Android 4.0 and older. This error will be much more difficult to prevent because different devices can handle different levels of processing power, therefore the application cannot be programmed to reject specific equations that are known to crash a device.

2.3.5 Satisfaction

My app will be as satisfying as possible for the user to interact with. I am aiming to make it satisfying by creating the design based around mainstream calculator designs that are already widely used.

For example, I can base my calculator's layout off the common calculator that students and schools use. This will make my app more satisfying to use because the user will have experience with other calculators that can be transferred over to my own.

Because of this design's easy accessibility, the potential user base will be as wide as possible. Therefore the solution can be used by any student, teacher, or parent no matter their age or level of computer literacy. This means each user will be satisfied with the solution because the design will be specifically created so it is easily accessible to sany user.

I will also carry out a survey targeted at GCSE and A Level students to ensure my design is aesthetically pleasing and can be easily interacted with. This will ensure that my final design is as satisfying as possible.

2.3.6 Design Drawings

2.4 Validation

2.4.1 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 specific ranges. 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.

2.5 Testing

2.5.1 Iterative Test Plan

To ensure that my program is working properly I will use a range of data to verify that the program interprets the data correctly and processes it without any errors.

To ensure that the tests are carried out properly I will need to use data that mimics every possible input the user can access, such as a range of numbers and characters.

This is required because I need to test that the program correctly processes the data that is supplied to it. All the possible inputs are needed so that a bug doesn't go unnoticed through the iterative development process. This could cause much larger problems further along the development cycle.

I will be testing my application using unit tests. Each function that I implement will have a range of unit tests ensuring that it works correctly. The unit tests will take a variety of inputs that cover any possible scenario that the user can do. Therefore, the unit tests will ensure that the application responds properly to every possible input for every function.

An example test for the addition / subtraction function:

Input	Reason	Expected	Actual	Pass/Fail and Actions
1 + 2	Adds two positive numbers	3		
1 + + 2	Two addition operators should result in an error.	Error Message		
1 + - 2	Adds/subtracts a negative and positive number.	-1		
-1 - 2	Subtracting 2 negative numbers	-3		
-1 + - 2	Subtracting 2 negative numbers with an addition operator	-3		
1 - - 2	Two subtraction operators should result in addition.	3		

2.5.2 Testing of Final Implementation

The final implementation will be tested with a white box method. This will be done by myself as I will be most knowledgeable about the program and so I will be most qualified to test every aspect of the program.

This is an important testing method to use because it will ensure that every function of the programs works as intended. This may not happen with black box testing because the people testing the program are not familiar with it and therefore they could miss errors.

Therefore, white box testing is very important as the developer is most likely to recognise every error that occurs so the final implementation will be as bug free as possible.

I will be testing the final implementation's user interface using black box testing. I will create a survey that has brief instructions of how to use the program and ask people who have little knowledge of the system to follow the survey's instructions to use the program.

I can use the survey to track how easily people can interact with my program with minimal guidance. This should test how accessible and user friendly my final implementation is. The use of a survey will also make it easy to get feedback on the user interface to make any potential improvements.

The use of black box testing will provide an unbiased perspective on the app. This will generate valuable feedback that I would not otherwise be able to get using other testing methods.

The final method that will be used is destructive testing. In this testing method, the developer tries to break the system when in full use.

It's important to use this method of testing with the final implementation because the final product should not have any fatal bugs that lead to a full crash. Destructive testing will reveal any of these potential fatal errors so safe guards can be put in place.

This type of testing will therefore ensure that user is provided with the cleanest possible experience and never experiences any fatal errors that crashes the application and interrupts the user.

In conclusion, I will use a combination of white, black and destructive testing because this will provide the largest range of different tests which should cover any potential errors that will affect the user's experience.

3. Development

3.1 Success Criteria and Testing

3.1.0 Version 0.1.0

V0.1.0 – Success Criteria

In Version 0.1, I attempted to complete the success criteria:

0. "Functionality for the four basic operators, +, -, *, ÷."
1. "Follow the rules of BODMASS."
2. "Have functionality for using square roots and powers."

This version covers the success criteria that make up the most basic functionality that a calculator.

For Version 0.1.0 to be successful, the calculator should be able to calculate any basic equation that includes the four basic operators, "+, -, *, ÷" and powers and roots.

Version 0.1.0 was started by creating an XML layout file which contained all the required parts of the calculator, such as the number input buttons, the calculator display, the operator buttons, a delete button, etc.

The layout was created using a *Constraint Layout*, instead of a *Relative* or *Linear layout*. A *Constraint Layout* is where each view has constraints that act as connections to other views, the parent layout or an invisible guideline.

A *Relative Layout* is where the views are positioned relative to each other, while a *Linear Layout* is where the views are organised into a single horizontal or vertical row.

I have chosen to use a *Constraint Layout* because although it is initially more complicated to understand and implement, you can build much more flexible layouts with it. This will be especially helpful in this project as the *Constraint Layout* will mean the layout will scale and change appropriately depending on different size devices.

In the *MainActvitity* class, variables were linked to their corresponding XML element. For example, the buttons were linked to Button variables and the calculator display was linked to a *TextView* variable.

The buttons variables were set an *OnClickListener* so they called the *onClick* method when tapped.

A switch case statement was used where cases are dependent on the ID of the clicked button. Each button has a unique ID that's defined in the layout's XML file.

Each button case adds the input of the button into the expression so it can later be used in the *ShuntingYard* class.

There are also special cases, such as the "AC" button which wipes all data, and the "DEL" button which removes the last character of the expression.

When the equals button is pressed ("="), a validation is first run to ensure that the expression string is not empty.

This is done to prevent an error occurring when the string is passed to the *ShuntingYard infixToPostfix* method.

If the validation check is failed, the stack trace is logged and an error message is shown to the user.

If the validation check is successful, the answer is passed to the *ShuntingYard* method called *infixToPostfix*. This method converts the user's expression into Reverse Polish Notation.

In this method, a string called *ops* is used to define each operator's precedence. The precedence is found by the index of the operator divided by 2.

A for-each loop is used to iterate through the user's expression by each character. If the current character is empty, a *continue* keyword is used to terminate the processing of the current iteration and continue with the for-each loop.

Variables are made to denote the current character and the index of the character in the operator string. If the character is not an operator, it will not exist in the string and therefore getting its index will return "-1".

If the index is returned as "-1", and the stack is currently empty, the value of the index is pushed onto the stack.

If the stack is not empty, a while loop is used so that while the stack is not empty, the value of the precedence before the current character is retrieved by using "*stack.peek() / 2"*. The *peek* method is used to get the value of the top of the stack. This will be the index of the previous character in the *ops* string and dividing it by two returns the precedence value. This is stored in a variable called *prec2*.

The precedence of the current character is calculated by dividing the index of the character in the *ops* string by 2. This is stored in a variable called *prec1.*

If the previous operator's precedence is greater than the second one or they are equal (if the current operator is not a power), the index of the previous operator is popped off the stack and its corresponding operator character is appended to the postfix string.

When the while loop ends (i.e. the expression is empty), the current index is pushed onto the stack.

If the index does not equal "-1", and the character equals an opening bracket "(", the value of "-2" is pushed onto the stack to represent the start of the bracket.

If the current character is a closing bracket instead, a while loop is used to add characters to the postfix string. The condition for the while loop is while the previous element in the stack does not equal "-2" (the integer representing an opening bracket).

This means that the part of the expression inside the bracket will take precedence over everything else and therefore obey the rules of BODMAS properly.

If none of these conditions (index != -1, character = "(" or ")") are met i.e. the current character is just a digit, the digit is appended to the postfix string.

Finally, the *infixToPostfix* method returns the string it has made.

In the *MainActivity* class, the postfix can be found by calling the *infixToPostfix* method on the user's input expression string.

The *evaluateRPN* method can then be used on the postfix string to calculate the answer to the user's expression.

The *evaluateRPN* method works by first creating a stack with a data type of double called “tokens” and then running a for-each loop where the loop iterates over an array created from the postfix that has been split up on the whitespace.

Each iteration, the method uses a HashMap made using a for-each loop which goes over the *enum* *Sign* which contains all the operators and puts each operator into the HashMap.

The *evaluateRPN* can then use the *find* method on the *HashMap* with the parameter of the current token to see if it is an operator. If the token is not an operator, it will return “null”.

If the token is an operator, the operator is applied to the last two elements of the stack using the *calcSign* method and the *enum* data type (which contains all the operators and returns the result for each one).

The *calcSign* method returns the stack after the tokens have been popped off, the result of the two with the operator has been calculated and then the result has been pushed back on.

If the current token is not an operator, it must be a number and therefore the token is converted from a string to a double and then pushed onto the stack.

At the end of the iteration, the only thing left in the stack will be a single double data type which is the result of the user’s input expression.

This value is returned where it can be displayed to the user using a *TextView*.

The source code that handles this is shown in Appendix 5.1.1 and 5.1.2.

Version 0.1.0:

V0.1.0 - Testing

The first of these success criteria is the four basic operators:

These screenshots show how the calculator properly handles the operators “+, -, x, ÷”. The input takes “2*3+1” and outputs “7”. This is correct and therefore shows that the calculator is handling multiplication and addition correctly.

The calculator also takes “6÷3-1” and outputs “1”. This is the correct result and there for shows that the calculator is performing division and subtraction correctly.

These screenshots also show that the calculator is properly following BODMAS when completing the calculations. This is because when inputted with “2*3+1”, the calculator first does “2*3” and then “+1”.

This is correct because multiplication takes precedence over addition in BODMAS. Therefore, when using the basic operators BODMAS is correctly used. This means that the first and second success criteria aimed to be completed in Version 0.1 were successful.

The last success criteria in Version 0.1 is for the calculator to “have functionality for using square roots and powers”.

These screenshots show how Version 0.1 of the calculator handles powers and how they interact with BODMAS.

In the first screenshot, the calculator interprets the equation as "$2x3^3$". Therefore, it follows BODMAS and calculates "3^3" to give "27" because powers have precedence over multiplication. It then multiples "27" by 2 to result in "54". This is an example of the calculator working correctly.

However, in the second screenshot the equation "$2(3^3)$" is inputted. This is using a different way of expressing the exact same equation as the first screenshot.

The calculator should be interpreting this expression as "$2*(3^3)$". However, the calculator does not recognise a number in front of brackets as shorthand for "$2*(3^3)$. Therefore, the base and power does not get multiplied by two and the output is incorrect.

Another problem with version 0.1 is that the functionality for roots was not completed in the first iteration of the iterative development process. However, this will be resolved in the next iteration.

Reason	Input	Expected	Actual	Required Actions
Test basic operators and how they work with BODMAS.	2*3+1	7 =(2*3)+1	7	N/A
Test basic operators and how they work with BODMAS.	6/3-1	1 =(6/3)-1	1	N/A
Test that powers are calculated correctly.	3^3	27	27	N/A
Test BODMAS precedence in an expression with different operators but no brackets to simplify it.	$2*4 -2^2$	4 $=(2*4)-(2^2)$	4	N/A
Test how brackets are handled when used as shorthand for multiplication.	$2(3^3)$	54 $=2*(3^3)$	27 $=(3^3)$	Calculator does not recognise a number next to a bracket as shorthand for multiplication and instead ignores the number. Actions: When interpreting the user's input expressions, insert multiplication operators where needed before evaluating the expression.
Test that square root function works correctly.	sqrt(4)	2	-	Root functions have not yet been created. Actions: Create a function that takes two inputs, the base and the root. It will root the base by the root variable and output the result to the user.
Test "DEL" button to make sure it deletes one operator/number at a time	Tap "DEL" on any expression. E.g.: 2-2+	2-2	2-2+	When the user inputs operator characters, the application automatically inserts " " characters either side of the user's input. This is done so the Reverse Polish Notation is created correctly. The "DEL" input does not include the whitespace that was inserted by the application. To delete this as well, the user needs to input "DEL" multiple times. Actions: Currently, the "DEL" button works by removing the last

				character of the expression string. To fix this, first remove all whitespac e in the string. Then remove the last character. Finally, the whitespac e can be inserted back into the string.
Test "AC" button to ensure that it deletes the entire expression and wipes anything in the calculator's memory that should be non-volatile.	Click "AC" on expressio n: 2+3	Wipes current display and everythin g stored in memory.	Wipes current display and everythin g stored in memory.	N/A

3.1.1 Version 0.1.1

V0.1.1– Success Criteria

V0.1.1 involves completing the success criteria for:

- "Have functionality for using … roots"
- Fix various bugs introduced in version 0.1.0 such as the *DEL* function behaving improperly and correcting how using brackets as shorthand for multiplication is handled.

This version covers the second iteration of the implementation of the first 3 basic success criteria.

This iteration is made up of fixing bugs that were introduced in the first iteration. This includes the errors where the "DEL" button was not working correctly because it only deleted the last character of the expression string variable.

This bug was fixed by removing all whitespace from the expression, then deleting the last character from the string. Then whitespace was inserted back into the string. This was done using the string function *.replaceAll* and the regex expression *"('.(?=.)', '$0 ')"*.

This regex expression replaces all characters in the string with itself and space except the last character.

In this iteration, the bug where using brackets as shorthand for multiplication was not interpreted correctly was also fixed.

This was fixed by creating a function that was called when an opening bracket was entered by the user.

This function checks if the character before it is a digit, e.g. "4(2)" or a closing bracket, e.g. "(2)(2)". If the check is positive, a *StringBuilder* is made and a multiplication sign and the appropriate whitespace is inserted into the string. This is done using the *StringBuilder* method *replace*.

This iteration also included the implementation of a root function. This was created by adding a button to the main XML file and linking it to the *MainActivity* class so it would call *onClick* when the button is clicked. When the root button is clicked, it inserts the radical sign (√) into the expression string and updates the calculator display to show this.

The radical sign was then included in the *ShuntingYard* class to assign its precedence in BODMAS when creating the Reverse Polish Notation. This ensures that the RPN is formed correctly so BODMAS is followed.

Extra parameters were also created to calculate the user's inputted roots. The roots are calculated by using Java's Math library: "*Math.pow(base, 1.0 / root)*". This is because mathematically √4 is equal to $4^{1/2}$ through indices laws.

Version 0.1.1 also included the addition of shortcut root and power buttons.

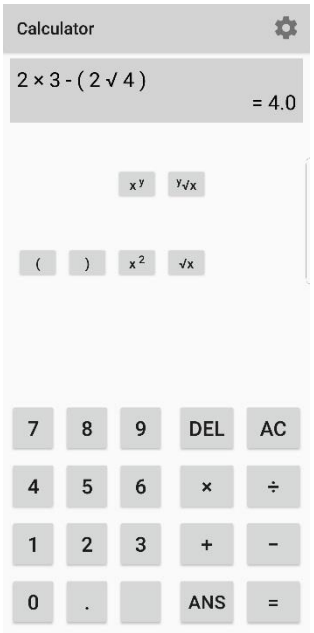
The shortcut power button automatically adds ^{2} to the user's base so the user does not need to enter their power as well.

Squaring numbers is the one of most common power used by students and therefore having a shortcut button makes the process of using the calculator much more efficient. This is important as it leaves the user to focus and spend their time and focus on other things.

The square root shortcut button acts similarly to this, it automatically adds a square root because square roots are the most common roots used.

These changes were made to Appendixes 5.1.1 and 5.1.2.

Version 0.1.1:



Another notable change in this version is that the size of the text in the calculator display was decreased to ensure that the user's entire expression and answer will be displayed clearly to the user.

V0.1.1 – Testing

Reason	Input	Expected	Actual	Required Actions
Test "DEL" button to make sure it deletes one operator/number at a time	Tap "DEL" on any expression. E.g.: 2-2+	2-2	2-2	N/A
Test how brackets are handled when used as shorthand for multiplication.	Input an expression where there's a number adjacent to an opening bracket, e.g.: $3(3^3)$	$3(3^3)$ = 54	$3*(3^3)$ = 54	N/A
Test that x/y root function works correctly.	Input any $\sqrt[y]{x}$ expression to test that the root and base are working correctly.	$\sqrt[3]{8}$ = 2	$\sqrt[3]{8}$ = 2	N/A
Test square root shortcut button works correctly.	Use the shortcut button to input any $\sqrt[2]{x}$ expression.	$\sqrt[2]{4}$ = 4	$\sqrt[2]{4}$ = 4	N/A
Test square power shortcut button works correctly.	Use the shortcut button to input any	2^3 = 8	2^3 = 8	N/A
	x^2 expression.			

3.1.3 Version 0.2.0

V0.2.0 – Success Criteria

Version 0.2 is intended to complete the success criteria:

- "Cursor controls that will move the cursor around the equation."

This means that for Version 0.2 to be considered successful, the app must have functionality that allows the user to traverse through the equation using cursor buttons. This will enable the user to edit their expression without either clearing it using *AC* or deleting part of it.

This was completed by introducing a variable that kept track of the user position in the expression's string.

This variable was incremented every time the user added to the expression and was decreased whenever the user deleted something from the expression.

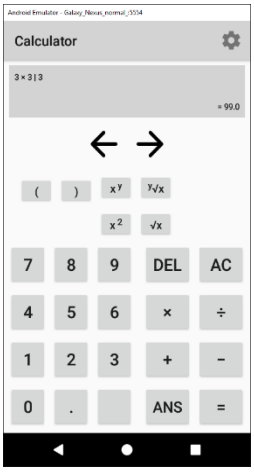
Two buttons were also added which were given the IDs of *shiftLeft* and *shiftRight*. They were both linked to a function that would either decrease or increase the position variable by one depending on which button was clicked.

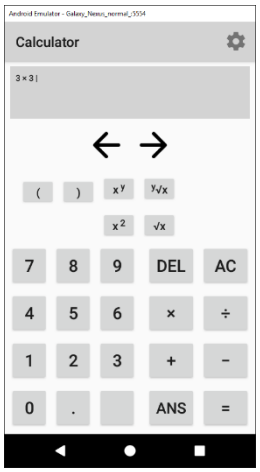
In this function, a vertical slash character ("|") is also added to the string in the current position index so the user has a graphical representation of where the position index is.

To finish the implementation of this success criteria, the method that handles the insertion of the user's input to the expression was changed from adding the input to the end of the string. Instead, it would use a StringBuilder and insert the user's input at the position index.

To make these changes, source code was added in Appendix 5.1.1, in the *shiftPosition* method.

Version 0.2.0





These screenshots show how the cursor can used to edit the expression by inserting a number and it will give the correct result.

V0.2.0 – Testing

Whilst testing Version 0.2.0, I noticed that the foundation of the project was not built as efficiently as it could have been, and this will affect the process of adding further features to the project.

The source of this inefficiency was located in the main calculator Java class which handled the user's interactions and the GUI. The *ShuntingYard* class (which handles the process of converting the user's input into an answer) did not experience these same problems.

The logic and structure of the code in the main activity was constructed improperly which created source code that was not easily understandable and therefore was difficult to maintain. This was especially clear as creating new features and adding them to the project became more complex as the project grew larger.

For example, each button the user could tap to enter input into the calculator was linked to an *OnClickListener* in the main activity. This overrode the android superclass *onClick* and handled each button separately using a switch-case statement. This was a very inefficient method to implement functionality for the buttons because it meant each button had to be programmed separately even when similar buttons did very similar things. E.g. tapping the "8" or "9" button does the same thing except enters a different digit.

Another effect of this illogical implementation is that the code being very long and complicated. This made maintaining the code almost impossible and making small changes to fix bugs became needlessly difficult.

Another example of how the codebase was built up incorrectly is a logical error that exists in the app. This logical error is that the variable that tracks the user's position in the expression was being incremented at the wrong time in the program's flow. This meant that when the position variable was used to insert input into the expression which often caused an off-by-one error.

This logical error was caused due to the lack of structure in the codebase which reduced the source codes readability and so lead to the code being difficult to understand and maintain.

In order to amend this, the next version will consist largely of refactoring the parts of the source code that handles the graphical user interface and the user's input.

For example, I will streamline the process of the user entering input. I will remove the many separate buttons and instead of using *OnClickListeners* I will use the *onClick* XML characteristic which will call a specific method.

With this, I can group the buttons into similar groups and each set of groups will call their own method. For example, all the digit buttons (0-9) will call the *inputDigit* method and all operator buttons (+, -, ÷, ×, etc.) will call the *inputOperator* method.

I can use the *tag* XML characteristic to define which input (i.e. which digit or operator) to insert into the string depending on which button was clicked.

3.1.4 Version 0.3.0

V0.3.0 – Success Criteria

The success criteria for Version 0.3.0 covers refactoring the parts of the source code that are relevant to the calculator's GUI and handling of the input. This does not include the *ShuntingYard* class.

The project was improved by changing the way the input buttons were handled to input the user's choices.

In Version 0.2.0, each individual was linked to the overridden *onClick* method using *OnClickListeners*. This led to having a very large function that handled each button specifically. This was incredibly inefficient and hard to manage whenever changes had to be made.

In Version 0.3.0 I made the changes of using the XML characteristic *onClick* to link each group of buttons (e.g. digits or operators) to their own method. In the method, the input was found using the XML characteristic *tag* where the tag was the value to be inputted into the expression.

Changing the project to use this method means that the amount of duplicate code is severely cut down and so the

project is made much more efficient to manage and to run. Because each group of buttons links to the same method, less code is required to achieve the same result.

This also means that when I make slight changes to the process, I only have to make one change to the button groups function rather than going through multiple cases and making individual changes.

The source code that handles the digit/operator inputs and the AC button input can be found in Appendix 5.1.1.

Another change made in V0.3.0 was to create new methods called *updateDisplay* and *removePositionMarker.* The *removePositionMarker* method removes all the underscores (character used to denote the user's position in the expression) in the expression.

This method is used in *updateDisplay* to ensure the string is clean before updating the position marker and then updating the *TextView* that shows the calculators output.

The method updateDisplay is called whenever the user makes a change to their expression such as inputting a number to traversing through the expression.

Using a separate method to handle this is improved upon the previous version because it reduces the amount of duplicate code required and makes the projects source code much more understandable.

However, some parts of the previous version did not require refactoring. For example, the method *validateBracketMultiplication* which allows the user to use brackets as shorthand for multiplication (e.g. "5(2) = 5*(2)") remained the same. This method already functioned properly and did not require any changes except being ported to the new class.

V0.3.0 – Testing

Reason	Input	Expected	Actual	Required Actions
Test digit inputs work correctly.	Any digit, e.g. "4"	GUI shows "4"	GUI is updated to show "4"	N/A
Test operator inputs work correctly.	Any operator e.g. "-"	GUI is updated to show "-"	GUI is updated to show "-"	N/A
Test bracket inputs work correctly.	Input "(" and ")".	GUI is updated to show brackets and they are interpreted correctly by the *ShuntingYard* class.	GUI is updated to show brackets and they are interpreted correctly by the *ShuntingYard* class.	N/A
Test expression to ensure they are being interpreted correctly by the *ShuntingYard* class.	An expression e.g. "4(3+2^2)"	"28" is outputted.	"28" is outputted.	N/A
Test whether shift left/right buttons update position correctly.	Tap left/right button.	Position increases if right button and decreases if left button.	Position increases if right button and decreases if left button.	N/A
Test whether shift buttons update the position marker (underscore character)	Tap left or right shift button.	Position marker moves to the character left or right of the previous one.	Position marker moves to the character left or right of the previous one.	N/A

3.1.5 Version 0.4.0

V0.4.0 – Success Criteria

Version 0.4.0 aims to complete the success criteria: "store relevant mathematical equations."

This means that for this version to be considered successful, there must be a way for the user to access a menu where they can select from a range of equations such as the Quadratic Equation, Pythagoras' Theorem, etc.

This was done by adding a new button to the main calculator layout which had the onClick characteristic set to call the function *onClickEquations* which starts the new activity, *Equations.*

This activity initialises the *RecyclerView* in the activity's corresponding layout in the overridden method *onCreate*. This method sets the RecyclerView's LayoutManager and adapter. The adapter is the class *EquationsRAdapter*.

The *RecyclerView* adapter populates the XML *RecyclerView* to show the user a list. In the method *EquationsRAdapter*, an *ArrayList* is created to make an array of strings that hold each equation's name. This array is later used to populate the RecyclerView.

In the method *OnBindViewHolder*, the strings in the *ArrayList* are set to *TextViews* in the *RecyclerView* and then the method *OnCreateViewHolder* inflates each view so they are displayed to the user.

In *OnCreateViewHolder, OnClickListeners* are also set on each element of the list. This is linked to an *onClick* method that handles the user's clicks.

In the *onClick* method, a switch-case statement is used on the variable *position*. This variable holds the index value of which element in the *RecyclerView* the user has clicked. Therefore, when the user clicks an element of the *RecyclerView*, it starts a different activity depending on the element clicked.

For example, if the user clicks the first element, (i.e. the Quadratic Equation element), it will start the activity that handles all Quadratic Equation.

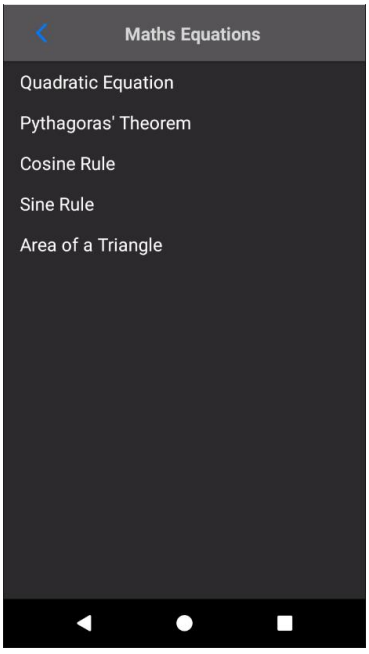
Also, in this version, a back button was added to the Toolbar to allow the user to return to the main calculator page.

This version has been successful because I have created an activity which displays the choices of the different equations to the user. These choices are "The Quadratic Equation", "Pythagoras' Theorem", "The Cosine Rule", "The Sine Rule" and the "Area of a Triangle".

In the next iteration of development, each equation that is relevant to the Maths GCSE will be added with the functionality.

The source code for the *Equations* and *EquationsRAdapter* classes can be found in Appendix 5.1.3 and 5.1.4 respectively.

The Equations RecyclerView:



V0.4.0 – Testing

Reason	Input	Expected	Actual	Required Actions
Click each element of the *RecyclerView* to ensure they are receiving the clicks.	Tap *RecyclerView* elements.	Activity responds correctly by sending user to the element's corresponding activity.	Activity responds correctly by sending user to the element's corresponding activity.	N/A
Tap the back button to ensure it send the user to the correct activity.	Tap back button.	User is sent to previous activity (*Calculator)*	Use is sent to previous activity (*Calculator)*	N/A

3.1.6 Version 0.4.1

V0.4.1 – Success Criteria

Version 0.4.1 will cover the success criteria off adding the Quadratic Equation. This is the first element in the list in the *Equations* class. When this element is clicked, the user is sent to the *QuadraticEquation* activity.

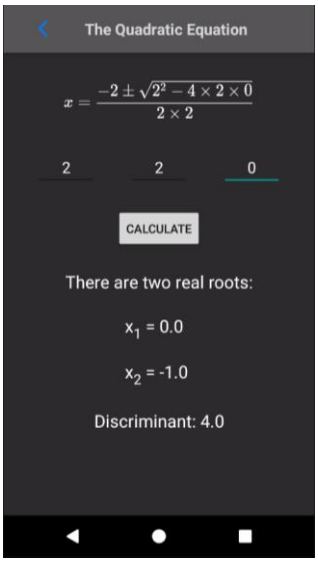
The quadratic equation is:

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The class *QuadraticEquation* handles all the relevant processing for this equation. In the *onCreate* method, the *contentView* is set to the layout that was created for this activity. The layout displays the equations general formula and 3 *EditTexts* where the user can enter their input.

Below this are *TextViews* that are updated with the result when the user clicks the *calculate* button.

The QuadraticEquation layout:



In *onCreate*, I added *TextWatchers* to each *EditText*. *TextWatchers* listen for input on the *EditText* they're attached to. By overriding the *TextWatcher* method *onTextChanged*, I can update the *String* that holds the variable value and call *updateDisplay.*

The method *updateDisplay* creates a new *String* called *quadraticEquation* that is then set as the text for the *MathView*. This process means that when the user changes the content of the *EditText*, the display equation will be updated in real time to display the user's input.

MathView is a library for display maths formula in Android apps. It was added to the project by implementing it in the project's Gradle file.

This library is essentially the same as a *TextView* except it can render *TeX* code into maths formulas. *TeX* code is a typesetting system. Using the *MathView* library is very helpful because it means that special characters like plus-minus signs and fractions can be displayed easily. This is important as it increases the readability of the equations and greatly improves the user experience.

The *TeX* code that was used to render the quadratic equation is:

```
String quadraticEquation =
String.format(
        "$$\\color{white}{x = \\frac{-
%2$s \\pm \\sqrt{%2$s^2 - 4 \\times
%1$s \\times %3$s}}{2 \\times
%1$s}}$$",
        aString, bString, cString);
```

- *$$* is used to signify the start and end of the *TeX* code.
- *\\color{white}{}* is used to change all the text in the empty brackets to the colour in the first brackets.
- *\\frac{a}{b}* is used to turn the content inside each bracket into fractions where the text in the first brackets are the numerator and the second are the denominator.
- ^ is used to put the text following the operator into superscript.
- *\\times* renders a multiplication sign.
- *\\pm* renders a plus-minus sign.
- *%1$s* and *aString* is string substitution where *%1$s* is replaced by the *String* stored in the *aString* variable.

This is rendered as:

$$x = \frac{-b \pm \sqrt{b^2 - 4 \times a \times c}}{2 \times a}$$

When the user clicks the *Calculate* button, it calls the method *calculateQuadratic*. This method handles all the processing required to calculate the answer for the user.

The method starts by closing Android's virtual keyboard so the user doesn't need to do it themselves. This makes the process smoother and improves the user's experience.

The following code is surrounded by a try-catch statement so that any *NumberFormatExceptions* are caught and an error message is displayed instead of the app crashing. *NumberFormatExceptions* are caused when the program tries to run the method *parseDouble* from the *Double* class on an empty string. In this case, that would mean that the user hasn't entered their input and so the error informs the user that they need to input values.

Inside the try-catch statement, firstly the discriminate is calculated. The discriminant is equal to "b^2-4ac" and the quadratic equation will give different outputs depending on the value of the discriminant.

After calculating the discriminant, an if-else statement is run where if the discriminant is greater than zero then the quadratic equation will have two unique roots. The two roots are then calculated inside this if statement and the results are displayed to the user.

If the discriminant is equal to zero, then the quadratic equation must have two repeated roots. If this is true, then only one root is calculated and then displayed to the user.

If the discriminant is not greater than or equal to zero, then it must be negative. In the quadratic equation, the discriminant is square rooted. This cannot be done without using imaginary numbers which are not part of the GCSE specification. Therefore, an error is outputted to the user to inform them of this.

The *QuadraticEquation* is found in Appendix 5.1.5.

V0.4.1 – Testing

Reason	Input	Expected	Actual	Required Actions
Test if back button sends user to correct activity.	Click back button	User is sent to *Equations* class.	As expected.	N/A
Enter input into the *EditTexts* to ensure display equation is being updated in real time.	Enter values into each *EditText.*	The display equation is updated to show the user's input.	As expected.	N/A
Calculate a quadratic equation that	a=2 b=2 c=0	Layout displays both roots and the discriminant.	As expected.	N/A

should result in two real roots.		X_1=0 and X_2=-1. Discriminant = 4		
Calculate a quadratic equation that should result in one repeated root.	a=9 b=12 c=4	Layout displays one root and the discriminant X = -2/3	As expected	N/A
Calculate a quadratic equation that should that have no real roots.	a=4 b=2 c=4	Layout shows message saying there are no real roots and that the discriminant equals "-60".	As expected.	N/A

3.1.7 Version 0.4.2

V0.4.2 – Success Criteria

Version 0.4.2 has the aim of adding the next equation to the calculator, Pythagoras' Theorem. When the Pythagoras' Theorem element is clicked in the *Equations RecyclerView*, it sends the user to the *PythagorasTheorem* activity

Pythagoras' Theorem is: $a^2 + b^2 = c^2$. In *onCreate* of *PythagorasTheorem*, a *MathView* is set to display the equation in its general formula. Much like *QuadraticEquation, EditTexts* are added with *TextWatchers* to update the display when the user enters input.

Pythagoras' Theorem layout:

When the user clicks the *calculate* button, it runs the method *calculatePythagoras*. In this method, the Android virtual keyboard is closed and a try-catch statement surrounds the code where the answer is calculated. This statement catches *NumberFormatExceptions,* which occur when *parseDouble* is run on a *String* that cannot be parsed. In context, this means that the user has not entered correct input into the *EditTexts.*

Pythagoras' Theorem has two different versions. The first is where the two sides are known and the hypotenuse is calculated ($a^2 + b^2 = c^2$). The second is where one side and the hypotenuse is known and the second side is calculated ($a^2 = c^2 - b^2$).

To implement this in the app, an if statement was included which tested if the *EditText* for the "c" variable was empty. If so, this means the user wants to calculate the hypotenuse and so the values for each side were found by running *parseDouble* on *aString* and *bString*. These variables hold the values for sides "a" and "b" respectively.

After this, the hypotenuse was calculated using Pythagoras' Theorem and displayed to the user.

If the input boxes for the variables "a" or "b" were empty, then it must mean that the user wants to calculate one of the sides. An exclusive or operator must be used here to ensure that only one variable was empty because Pythagoras' Theorem wouldn't work without at least one side.

If it is the variable "a" that was empty, the values for "b" and "c are fetched. A validation check must be run to ensure that the value for "b" is greater than the value for "c". This is because the hypotenuse must always be the largest side and if it is not, then the user must have made an error. Therefore, an error message is displayed to the user informing them that their values are incorrect.

If the validation check is passed, then the side "a" is calculated using Pythagoras' Theorem and displayed to the user.

If it is the variable "b" that was empty, the values for "a" and "c" are fetched. A similar validation check is run to ensure that the hypotenuse is greater and an error message is displayed if it is not.

If the validation check is passed, then the side "b" is calculated using Pythagoras' Theorem and displayed to the user.

The *PythagorasTheorem* class source code can be found in Appendix 5.1.6.

V0.4.2 – Testing

Reason	Input	Expected	Actual	Required Actions

Calculate the hypotenuse ("c").	a=4 b=3	Pythagoras' Theorem gives the result: 5.0	As expected.	N/A
Calculate side "a".	b=5 c=12	Using Pythagoras' Theorem gives the result: 12.0	As expected.	N/A
Calculate side "b".	a=5 c=12	Using Pythagoras' Theorem gives the result: 12.0	As expected.	N/A
Test back button to ensure it takes the user back to the *Equations* activity.	Click the back button.	The *Equations* class is loaded and the current activity is destroyed.	As expected.	N/A

3.1.8 Version 0.4.3

V0.4.3 – Success Criteria

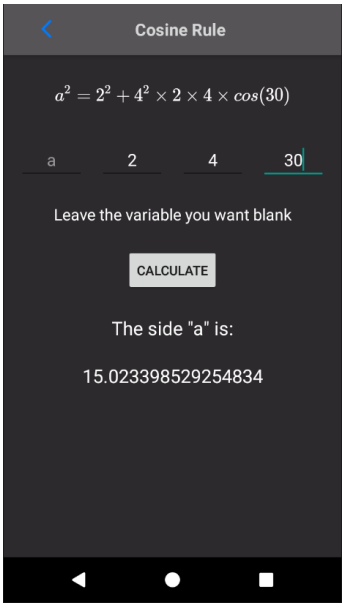
Version 0.4.3 will aim to complete activity that handles the Cosine Rule. This equation's activity will be started when the user clicks the corresponding element in the *Equations RecyclerView*.

In the *onCreate* method of *CosineRule*, the *MathView* that displays the equation to the user is set to display the Cosine Rule. This looks like:

$$a^2 = b^2 + c^2 - 2bcCos(A)$$

The activity's layout is created in a similar way to the other equations, where *EditTexts* are used for the user to enter their input and edit the display equation.

The CosineRule layout:



When the user clicks the "calculate" button in the activity's layout, it calls the *caclulateCosineRule* method in the *CosineRule* class. First, this method hides Android's virtual keyboard to clear up space on the screen.

The code required to calculate the Cosine Rule is surrounded by a try-catch statement. This will catch the *NumberFormatException* error which is caused by trying to parse a double from a string that doesn't contain a number.

The Cosine Rule has two different forms. The normal form is as shown above and is used to calculate the side of the triangle. This can be rearranged to:

$$Cos(A) = \frac{a^2 - b^2 - c^2}{2bc}$$

This form of the equation is used to calculate a specific angle of the triangle.

Both forms are required for Version 0.4.3 to be successful. To implement this, an if statement is run to check the *EditText* holding the input for the variable "a" is empty. If it is empty, then the user must want to calculate the side a, and therefore the values for the two other sides and the angle.

The angle values must first be converted to radians because as the Java library *Math* uses radians instead of degrees in the trigonometric functions.

The result can then be calculated using:

```
double a = Math.sqrt((b * b) + (c * c) * 2.0 * b
* c * Math.cos(angle));
```

This is then displayed to the user.

If the user does not want to calculate the side "a", then an if statement is used to see if the angle input is empty. If it is, then the user must want to calculate the angle. Therefore, the values for all the sides are parsed from the

member *String* variables. The result is then calculated using:

```
double angle = Math.toDegrees(Math.acos(((a * a) - (b * b) - (c * c)) / (2 * b * c)));
```

The calculated result is then displayed to the user.

If the user does not leave the *EditText* for either the "a" or the angle input then an error is displayed to the user informing them that their input is invalid.

The source code for the *CosineRule* class activity can be found in Appendix 5.1.7.

V0.4.3 – Testing

Reason	Input	Expected	Actual	Required Actions
Calculate side "a"	b=3 c=5 Angle A = 90	Calculator outputs side "a" = 3.0	As expected.	N/A
Calculate angle "A".	a=3 b=4 c=5	Calculator outputs angle "A" = 143.13 degrees.	As expected.	N/A
Test back button to ensure it sends the user to the correct activity.	User taps back button.	The previous activity, *Equations* is started and *CosineRule* is destroyed.	As expected.	N/A

3.1.9 Version 0.4.4

V0.4.4 – Success Criteria

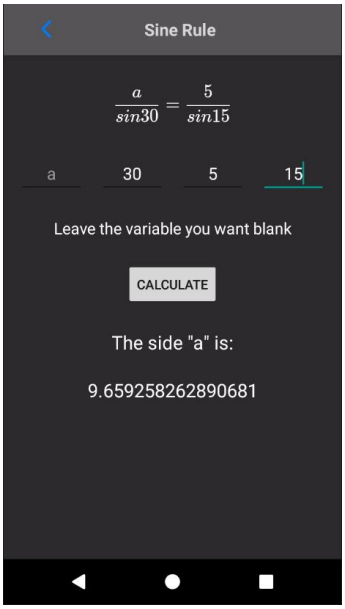
Version 0.4.4 aims to complete the *SineRule* activity. This activity is opened when the user taps on the corresponding element in the *RecyclerView* in the *Equations* activity.

In the *onCreate* method of the *SineRule* activity, the text in the *MathView* in the layout is set to the Sine Rule equation:

$$\frac{sinA}{a} = \frac{sinB}{b}$$

A similar layout is used with this activity as with the other equation layouts. This means that the *EditTexts* can be used to update the display equation in real time.

The SineRule layout:



When the user clicks the "calculate" button, the virtual keyboard is closed and, inside a try-catch statement to prevent crashes and output errors instead, the result is calculated.

The result is calculated differently depending on which variable the user left empty (i.e. which variable the user wants to calculate).

The different variables can be split up into two groups, the sides and the angles. If either of the side inputs ("a" or "b") are left empty, then the angle variables must be known and so they are retrieved and converted to radians.

An if statement is then run to determine if the "a" or "b" input was left empty. If it was the "a" input, then the value for side "b" is retrieved and the variable "a" is calculated. This is then outputted to the user using the *TextViews* in the layout.

If the "b" input is left empty, the value for side "a" is retrieved and used to calculate the result, which is then displayed to the user.

The sides are calculated with the expression:

```
double a = Math.sin(angleA) * (b / Math.sin(angleB));
```

If the user wants to calculate an angle instead, the values for the sides are retrieved. Another if statement is then used to decide whether to calculate angle "A" or "B".

If angle "A" is chosen, then the value for angle "B" is retrieved, converted to radians and then used in the Sine Rule to calculate the result. This answer is then displayed to the user.

The same thing is done if angle "B" is chosen, but with reversed variables.

The angles are calculated with:

```
double aAngle = Math.toDegrees(Math.asin(a * Math.sin(bAngle) / b));
```

If neither the angles nor sides are left empty, then an error is displayed to the user. This error informs the user that they need to leave one input box blank.

The source code for the *SineRule* activity can be found in Appendix 5.1.8.

V0.4.4 – Testing

Reason	Input	Expected	Actual	Required Actions
Calculate side "a"	Angle A = 15 Angle Angle B = 15 Side b = 5	Side a = 5	As expected.	N/A
Calculate side "b"	Angle A=15 Angle B=15 Side a = 5	Side b = 5	As expected.	N/A
Calculate angle A.	Side a = 2 Side b = 3 Angle A = 30	Angle A = 19.47	As expected.	N/A
Calculate angle B.	Side a = 2 Side b = 3 Angle A = 30	Angle B = 48.59	As expected.	N/A
Tap back button to ensure it sends user back to correct activity.	Tap back button.	User is sent to previous activity – *Equations.*	As expected.	N/A

3.1.8 Version 0.4.5

V0.4.5 – Success Criteria

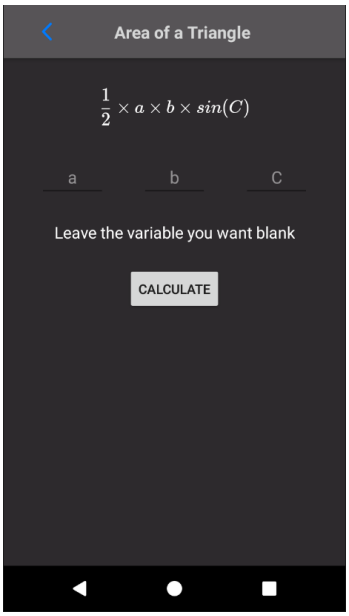
Version 0.4.5 has the aim of adding the Area of a Triangle to the list of equations. This is done by first adding the Area of a Triangle to the *Equations* activity *RecyclerView.* When the user taps this element, it sends the user to the *AreaTriangle* activity.

In the *onCreate* method of *AreaTriangle*, a *MathView* is used to display the general equation:

$$A = \frac{1}{2}abSinC$$

A similar layout to the ones used in the other equations is used here.

Area of a Triangle layout:



When the user clicks the "calculate" button, it runs the method *calculateAreaTriangle*. In this method, the virtual keyboard is closed to free up screen space and a try-catch statement is used to prevent crashes from *NumberFormatExceptions*.

The variables for the sides and angle are retrieved and the angle is converted to radians for use in Java's Math library. The area is then calculated using the code:

```
double area = 0.5 * a * b * Math.sin(cAngle);
```

The result from this is then displayed to the user.

V0.4.5 – Testing

Reason	Input	Expected	Actual	Required Actions
Calculate area "A".	a = 3 a = 5 C = 30	A = 3.75	As expected.	N/A
Tap back button to ensure it sends user to correct activity.	Tap back button.	User is sent back to *Equations* activity.	As expected.	N/A

3.1.9 Version 0.5.0

V0.5.0 – Success Criteria

In version 0.5.0, I aimed to fulfil the success criteria:

- "Conversions between different base number systems, namely, binary, hexadecimal, and denary."

To implement this, a button was added to the main *Calculator* activity which, when clicked, started the new *Conversions* activity.

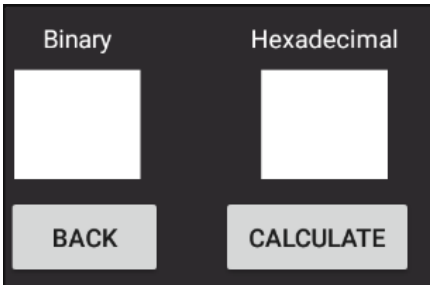
In this activity, the class *ConversionsRAdapter* is used to create and populate the *RecyclerView* in the activity's layout. The *RecyclerView's* elements are populated with the different conversions the user can perform. In this version, the possible conversions will consist of:

- Denary to binary
- Binary to hex
- Denary to hex

And vice a versa. In later versions this list may be expanded to include conversions between common units such as miles to kilometres and litres to millilitres.

When the user taps an element of the *RecyclerView*, the *OnClickListener* recognises which element of the *RecyclerView* was clicked and creates a *Dialog* object. The content view of this object is set to the conversion XML layout file that I created.

The Dialog layout:



OnClickListeners are added to the buttons in the *Dialog's* layout and the *Dialog* is displayed using the *show* method.

The layout has two buttons, a "back" and a "calculate" button. When the user taps the "back" button, the *Dialog* is dismissed, and the user is returned the conversions *RecyclerView*.

When the user taps the "calculate" button, the user's input value will be retrieved from the *EditText*. By seeing which *EditText* is empty, the calculator can decide which number system to convert to and from. For example, when converting between denary and binary, if the user has entered a value in the denary *EditText* and left the binary *EditText* empty, the calculator will convert the given value to the unknown.

The values are converted using predefined methods such as:

- *Integer.toBinaryString(STRING)*
- *Integer.toHexString(STRING)*
- *Integer.parseString(STRING, BASE)*

Where "STRING" represents a String variable and the number denotes the base number system that the string is in.

If the user leaves both *EditTexts* or enters a value into both of them, an error message will be displayed explaining that one input box must be left empty for the conversion be successfully calculated.

V0.5.0 – Testing

Reason	Input	Expected	Actual	Required Actions
Ensure denary to binary conversions are correctly calculated.	24	11000	11000	N/A
Ensure binary to denary conversions are correctly calculated.	11000	24	24	N/A
Ensure Binary to Hex conversions are correctly calculated.	1010	A	A	N/A
Ensure Hex to Binary conversions are correctly calculated.	A	1010	1010	N/A
Ensure Denary to Hex conversions are correctly calculated.	58	3A	3A	N/A
Ensure Hex to Denary conversions are correctly calculated.	3A	58	58	N/A
Ensure "back" button closes the Dialog correctly.	Click "back" button.	The Dialog is dismissed.	As expected.	N/A
Ensure "calculate"	Click "calculate" button.	The correct values are retrieved	As expected.	N/A

button calls the co		and used correctly to provide the user with a result.		

3.1.10 Version 0.6.0

V0.6.0 – Success Criteria

The aim of version 0.6.0 was to fulfil the success criteria:

- "Be able to use sin, cos, and tan and the inverses of."

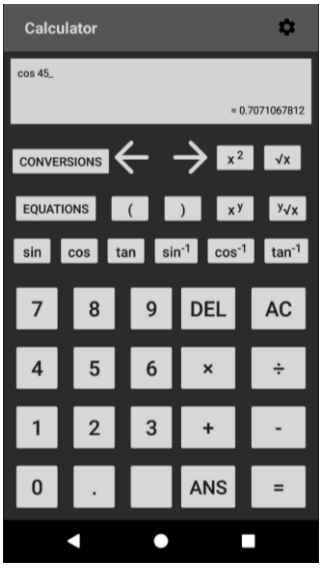
In the process of implementing this feature, new buttons for each trigonometry function and their inverse was added to the main calculator layout. These buttons were linked the main *Calculator* activity using the XML characteristic *onClick*. This characteristic calls a specific method when the button is clicked. In this case, the button calls the method *inputTrigonometry.*

This method gets the tag of the view that called. The tag is a string that contains the specific operator denoting the trigonometric function unique to each button. The operator is inserted into the infix using a *StringBuilder*.

The corresponding operators for each function were added the *String mOps* so that the function's precedence could be calculated when the infix is translated to the Reverse Polish Notation.

The *ShuntingYard enum, Sign* was also updated to include conditions for all the trigonometric functions and their inverses. Therefore, when the method *evaluateRPN* is called, the trigonometric functions can be calculated and used to calculate a result for the user.

For example:



V0.6.0 – Testing

Reason	Input	Expected	Actual	Required Actions
Ensure sine function gives correct result.	sin90	1.0	As expected.	N/A
Ensure cosine function gives correct result.	cos90	0.0	As expected.	N/A
Ensure tangent function gives correct result.	tan45	1.0	As expected.	N/A
Ensure inverse sine function gives correct result.	arcsin0.5	30.0	As expected.	N/A
Ensure inverse cosine function gives correct result.	arccos0.5	60.0	As expected.	N/A
Ensure inverse tangent function gives correct result.	arctan0.5	26.6 (to 3 significant figures)	As Expected.	N/A
Ensure trigonometric functions handle brackets properly.	sin(45*2)	1.0	As expected.	N/A
Ensure trigonometric functions are calculated with proper precedence.	sin90 * 2	2.0	As expected.	N/A

3.1.11 Version 0.7.0

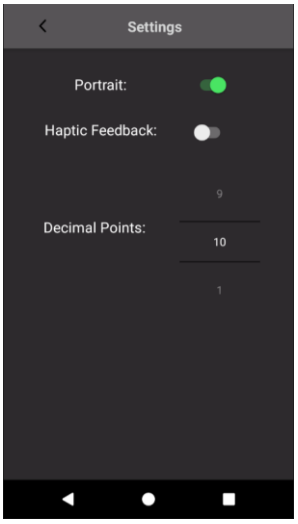
V0.7.0 – Success Criteria

The aim of version 0.7.0 was to create and add functionality to a settings page. The settings page will complete the success criteria:

- "The capability to change the layout of the app"

To complete this, a layout was created to provide the user with an interface with which to alter their settings. This layout includes an orientation setting, a switch for haptic feedback and a *NumberPicker* for the user to select an amount of decimal points to be displayed.

Settings layout:



The layout was made accessible for the user by adding functionality to the gear *ImageButton* of the *main_calc* layout. When clicked, the *ImageButton* calls the method *onClickSettings* which starts the *Settings* activity.

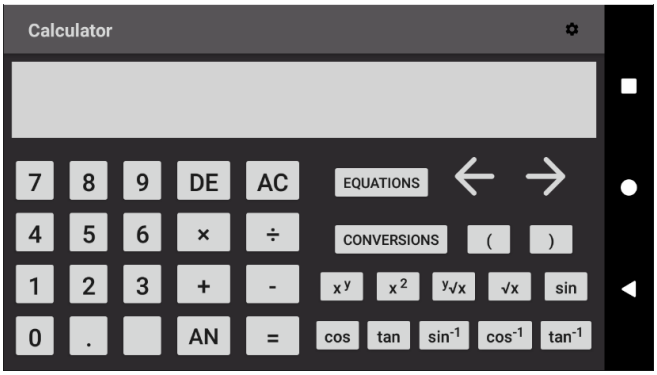
In the settings layout, a *onCheckedChangeListener* is set on the orientation switch. When the switch is toggled, the listener runs an *onCheckedChanged* method which alters the label *TextView* to either "Portrait" or "Landscape" depending on the switch's state. The method then saves a Boolean *isPortrait* to the *SharedPreferences* and updates the current orientation.

As there is no way to globally lock the orientation throughout the app, it must be done in each activity. Therefore, in the *onCreate* method of each activity, an if statement is used to check the status of the *isPortrait* Boolean. The orientation is then locked to portrait or landscape, depending on the variable's value.

Most of the application's layouts were created to work in either landscape or portrait by using *ConstraintLayouts* which adapted to the different orientations and device sizes. However, the main *Calculator* layout didn't adapt properly. This led to some buttons being pushed off the page, becoming inaccessible and the display becoming cluttered.

To remedy this, an alternative layout was created to be used when in landscape while the default can be used for portrait.

This is as shown below:



V0.7.0 – Testing

Reason	Input	Expected	Actual	Required Actions
Ensure orientation label is changed on switch toggle.	Toggle orientation switch.	Label is changed to represent the switch's value. On for portrait, off for landscape.	As expected.	N/A
Ensure other activities load in the correct orientation.	Change orientation to landscape, then navigate to the *Calculator* activity.	Layout is shown as landscape.	As expected, with the exception of the main *Calculator* layout which adapted improperly.	Create a alter layout for the landscape version.
Ensure back button sends user to correct activity.	Tap back button.	User is sent back to the main *Calculator* activity.	As expected.	N/A.

3.1.12 Version 0.7.1

V0.7.1 – Success Criteria

The aim of version 0.7.1 was to add functionality to the haptic feedback setting.

This was done by setting a *onCheckedChangeListener* to the haptic feedback *SwitchCompat*. When the switch's state is changed by the user it calls a *onCheckedChange* method. This method then edits the *SharedPreferences* by adding a Boolean variable called *feedbackIsChecked.* The variable is *true* when the switch is checked and *false* when it isn't.

In the other activities, this variable can be used to provide haptic feedback to the user. To do this, in the *onCreate*

method of the activity the *SharedPreferences* are initialised and the value for the variable, *feedbackIsChecked* is retrieved. If the variable does not exist in the *SharedPreferences*, the value defaults to *true*. This occurs when the user hasn't made a change to their settings so nothing has been saved.

To create the haptic feedback, a *Vibrator* is initialised. If the *feedbackIsChecked* variable is true, the *Vibrator* is used to vibrate the phone for 100 milliseconds in the methods that are called when the user clicks a button.

V0.7.1 – Testing

Reason	Input	Expected	Actual	Required Actions
Ensure value is saved in *SharedPreferences* when user changes the switch's state.	Change state of switch to the on position.	A *Toast* that displays the value of the variable displays "true".	As expected.	N/A
Ensure that haptic feedback is provided when enabled.	Change state of switch to the on position and test every button.	When the button is clicked, the phone vibrates for 100 milliseconds.	Some buttons didn't vibrate the phone.	Ensure that every method called by buttons has the code required to vibrate the phone.
Ensure haptic feedback isn't provided when it is disabled.	Change state of the switch to the off position and test the buttons.	When the buttons are clicked, they do not vibrate the phone but function as normally.	As expected.	N/A

3.1.13 Version 0.7.2

V0.7.2 – Success Criteria

The aim of version 0.7.2 was to add functionality to the *NumberPicker* so the user can select the amount of decimal points they want to be displayed when making calculations.

This was implemented by setting an *onValueChangedListener* to the *NumberPicker*. This means that when the user changes the value of the *NumberPicker*, a *onValueChange* method is called. The method saves the user's new value in the *SharedPreferences* with the variable name *decimalPoints*.

In the *onCreate* method of the *Calculator* activity, the user's value is retrieved from the *SharedPreferences.* A specific amount of decimal points can be displayed by using the *Math* method *round*. This method rounds the given value to the nearest integer. Therefore, a rounded value can be found. by multiplying the value by 10 to the power of the number of decimal points required, rounding to the nearest integer and then dividing by the number of decimal points.

For example, to round the value "11.009" to 2 decimal points:

- Multiplied by 100 = "1100.9"
- Rounded to nearest integer = "1101"
- Divided by 100 = "11.01".

The gives the correct result of "11.01".

To implement this into my application, the user's given value, which could range from 1-10 must be translated into a value of $10\text{-}1x10^{10}$. This was done by using Java's *Math* library to raise 10 to the power of *decimalPoints* to calculate the value that should be used when rounding.

In the method *submitInfix*, after the answer is calculated from the *ShuntingYard* method *evaluateRPN*, the *round* method from Java's *Math* library is used with the rounding value that was calculated to calculate the answer to the correct amount of decimal points. This value is then displayed to the user as usual.

V0.7.2 – Testing

Reason	Input	Expected	Actual	Required Actions
Ensure user's value is saved properly in *SharedPreferences*.	Change value in *NumberPicker* to "5"	A *Toast* that displays the value of the saved value displays "5".	As expected.	N/A
Ensure result is rounded correctly to user's given value.	Decimal points set to "5". Input "10 / 3" in *Calculator* activity.	Result is: 3.33333	As expected.	N/A
Ensure result is rounded to default value (10 decimal points) when user has not selected a specific value.	Clear *SharedPreferences* and input: "10 / 3"	Result is: "3.3333333333"	As expected.	N/A

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 2^2 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:

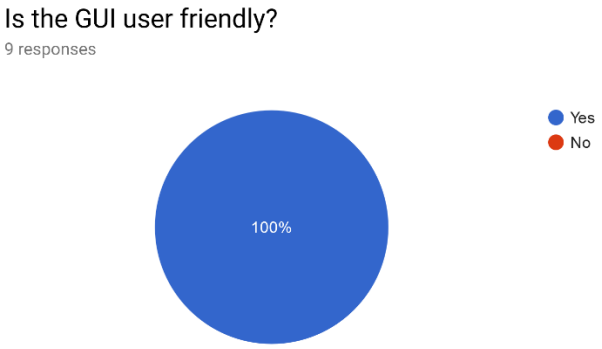
This increases the usability of the app as it makes results much more readable when they are larger.

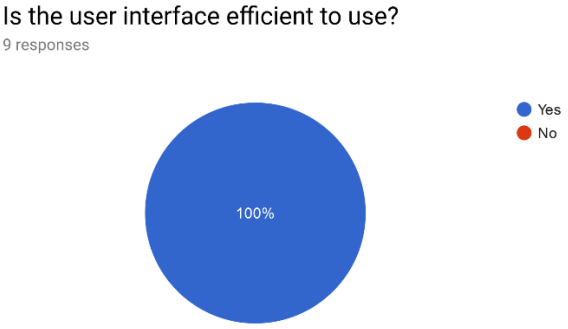
However, this success criterion is only partially met because fractions were not implemented. This is because due to time constraints they could not be included. In future iterations of the application, I would add a button to the *Calculator* activity where the user could switch between each notation.

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."

This success criterion was met as the survey proves:





As 100% of the responses were positive, success criterion is fully met.

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: $a = \frac{b}{c}$.

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, $a = \frac{b}{c}$ 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 10

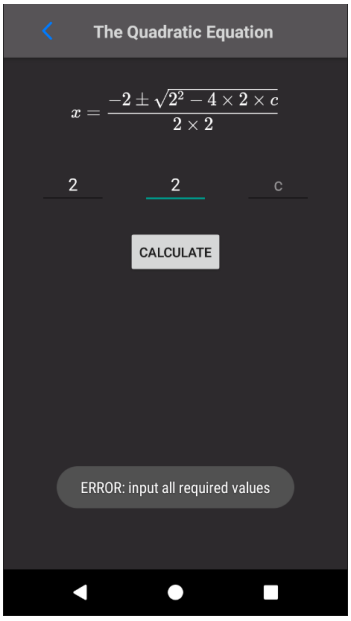
- "Detailed, helpful error messages that inform the user."

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 11

- "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.

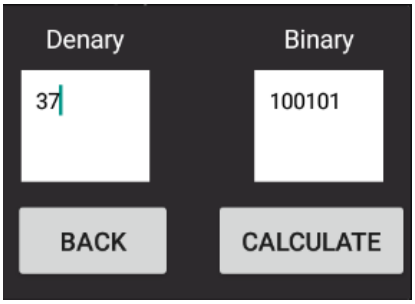
4.1.11 Success Criterion 12

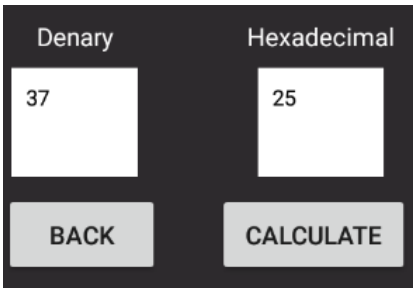
- "Conversions between different base number systems, namely, binary, hexadecimal, and denary."

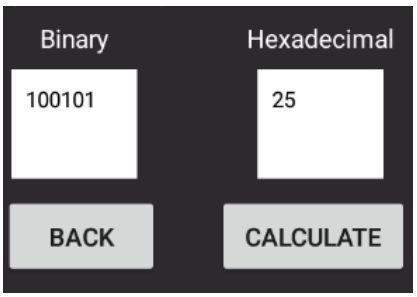
This success criterion was implemented by creating a button in the *Calculator* activity that sent the user to a *RecyclerView* containing each conversion. (As shown in Appendix 5.2.8).

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 13

- "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.

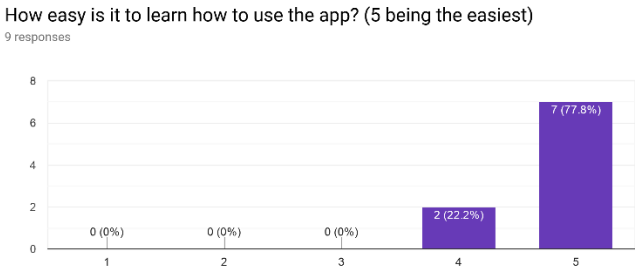
4.2 Justification of Usability Features

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.

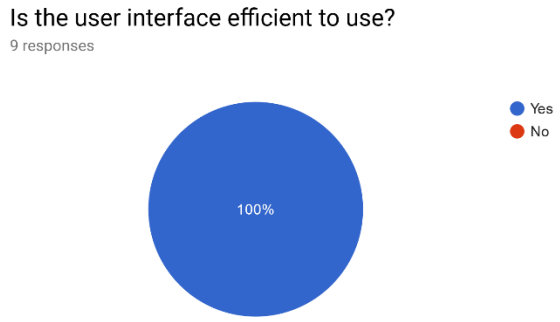
Every White Box tester found that the app was very easy to learn and became capable of using the features quickly. Therefore, I can conclude that the learnability aspect of the solution requires no more improvements.

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.

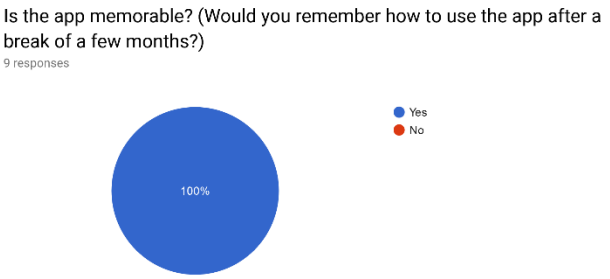
This is likely due to the design of the app being based on other calculators so users did not have to relearn the format and were already familiar with the interface. Because of this, efficiency in the White Box testers was especially high.

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.

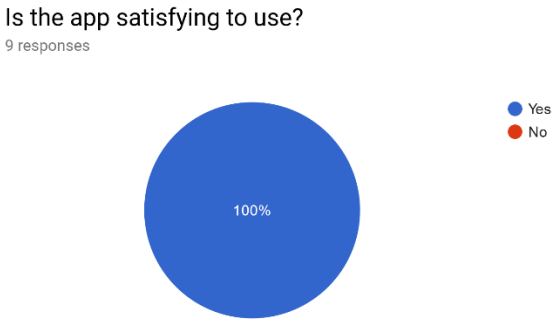
Therefore, I can conclude that there are no fatal errors that would cause the app to completely crash.

4.2.5 Satisfaction

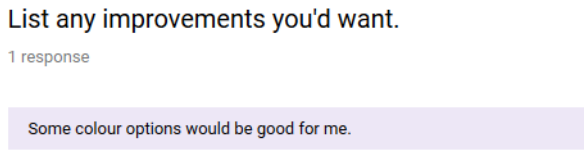
Satisfaction tests how enjoyable the user's experience of the software is.

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:

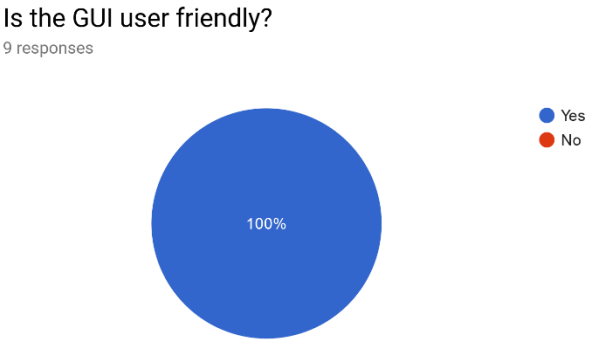


This means that some testers found that aspects of the design were unsatisfactory and detracted from the user experience.

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 and doesn't exceed 100 characters per line 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 $\frac{x}{y}$ rather than x/y. 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 "2^3". 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 its capabilities.

5. Appendix

5.1 – Source Code (Java)

5.1.1 *Calculator.java*

```
package com.calculatorproject;

import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.pm.ActivityInfo;
import android.os.Bundle;
import android.os.Vibrator;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.text.Html;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.EmptyStackException;

public class Calculator extends AppCompatActivity {

    private String mInfix;
    private TextView mCalculatorDisplay;
    private TextView mOutputDisplay;
    final String ops = "-+÷x^√";
    private int mPosition;
    final String trigOps = "sctzef";
    private int counter;
    private double roundValue;
    private boolean hapticFeedback;
    private Vibrator v;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.calc_main);

        // Initialise toolbar
        Toolbar toolbar = findViewById(R.id.my_toolbar);
        setSupportActionBar(toolbar);

        // reset the display text sizes onCreate().
        mCalculatorDisplay = findViewById(R.id.calculatorDisplay);
        mCalculatorDisplay.setTextSize(getResources().getDimension(R.dimen.regular));
        mOutputDisplay = findViewById(R.id.outputDisplay);
        mOutputDisplay.setTextSize(getResources().getDimension(R.dimen.regular));

        // set HTML markup on buttons so the superscript and appears correctly.
        Button xyPowerButton = findViewById(R.id.xyPower);
        Button squarePowerButton = findViewById(R.id.squarePower);
        Button xyRootButton = findViewById(R.id.xyRoot);

        xyPowerButton.setText(Html.fromHtml(getString(R.string.xyPower)));
        squarePowerButton.setText(Html.fromHtml(getString(R.string.squarePower)));
        xyRootButton.setText(Html.fromHtml(getString(R.string.xyRoot)));

        Button arcsinButton = findViewById(R.id.sin_inverse);
```

```
        Button arccosButton = findViewById(R.id.cos_inverse);
        Button arctanButton = findViewById(R.id.tan_inverse);

        arcsinButton.setText(Html.fromHtml(getString(R.string.sin_inverse)));
        arccosButton.setText(Html.fromHtml(getString(R.string.cos_inverse)));
        arctanButton.setText(Html.fromHtml(getString(R.string.tan_inverse)));

        // reset the mInfix onCreate().
        mInfix = "";
        counter = 0;

        // gets SharedPreferences
        SharedPreferences mPrefs = this.getSharedPreferences("calculator",
Context.MODE_PRIVATE);

        if (mPrefs.contains("decimalPoints")) {

            // gets user specified rounding value
            int decimalPoints = mPrefs.getInt("decimalPoints", -1);

            StringBuilder roundValueString = new StringBuilder("10");

            // creates string with zeroes corresponding to the user's value
            for (int i = 0; i < decimalPoints; i++) {

                roundValueString.append("0");
            }

            //  converts string to a double
            roundValue = Double.valueOf(roundValueString.append(".0").toString());

        } else {

            // default rounding rounds to 10 decimal points
            roundValue = 10000000000.0;
        }


        // orientation is locked depending on user's choice. Defaults to portrait if no choice
is
        // made in Settings class (as true corresponds to portrait)
        if (mPrefs.getBoolean("isPortrait", true)) {

            setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
        } else {

            setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
        }

        v = (Vibrator) this.getSystemService(Context.VIBRATOR_SERVICE);
        hapticFeedback = mPrefs.getBoolean("feedbackIsChecked", true);
    }

    public void inputAC(View view) {

        vibratePhone();

        // reset all displays and variables
        mInfix = "";
        mPosition = 0;
        mCalculatorDisplay.setText("");
        mOutputDisplay.setText("");
        counter = 0;
    }

    public void inputDigit(View view) {

        vibratePhone();

        // insert input into correct mPosition
        String input = (String) view.getTag();
```

```
        StringBuilder string = new StringBuilder(mInfix);
        string.insert(mPosition, input);
        mInfix = string.toString();

        // increment mPosition variable
        mPosition++;

        // display to user
        updateDisplay();
    }

    public void inputTrigonometry(View view) {

        vibratePhone();

        // get operator from view's tag
        String input = (String) view.getTag();

        // add buffer spaces to input
        StringBuilder stringInput = new StringBuilder(input);
        stringInput.
                insert(stringInput.length(), " ")
                .insert(0, "0 ");

        input = stringInput.toString();


        // insert input into infix String
        StringBuilder string = new StringBuilder(mInfix);
        string.insert(mPosition, input);
        mInfix = string.toString();

        // update position
        mPosition = mPosition + 4;

        // update display
        updateDisplay();
    }

    public void inputOperator(View view) {

        vibratePhone();

        // get input operator from the view parameter's Tag (XML characteristic)
        String input = (String) view.getTag();

        // add buffer spaces
        StringBuilder stringInput = new StringBuilder(input);
        stringInput.insert(0, " ")
                .insert(stringInput.length(), " ");
        input = stringInput.toString();

        // insert input into correct mPosition
        StringBuilder string = new StringBuilder(mInfix);
        string.insert(mPosition, input);
        mInfix = string.toString();

        // adds +3 for the character and 2 whitespaces
        mPosition = mPosition + 3;

        // display to user
        updateDisplay();
    }

    public void inputOpShortcut(View view) {

        vibratePhone();

        StringBuilder stringBuilderInfix = new StringBuilder(mInfix);
```

```
        // if user taps square power button
        if (view.getId() == R.id.squarePower) {

            // requires a whitespace at the start as it will always immediately follow a digit
            stringBuilderInfix.insert(mPosition, " ^ 2");

        } else { // if user taps square root button

            // requires a trailing whitespace as a number or bracket will immediately follow
            stringBuilderInfix.insert(mPosition, "2 √ ");
        }

        // update position
        mPosition = mPosition + 4;

        // set member infix variable to stringBuilder
        mInfix = stringBuilderInfix.toString();

        // update GUI
        updateDisplay();
    }

    public void inputBracket(View view) {

        vibratePhone();

        // get input from the view's tag
        String input = (String) view.getTag();

        // add buffer spaces to input
        StringBuilder stringBuilderInput = new StringBuilder(input);
        stringBuilderInput.insert(0, " ")
                          .append(" ");
        input = stringBuilderInput.toString();

        // insert input into infix
        StringBuilder stringBuilderInfix = new StringBuilder(mInfix);
        stringBuilderInfix.insert(mPosition, input);
        mInfix = stringBuilderInfix.toString();

        // update position
        mPosition = mPosition + 3;

        validateBracketMultiplication();
        // update GUI to show new infix
        updateDisplay();

    }

    public void validateBracketMultiplication() {

        vibratePhone();

        StringBuilder stringBuilderInfix = new StringBuilder(mInfix);

        // validate length to prevent StringIndexOutOfBounds
        if (mInfix.length() > 4) {

            // check if opening bracket
            if (mInfix.charAt(mPosition - 2) == '('
                    // check if previous token is a digit
                    && Character.isDigit(mInfix.charAt(mPosition - 4))) {

                // insert multiplication token
                stringBuilderInfix.insert(mPosition - 2, "× ");

                // update position variable
                mPosition = mPosition + 2;
            }
        }
```

```
        mInfix = stringBuilderInfix.toString();
    }

    public void delete(View view) {

        vibratePhone();

        StringBuilder stringBuilderInfix = new StringBuilder(mInfix);


        // validate string length to prevent crashes from StringIndexOutOfBounds
        if (stringBuilderInfix.length() > 1) {

            // delete a single character by itself
            if (Character.isDigit(stringBuilderInfix.charAt(mPosition - 1))
                    || ops.contains(String.valueOf(stringBuilderInfix.charAt(mPosition - 1)))
                    || stringBuilderInfix.charAt(mPosition - 1) == '.') {

                // delete character
                stringBuilderInfix.deleteCharAt(mPosition - 1);
                mPosition --;

            // delete single character including whitespace (e.g. when there's an operator)
            } else if (stringBuilderInfix.charAt(mPosition -1) == ' ') {

                // remove token
                stringBuilderInfix.deleteCharAt(mPosition - 1);

                // position must be updated before editing infix to prevent crashes
                mPosition --;

                // remove whitespace
                stringBuilderInfix.deleteCharAt(mPosition - 1);
                mPosition --;
            }


        }

        mInfix = stringBuilderInfix.toString();
        updateDisplay();
    }

    public void submitInfix(View view) {

        vibratePhone();

        // validation check to prevent StringIndexOutOfBounds
        if (mInfix.length() > 0) {

            // try-catch to prevent crashes from badly formed input expressions
            try {


                // get postfix
                String mPostfix = ShuntingYard.infixToPostfix(removePositionMarker(mInfix));

                // get answer
                double mAnswer = ShuntingYard.evaluateRPN(mPostfix);

                mAnswer = Math.round(mAnswer * roundValue) / roundValue;
                // display answer to user
                mOutputDisplay.setText(getString(R.string.answer, String.valueOf(mAnswer)));

            } catch (EmptyStackException e) {
                e.printStackTrace();

                Toast.makeText(this, "ERROR: expression is malformed",
                        Toast.LENGTH_SHORT).show();
            }
        }
```

```
    }

    public void shiftPosition(View view) {

        vibratePhone();

        // user taps right button
        if (view.getId() == R.id.shiftRight) {

            // validation to prevent StringIndexOutOfBoundsException
            if (mPosition < mInfix.length() - 1) {

                // validation check to prevent StringIndexOutOfBoundsException
                if (mInfix.length() - mPosition > 2) {

                    // skips whitespace
                    if (mInfix.charAt(mPosition + 2) == ' ') {

                        // increments position variable by 2 to cover the whitespace
                        mPosition = mPosition + 2;

                        // skips two whitespaces
                    } else if(mInfix.charAt(mPosition + 1) == ' ') {

                        // increment position by 3 when cursor is to the right of a digit to
cover
                        // two whitespaces
                        mPosition = mPosition + 3;

                    } else if (Character.isDigit(mInfix.charAt(mPosition + 1))
                            || trigOps.contains(String.valueOf(mInfix.charAt(mPosition + 1)))) {

                        mPosition = mPosition + 1;
                    }

                } else {

                    // increments position variable
                    mPosition++;
                }
            }

        } else { // user shifts left

            // validation check to prevent StringIndexOutOfBoundsException
            if (mPosition > 0) {

                // skips whitespaces
                if (mInfix.charAt(mPosition - 1) == ' ') {

                    // decreases position variable by 2 to cover the whitespace
                    mPosition = mPosition - 3;

                } else {

                    // decreases position variable
                    mPosition--;
                }
            }
        }

        // update display
        updateDisplay();

    }

    public void updateDisplay() {

        // remove the position marker
        String cleanExpression = removePositionMarker(mInfix);
```

```
        // add underscore as a position marker to GUI
        StringBuilder underscoreString = new StringBuilder(cleanExpression);
        underscoreString.insert(mPosition, "_");
        mInfix = underscoreString.toString();

        // convert infix to array so it can be used in a for-each loop
        ArrayList<String> infixList = new ArrayList<>(Arrays.asList(mInfix.split("")));

        // create array of replacement strings
        String[] trigArray = new String[]{"sin", "cos", "tan", "arcsin", "arccos", "arctan"};

        // create new display StringBuilder
        StringBuilder displayInfix = new StringBuilder();

        // iterate over each element in the infix
        for (String character : infixList) {

            // if the current element is a trigonometric operator
            if (trigOps.contains(character)) {

                // if counter is above 0 to prevent strange bug where there was always a "sin"
at
                // the start of the string
                if (counter > 0) {

                    // get index of character in the trigOps string
                    int trigIndex = trigOps.indexOf(character);

                    // remove the place holder "0 "
                    displayInfix.deleteCharAt(displayInfix.length() -1 )
                            .deleteCharAt(displayInfix.length() - 1);
                    // replace operator with string from the array
                    displayInfix.append(trigArray[trigIndex]);
                }

                // iterate counter
                counter ++;

            } else {

                // add character to string as normal
                displayInfix.append(character);
            }
        }

        // reset counter
        counter = 0;

        // update display
        mCalculatorDisplay.setText(displayInfix.toString());
    }

    public void vibratePhone() {
        // if haptic feedback is enabled, vibrate the phone for 500ms
        if(hapticFeedback) {
            v.vibrate(100);
        }
    }

    public String removePositionMarker(String infix) {

        // replace underscore with empty string to remove it
        return infix.replaceAll("_", "");
    }

    public void onClickEquations(View view) {

        vibratePhone();

        // send to Equations activity
        startActivity(new Intent(Calculator.this, Equations.class));
```

```
    }

    public void onClickConversions(View view) {

        vibratePhone();

        // send to Conversion activity
        startActivity(new Intent(Calculator.this, Conversions.class));
    }

    public void settings(View view) {

        vibratePhone();

        // send to Settings activity
        startActivity(new Intent(Calculator.this, Settings.class));
    }
}
```

5.1.2 *ShuntingYard.java*

```
package com.calculatorproject;

import java.util.HashMap;
import java.util.Map;
import java.util.Stack;

/**
 * This class handles everything relevant to the process of getting a value from the user's
 * input expression.
 *
 * The user's input is converted to a postfix in Reverse Polish Notation using the
infixToPostfix
 * method. The postfix can then be used to calculate a result that is returned to the user.
 *
 * @author David Denny
 * */

public class ShuntingYard {

    /**
     * Method that takes the user's input and iterates through it to create a postfix in the
form
     * of Reverse Polish Notation and returns the postfix.
     *
     * @param infix user's input string
     * @return postfix
     * */

    static String infixToPostfix(String infix) {

        // string that represents every operator. Each operator's precedence can be found by
        // dividing the index of the operator by 2
        final String mOps = "-+÷x^√";

        // creates the postfix stringBuilder
        StringBuilder mPostfix = new StringBuilder();

        // Create new stack containing integers
        Stack<Integer> mStack = new Stack<>();

        // iteras through each token in the user's infix
        for (String token : infix.split("\\s")) {
            if (token.isEmpty()) {

                // if there isn't a token, it returns to the for-each loop
                continue;
            }

            // Char variable containing current token
```

```
            char character = token.charAt(0);

            // index of where the token exists in the operator member variable
            int index = mOps.indexOf(character);

            // if the token is an operator (i.e. exists in the mOps string)
            if (index != -1) {

                if (mStack.isEmpty()) {

                    // push index of token onto the stack
                    mStack.push(index);

                } else {

                    while (!mStack.isEmpty()) {

                        // find precedence value of current and previous operators by dividing
                        // the index by two
                        int previousPrecedence = mStack.peek() / 2;
                        int currentPrecedence = index / 2;

                        //if the previous operator is greater than the current operator or is
the
                        // same as long as the current isn't a power
                        if (previousPrecedence > currentPrecedence ||
                                (previousPrecedence == currentPrecedence && character != '^')) {

                            // index of previous operator is popped off the stack and appends
its
                            // corresponding character in the mOps string to the mPostfix string
                            mPostfix.append(mOps.charAt(mStack.pop())).append(' ');

                        } else {

                            // break out of loop if prevPrecedence is not greater
                            break;
                        }
                    }

                    // at the end of the stack, push the current token's index to the stack
                    mStack.push(index);
                }

            } else if (character == '(') {

                // push "-2" onto the stack to represent the starting bracket
                mStack.push(-2);

            } else if (character == ')') {

                // loops over the tokens inside the brackets
                while (mStack.peek() != -2) {

                    // appends all tokens inside the brackets to postfix stringBuilder to ensure
                    // that calculations inside the brackets are done first
                    mPostfix.append(mOps.charAt(mStack.pop())).append(' ');
                }

                // pops the bracket off the stack
                mStack.pop();

            } else {

                // if the token is a digit (not bracket or operator), append it to the mPostfix
                // stringBuilder
                mPostfix.append(token).append(' ');
            }
        }

        while (!mStack.isEmpty()) {
```

```
            // pops off and appends the remaining tokens to the mPostfix stringBuilder
            mPostfix.append(mOps.charAt(mStack.pop())).append(' ');
        }

        // returns the created postfix string which is now in Reverse Polish Notation
        return mPostfix.toString();
    }

    /**
     * Method to take the postfix string and convert it into a numerical result to be returned
the
     * user.
     *
     * @param postfix user's postfix that is used to calculate a result
     * @return numerical value that is the result of the user's input expression
     **/
    static Double evaluateRPN(String postfix) {

        // make a stack containing Doubles
        Stack<Double> tokens = new Stack<>();

        // for-each loop iterating over every token in the postfix (removing whitespaces to
split
        // each token up)
        for (String token : postfix.split(" ")) {

            // finds the corresponding enum to the token
            Sign sign = Sign.find(token);

            // if the .find() function succesfully finds a corresponding enum
            // (i.e. the token is an operator)
            if (sign != null) {

                // calls calcSign() function to apply the current operator to the first two
                // doubles popped off the stack
                calcSign(tokens, sign);

            } else { // i.e. if the token is a number

                // the token is casted to a Double to ensure data types are compatible
                Double doubleToken = new Double(token);

                // push double onto the stack
                tokens.push(doubleToken);

            }
        }

        // pops off the only number remaining in the stack after the loop. This will be the
user's
        // answer and returns it.
        return tokens.pop();
    }


    protected static Stack<Double> calcSign(Stack<Double> tokens, Sign sign) {

        // pushes the result of the sign parameter with the first two tokens popped off the
stack
        tokens.push(sign.apply(tokens.pop(), tokens.pop()));

        // returns the stack where an operator has been used on two numbers
        return tokens;
    }

    public enum Sign {

        ADD("+") {
            public Double apply(Double num1, Double num2) {
```

```
            // adds the two input numbers
            return num2 + num1;
        }
    },
    SUBTRACT("-") {
        public Double apply(Double num1, Double num2) {

            // subtracts the first number from the second number
            return num2 - num1;
        }
    },
    MULTIPLY("×") {
        public Double apply(Double num1, Double num2) {

            // multiplies the input numbers
            return num2 * num1;
        }
    },
    DIVIDE("÷") {
        public Double apply(Double num1, Double num2) {

            // returns the second number divided by the first
            return num2 / num1;
        }
    },
    POWER("^") {
        public Double apply(Double num1, Double num2) {

            // returns the second number raised by the first
            return Math.pow(num2, num1);
        }
    },
    ROOT("√") {
        public Double apply(Double num1, Double num2) {

            // returns the second number rooted by the first
            return Math.pow(num1, 1.0 / num2);
        }
    };

    // operator text constructor
    private final String mOperatorText;

    // sets the corresponding string denoting the operator to the constructor
    Sign(String operatorText) {
        this.mOperatorText = operatorText;
    }

    // abstract class to make the Sign's corresponding subclasses
    public abstract Double apply(Double num1, Double num2);

    // create Map member variable made up of a string and enum
    private static final Map<String, Sign> mMap;

    static {

        // initialise mMap variable as a HashMap
        mMap = new HashMap<>();

        // use a for-each loop to populate HashMap with the enum and it's corresponding
string
        for (Sign sign : Sign.values()) {
            mMap.put(sign.mOperatorText, sign);
        }
    }


    public static Sign find(String sign) {

        // returns the Sign enum if it exists in the HashMap
        return mMap.get(sign);
```

```
        }

    }

}
```

5.1.3 *Equations.java*

```
package com.calculatorproject;

import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.pm.ActivityInfo;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.support.v7.widget.Toolbar;
import android.view.View;

public class Equations extends AppCompatActivity {
    Toolbar mToolbar;
    RecyclerView mRecyclerView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.maths_equations);

        // initialise toolbar
        mToolbar = findViewById(R.id.quadratic_equation_toolbar);
        setSupportActionBar(mToolbar);

        // initialise RecyclerView
        mRecyclerView = findViewById(R.id.maths_equation_recycler);
        EquationsRAdapter adapter = new EquationsRAdapter(this);

        mRecyclerView.setAdapter(adapter);
        mRecyclerView.setLayoutManager(new LinearLayoutManager(this));

        // gets SharedPreferences
        SharedPreferences mPrefs = this.getSharedPreferences("calculator",
Context.MODE_PRIVATE);

        // orientation is locked depending on user's choice. Defaults to portrait if no choice
is
        // made in Settings class (as true corresponds to portrait)
        if (mPrefs.getBoolean("isPortrait", true)) {

            setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
        } else {

            setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
        }
    }

    public void handleBackButton(View view) {

        startActivity(new Intent(Equations.this, Calculator.class));
    }
}
```

5.1.4 *EquationsRAdapter.java*

```
package com.calculatorproject;

import android.content.Context;
import android.content.Intent;
import android.support.annotation.NonNull;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.RelativeLayout;
import android.widget.TextView;

import java.util.ArrayList;

public class EquationsRAdapter extends RecyclerView.Adapter<EquationsRAdapter.ViewHolder> {

    private ArrayList<String> mContentArray;
    private RecyclerView mRecyclerView;
    private View.OnClickListener mOnClickListener = new View.OnClickListener() {
        @Override
        public void onClick(View v) {

            // get position for switch-case statement
            int position = mRecyclerView.getChildLayoutPosition(v);

            switch(position) {

                //Quadratic Equation
                case 0:
                    v.getContext().startActivity(
                            new Intent(v.getContext(), QuadraticEquation.class));
                    break;

                // Pythagoras' Theorem
                case 1:

                    v.getContext().startActivity(
                            new Intent(v.getContext(), PythagorasTheorem.class));
                    break;

                // Cosine Rule
                case 2:

                    v.getContext().startActivity(new Intent(v.getContext(), CosineRule.class));
                    break;

                // Sine Rule
                case 3:
                    v.getContext().startActivity(new Intent(v.getContext(), SineRule.class));
                    break;

                // Area of a Triangle
                case 4:
                    v.getContext().startActivity(new Intent(v.getContext(),
AreaTriangle.class));
                    break;
            }

        }
    };

    public static class ViewHolder extends RecyclerView.ViewHolder {

        public RelativeLayout mRow;
        public TextView mContent;

        public ViewHolder(View itemView) {
            super(itemView);

            // set member variables
            mRow = itemView.findViewById(R.id.default_recycler_row);
```

```
            mContent = itemView.findViewById(R.id.default_recycler_content);
        }
    }

    public EquationsRAdapter(Context context) {

        mContentArray = new ArrayList<>();

        // add content to array to be later used to populate the RecyclerView
        mContentArray.add("Quadratic Equation");
        mContentArray.add("Pythagoras' Theorem");
        mContentArray.add("Cosine Rule");
        mContentArray.add("Sine Rule");
        mContentArray.add("Area of a Triangle");
    }

    @NonNull
    @Override
    public EquationsRAdapter.ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int
viewType) {

        // inflates each row of the RecyclerView
        LayoutInflater inflater = LayoutInflater.from(parent.getContext());
        View view = inflater.inflate(R.layout.recycler_default_row, parent, false);

        // sets the onClickListener on each row
        view.setOnClickListener(mOnClickListener);
        return new ViewHolder(view);
    }

    @Override
    public void onBindViewHolder(@NonNull EquationsRAdapter.ViewHolder holder, int position) {

        // set text for each element of the RecyclerView
        TextView contentTextView = holder.mContent;
        contentTextView.setText(mContentArray.get(position));
    }

    @Override
    public int getItemCount() {

        // returns size of RecyclerView
        return mContentArray.size();
    }

    @Override
    public void onAttachedToRecyclerView(@NonNull RecyclerView recyclerView) {
        super.onAttachedToRecyclerView(recyclerView);

        // set member variable to RecyclerView so it can be used to find the position
        mRecyclerView = recyclerView;
    }
}
```

5.1.5 *QuadraticEquation.java*

```
package com.calculatorproject;

import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.pm.ActivityInfo;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.text.Editable;
import android.text.Html;
import android.text.TextWatcher;
import android.view.View;
import android.view.inputmethod.InputMethodManager;
import android.widget.EditText;
import android.widget.TextView;
```

```
import android.widget.Toast;

import io.github.kexanie.library.MathView;

public class QuadraticEquation extends AppCompatActivity {
    MathView quadraticDisplay;
    String aString;
    String bString;
    String cString;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.quadratic_equation);

        // set default quadratic equation
        aString = "a";
        bString = "b";
        cString = "c";

        // initialise display TextView
        quadraticDisplay = findViewById(R.id.quadratic_display);

        // Use string substitution to get text to be displayed and set it to the textview
        updateDisplay();

        // get EditText
        EditText aInput = findViewById(R.id.a_input);
        EditText bInput = findViewById(R.id.b_input);
        EditText cInput = findViewById(R.id.c_input);

        // Set TextWatcher to update display when user inputs into "a" EditText
        TextWatcher aTextWatcher = new TextWatcher() {
            @Override
            public void beforeTextChanged(CharSequence s, int start, int count, int after) {

            }

            @Override
            public void onTextChanged(CharSequence s, int start, int before, int count) {

            }

            @Override
            public void afterTextChanged(Editable s) {
                aString = s.toString();
                updateDisplay();

            }
        };

        // Set TextWatcher to update display when user inputs into "b" EditText
        TextWatcher bTextWatcher = new TextWatcher() {
            @Override
            public void beforeTextChanged(CharSequence s, int start, int count, int after) {

            }

            @Override
            public void onTextChanged(CharSequence s, int start, int before, int count) {

            }

            @Override
            public void afterTextChanged(Editable s) {
                bString = s.toString();
                updateDisplay();
            }
        };

        // Set TextWatcher to update display when user inputs into "c" EditText
```

```
        TextWatcher cTextWatcher = new TextWatcher() {
            @Override
            public void beforeTextChanged(CharSequence s, int start, int count, int after) {

            }

            @Override
            public void onTextChanged(CharSequence s, int start, int before, int count) {

            }

            @Override
            public void afterTextChanged(Editable s) {
                cString = s.toString();
                updateDisplay();
            }
        };

        // add TextWatchers to the EditTexts
        aInput.addTextChangedListener(aTextWatcher);
        bInput.addTextChangedListener(bTextWatcher);
        cInput.addTextChangedListener(cTextWatcher);

        // gets SharedPreferences
        SharedPreferences mPrefs = this.getSharedPreferences("calculator",
Context.MODE_PRIVATE);

        // orientation is locked depending on user's choice. Defaults to portrait if no choice
is
        // made in Settings class (as true corresponds to portrait)
        if (mPrefs.getBoolean("isPortrait", true)) {

            setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
        } else {

            setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
        }
    }

    public void calculateQuadratic(View view) {


        // hide keyboard on button click
        InputMethodManager inputManager = (InputMethodManager)
                getSystemService(Context.INPUT_METHOD_SERVICE);
        assert inputManager != null;
        inputManager.hideSoftInputFromWindow(getCurrentFocus().getWindowToken(),
                InputMethodManager.HIDE_NOT_ALWAYS);

        // initialise output TextViews
        TextView textOutput = findViewById(R.id.quadratic_output_text);
        TextView rootOutput = findViewById(R.id.quadratic_output_roots);
        TextView discriminantOutput = findViewById(R.id.quadratic_output_discriminant);

        // validation to prevent crashes from NumberFormatException caused by the user not
        // inputting all the required values
        try {

            // initialise variable values
            double a = Double.parseDouble(aString);
            double b = Double.parseDouble(bString);
            double c = Double.parseDouble(cString);

            // calculate the discriminant
            double discriminant = b * b - 4 * a * c;

            // two real roots
            if (discriminant > 0) {

                // calculate roots
                double firstRoot = (-b + Math.sqrt(discriminant)) / (2 * a);
```

```
                double secondRoot = (-b - Math.sqrt(discriminant)) / (2 * a);

                // display roots
                textOutput.setText(R.string.quadraticTextMultipleRoots);

rootOutput.setText(Html.fromHtml(getString(R.string.quadraticOutputMultipleRoots,
                        String.valueOf(firstRoot), String.valueOf(secondRoot)))
                );

                // repeated real roots
            } else if (discriminant == 0) {

                // calculate root
                double root = (-b + Math.sqrt(discriminant)) / (2 * a);

                // display root
                textOutput.setText(R.string.quadraticRepeatedRootsText);
                rootOutput.setText(getString(R.string.quadraticOutputRepeatedRoots,
String.valueOf(root)));

                // no real roots
            } else {
                // display lack of roots to the user.
                textOutput.setText(R.string.quadraticOutputNoRealRoots);
            }

            // display the discriminant
            discriminantOutput.setText(
                    getString(R.string.quadraticOutputDiscriminant,
String.valueOf(discriminant)));
        } catch (NumberFormatException e) {

            // inform user of error
            Toast.makeText(this, "ERROR: input all required values", Toast.LENGTH_SHORT).show();
        }
    }

    public void updateDisplay() {

        // create TeX code
        String quadraticEquation = String.format(
                "$$\\color{white}{x = \\frac{- %2$s \\pm \\sqrt{%2$s^2 - 4 \\times %1$s \\times
%3$s}}{2 \\times %1$s}}$$",
                aString, bString, cString);

        // render and display TeX code
        quadraticDisplay.setText(quadraticEquation);

    }

    // send user to Equations class onclick
    public void handleBackButton(View view) {
        startActivity(new Intent(QuadraticEquation.this, Equations.class));
    }

}
```

5.1.6 *PythagorasThereom.java*

```
package com.calculatorproject;

import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.pm.ActivityInfo;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.text.Editable;
import android.text.TextWatcher;
```

```
import android.view.View;
import android.view.inputmethod.InputMethodManager;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import io.github.kexanie.library.MathView;

public class PythagorasTheorem extends AppCompatActivity {

    MathView pythagorasDisplay;
    String aString;
    String bString;
    String cString;
    EditText aInput;
    EditText bInput;
    EditText cInput;
    TextView pythagorasTextOutput;
    TextView pythagorasAnswerOutput;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.pythagoras_theorem);

        // set default quadratic equation
        aString = "a";
        bString = "b";
        cString = "c";

        // initialise display TextView
        pythagorasDisplay = findViewById(R.id.pythagoras_display);

        // find and set display text
        updateDisplay();

        // initialise output TextViews
        pythagorasTextOutput = findViewById(R.id.pythagoras_output_text);
        pythagorasAnswerOutput = findViewById(R.id.pythagoras_output_answer);

        // initialise input EditTexts
        aInput = findViewById(R.id.pythagorasAInput);
        bInput = findViewById(R.id.pythagorasBInput);
        cInput = findViewById(R.id.pythagorasCInput);

        // set TextWatcher to update display after user input
        TextWatcher aTextWatcher = new TextWatcher() {
            @Override
            public void beforeTextChanged(CharSequence s, int start, int count, int after) {

            }

            @Override
            public void onTextChanged(CharSequence s, int start, int before, int count) {

            }

            @Override
            public void afterTextChanged(Editable s) {
                // get input
                aString = s.toString();

                // display new input to user
                updateDisplay();

            }
        };

        TextWatcher bTextWatcher = new TextWatcher() {
            @Override
            public void beforeTextChanged(CharSequence s, int start, int count, int after) {
```

```
            }

            @Override
            public void onTextChanged(CharSequence s, int start, int before, int count) {

            }

            @Override
            public void afterTextChanged(Editable s) {
                // get input
                bString = s.toString();

                // apply new input
                updateDisplay();

            }
        };

        TextWatcher cTextWatcher = new TextWatcher() {
            @Override
            public void beforeTextChanged(CharSequence s, int start, int count, int after) {

            }

            @Override
            public void onTextChanged(CharSequence s, int start, int before, int count) {

            }

            @Override
            public void afterTextChanged(Editable s) {
                //get input
                cString = s.toString();

                // apply input
                updateDisplay();

            }
        };

        // add listeners to each EditText
        aInput.addTextChangedListener(aTextWatcher);
        bInput.addTextChangedListener(bTextWatcher);
        cInput.addTextChangedListener(cTextWatcher);

        // gets SharedPreferences
        SharedPreferences mPrefs = this.getSharedPreferences("calculator",
Context.MODE_PRIVATE);

        // orientation is locked depending on user's choice. Defaults to portrait if no choice
is
        // made in Settings class (as true corresponds to portrait)
        if (mPrefs.getBoolean("isPortrait", true)) {

            setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
        } else {

            setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
        }
    }

    public void calculatePythagoras(View view) {

        // hide keyboard on button click
        InputMethodManager inputManager = (InputMethodManager)
                getSystemService(Context.INPUT_METHOD_SERVICE);
        assert inputManager != null;
        inputManager.hideSoftInputFromWindow(getCurrentFocus().getWindowToken(),
                InputMethodManager.HIDE_NOT_ALWAYS);
```

```
        // prevents crashes from NumberFormatException when the user hasn't input the required
        // values
        try {

            // "c" is left empty, so calculate hypotenuse
            if (cInput.getText().toString().equals("")) {

                // get values of triangle sides
                double a = Double.parseDouble(aString);
                double b = Double.parseDouble(bString);

                // calculate the hypotenuse
                double c = Math.sqrt((a * a) + (b * b));

                // display answer
                pythagorasTextOutput.setText(R.string.pythagoras_text_hypotenuse);
                pythagorasAnswerOutput.setText(String.valueOf(c));

                // "a" XOR "b" is empty so calculate the side. Exclusive Or is used here to make
sure
                // at least one side is inputted
            } else if (aInput.getText().toString().equals("") ^
bInput.getText().toString().equals("")) {

                // calculate a
                if (aInput.getText().toString().equals("")) {

                    // get side and hypotenuse values
                    double b = Double.parseDouble(bString);
                    double c = Double.parseDouble(cString);

                    // only continue if hypotenuse is greater than the side
                    if (c > b) {

                        // calculate side a
                        double a = Math.sqrt((c * c) - (b * b));

                        // display answer
                        pythagorasTextOutput.setText(R.string.pythagoras_text_a);
                        pythagorasAnswerOutput.setText(String.valueOf(a));
                    } else {

                        Toast.makeText(this,
                                "ERROR: a side cannot be larger than the hypotenuse.",
                                Toast.LENGTH_SHORT).show();
                    }

                    // calculate b
                } else {

                    // get side and hypotenuse values
                    double a = Double.parseDouble(aString);
                    double c = Double.parseDouble(cString);

                    // only continue if hypotenuse is greater than the side
                    if (c > a) {

                        // calculate side b
                        double b = Math.sqrt((c * c) - (a * a));

                        // display answer
                        pythagorasTextOutput.setText(R.string.pythagoras_text_b);
                        pythagorasAnswerOutput.setText(String.valueOf(b));
                    } else {

                        Toast.makeText(this,
                                "ERROR: a side cannot be larger than the hypotenuse.",
                                Toast.LENGTH_SHORT).show();
                    }
                }
            } else {
```

```
                Toast.makeText(this,

                        "ERROR: make sure you leave one variable blank",
Toast.LENGTH_SHORT).show();
            }

        } catch (NumberFormatException e) {

            // inform user of error
            Toast.makeText(this,
                    "ERROR: input all required values", Toast.LENGTH_SHORT).show();
        }
    }

    // update user's input in display expression
    public void updateDisplay() {

        // create Spanned containing new input
        String pythagorasExpression = String.format(
                "$$\\color{white}{%1$s^2 + %2$s^2 = %3$s^2}$$",
                aString, bString, cString);

        // display new input
        pythagorasDisplay.setText(pythagorasExpression);
    }

    // send user back to MathsEquation
    public void handleBackButton(View view) {

        startActivity(new Intent(PythagorasTheorem.this, Equations.class));
    }
}
```

5.1.7 *CosineRule.java*

```
package com.calculatorproject;

import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.pm.ActivityInfo;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.text.Editable;
import android.text.TextWatcher;
import android.view.View;
import android.view.inputmethod.InputMethodManager;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import io.github.kexanie.library.MathView;

public class CosineRule extends AppCompatActivity {

    MathView cosineDisplay;
    String aString;
    String bString;
    String cString;
    String angleString;
    EditText aInput;
    EditText bInput;
    EditText cInput;
    EditText angleInput;
    TextView cosineTextOutput;
    TextView cosineAnswerOutput;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
super.onCreate(savedInstanceState);
setContentView(R.layout.cosine_rule);

// set generic cosine equation values
aString = "a";
bString = "b";
cString = "c";
angleString = "A";

// initialise display MathView
cosineDisplay = findViewById(R.id.cosine_display);

// find and set display text
setDisplayExpression();

// initialise output TextViews
cosineTextOutput = findViewById(R.id.cosine_output_text);
cosineAnswerOutput = findViewById(R.id.cosine_output_answer);

// initialise input EditTexts
aInput = findViewById(R.id.cosine_a_input);
bInput = findViewById(R.id.cosine_b_input);
cInput = findViewById(R.id.cosine_c_input);
angleInput = findViewById(R.id.cosine_angle_input);

// set TextWatchers to update display after user input
TextWatcher aTextWatcher = new TextWatcher() {
    @Override
    public void beforeTextChanged(CharSequence s, int start, int count, int after) {}

    @Override
    public void onTextChanged(CharSequence s, int start, int before, int count) {}

    @Override
    public void afterTextChanged(Editable s) {
        // get input
        aString = s.toString();

        // display new input to user
        setDisplayExpression();
    }
};

TextWatcher bTextWatcher = new TextWatcher() {
    @Override
    public void beforeTextChanged(CharSequence s, int start, int count, int after) {}

    @Override
    public void onTextChanged(CharSequence s, int start, int before, int count) {}

    @Override
    public void afterTextChanged(Editable s) {
        // get input
        bString = s.toString();

        // display new input to user
        setDisplayExpression();
    }
};

TextWatcher cTextWatcher = new TextWatcher() {
    @Override
    public void beforeTextChanged(CharSequence s, int start, int count, int after) {}

    @Override
    public void onTextChanged(CharSequence s, int start, int before, int count) {}

    @Override
    public void afterTextChanged(Editable s) {
        // get input
        cString = s.toString();
```

```
                // display new input to user
                setDisplayExpression();
            }
        };

        TextWatcher angleTextWatcher = new TextWatcher() {
            @Override
            public void beforeTextChanged(CharSequence s, int start, int count, int after) {}

            @Override
            public void onTextChanged(CharSequence s, int start, int before, int count) {}

            @Override
            public void afterTextChanged(Editable s) {
                // get input
                angleString = s.toString();

                // display new input to user
                setDisplayExpression();
            }
        };

        aInput.addTextChangedListener(aTextWatcher);
        bInput.addTextChangedListener(bTextWatcher);
        cInput.addTextChangedListener(cTextWatcher);
        angleInput.addTextChangedListener(angleTextWatcher);

        // gets SharedPreferences
        SharedPreferences mPrefs = this.getSharedPreferences("calculator",
Context.MODE_PRIVATE);

        // orientation is locked depending on user's choice. Defaults to portrait if no choice
is
        // made in Settings class (as true corresponds to portrait)
        if (mPrefs.getBoolean("isPortrait", true)) {

            setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
        } else {

            setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
        }
    }

    public void setDisplayExpression() {

        // get the String to be used with string substitution string substitution
        String cosineRule = String.format(
                "$$\\color{white}{%1$s^2 = %2$s^2 + %3$s^2 \\times %2$s \\times %3$s \\times
cos(%4$s)}$$",
                aString, bString, cString, angleString);


        // set TextView to user's new expression
        cosineDisplay.setText(cosineRule);
    }

    public void calculateCosineRule(View view) {

        // hide keyboard on button click
        InputMethodManager inputManager = (InputMethodManager)
                getSystemService(Context.INPUT_METHOD_SERVICE);
        assert inputManager != null;
        inputManager.hideSoftInputFromWindow(getCurrentFocus().getWindowToken(),
                InputMethodManager.HIDE_NOT_ALWAYS);

        try {


            // user wants to calculate "a"
            if (aInput.getText().toString().equals("")) {
```

```
                // get sides and angle values
                double b = Double.parseDouble(bString);
                double c = Double.parseDouble(cString);
                double angle = Double.parseDouble(angleString);

                // convert angle to radians as Math.cos() interprets the input as radians
                angle = Math.toRadians(angle);

                // calculate the side a
                double a = Math.sqrt((b * b) + (c * c) * 2.0 * b * c * Math.cos(angle));

                // display answer to user
                cosineTextOutput.setText(R.string.cosine_text_side);
                cosineAnswerOutput.setText(String.valueOf(a));

                // user wants to calculate the angle
            } else if (angleInput.getText().toString().equals("")) {

                double a = Double.parseDouble(aString);
                double b = Double.parseDouble(bString);
                double c = Double.parseDouble(cString);

                double angle = Math.toDegrees(Math.acos(((a * a) - (b * b) - (c * c)) / (2 * b *
c)));

                cosineTextOutput.setText(R.string.cosine_text_angle);
                cosineAnswerOutput.setText(getString(R.string.cosine_answer_angle,
String.valueOf(angle)));

                // user has made an error inputting their values
            } else {
                Toast.makeText(this,
                        "ERROR: your inputs are not valid", Toast.LENGTH_SHORT).show();
            }
        } catch (NumberFormatException e) {

            Toast.makeText(this,
                    "ERROR: input all required values", Toast.LENGTH_SHORT).show();
        }
    }

    public void handleBackButton(View view) {

        startActivity(new Intent(CosineRule.this, Equations.class));
    }
}
```

5.1.8 *SineRule.java*

```
package com.calculatorproject;

import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.pm.ActivityInfo;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.text.Editable;
import android.text.TextWatcher;
import android.view.View;
import android.view.inputmethod.InputMethodManager;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import io.github.kexanie.library.MathView;
```

```
public class SineRule extends AppCompatActivity {
    MathView sineDisplay;
    String aString;
    String bString;
    String aAngleString;
    String bAngleString;
    EditText aInput;
    EditText aAngleInput;
    EditText bInput;
    EditText bAngleInput;
    TextView sineTextOutput;
    TextView sineAnswerOutput;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.sine_rule);

        // initialise variables
        aString = "a";
        bString = "b";
        aAngleString = "A";
        bAngleString = "B";

        // get MathView display
        sineDisplay = findViewById(R.id.sine_display);

        // update display
        updateDisplay();

        // initialise EditTexts
        aInput = findViewById(R.id.a_sine_input);
        aAngleInput = findViewById(R.id.a_capital_sine_input);
        bInput = findViewById(R.id.b_sine_input);
        bAngleInput = findViewById(R.id.b_capital_sine_input);

        // initialise output TextViews
        sineTextOutput = findViewById(R.id.sine_output_text);
        sineAnswerOutput = findViewById(R.id.sine_output_answer);

        // set TextWatchers to update display after user input
        TextWatcher aTextWatcher = new TextWatcher() {
            @Override
            public void beforeTextChanged(CharSequence s, int start, int count, int after) {}

            @Override
            public void onTextChanged(CharSequence s, int start, int before, int count) {}

            @Override
            public void afterTextChanged(Editable s) {
                // get input
                aString = s.toString();

                // display new input to user
                updateDisplay();
            }
        };

        TextWatcher aCapitalTextWatcher = new TextWatcher() {
            @Override
            public void beforeTextChanged(CharSequence s, int start, int count, int after) {}

            @Override
            public void onTextChanged(CharSequence s, int start, int before, int count) {}

            @Override
            public void afterTextChanged(Editable s) {
                // get input
                aAngleString = s.toString();

                // display new input to user
```

```
                updateDisplay();
            }
        };

        TextWatcher bTextWatcher = new TextWatcher() {
            @Override
            public void beforeTextChanged(CharSequence s, int start, int count, int after) {}

            @Override
            public void onTextChanged(CharSequence s, int start, int before, int count) {}

            @Override
            public void afterTextChanged(Editable s) {
                // get input
                bString = s.toString();

                // display new input to user
                updateDisplay();
            }
        };

        TextWatcher bCapitalTextWatcher = new TextWatcher() {
            @Override
            public void beforeTextChanged(CharSequence s, int start, int count, int after) {}

            @Override
            public void onTextChanged(CharSequence s, int start, int before, int count) {}

            @Override
            public void afterTextChanged(Editable s) {
                // get input
                bAngleString = s.toString();

                // display new input to user
                updateDisplay();
            }
        };

        // add TextWatchers
        aInput.addTextChangedListener(aTextWatcher);
        aAngleInput.addTextChangedListener(aCapitalTextWatcher);
        bInput.addTextChangedListener(bTextWatcher);
        bAngleInput.addTextChangedListener(bCapitalTextWatcher);

        // gets SharedPreferences
        SharedPreferences mPrefs = this.getSharedPreferences("calculator",
Context.MODE_PRIVATE);

        // orientation is locked depending on user's choice. Defaults to portrait if no choice
is
        // made in Settings class (as true corresponds to portrait)
        if (mPrefs.getBoolean("isPortrait", true)) {

            setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
        } else {

            setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
        }
    }

    public void updateDisplay() {

        // create new TeX code string
        String sineEquation = String.format(
                "$$\\color{white}{\\frac{%1$s}{sin%2$s} = \\frac{%3$s}{sin%4$s}}$$",
                aString, aAngleString, bString, bAngleString);

        // display new equation
        sineDisplay.setText(sineEquation);
    }
```

```
    public void calculateSineRule(View view) {

        // hide keyboard on button click
        InputMethodManager inputManager = (InputMethodManager)
                getSystemService(Context.INPUT_METHOD_SERVICE);
        assert inputManager != null;
        inputManager.hideSoftInputFromWindow(getCurrentFocus().getWindowToken(),
                InputMethodManager.HIDE_NOT_ALWAYS);

        // prevent crashes from NumberFormatExceptions (when user hasn't entered input)
        try {

            // if the user wants to calculate one of the sides
            if (aInput.getText().toString().equals("") ||
bInput.getText().toString().equals("")) {

                // get angle values and convert to radians
                double angleA = Double.parseDouble(aAngleString);
                double angleB = Double.parseDouble(bAngleString);
                angleA = Math.toRadians(angleA);
                angleB = Math.toRadians(angleB);

                // if user wants to calculate side a
                if (aInput.getText().toString().equals("")) {

                    // get side b value
                    double b = Double.parseDouble(bString);

                    // calculate side a using the Sine Rule
                    double a = Math.sin(angleA) * (b / Math.sin(angleB));

                    // display the answer
                    sineTextOutput.setText(R.string.sine_text_a);
                    sineAnswerOutput.setText(String.valueOf(a));

                    // if the user wants to calculate side b
                } else {

                    // get value of side a
                    double a =  Double.parseDouble(aString);

                    // calculate side a using the Sine Rule
                    double b = Math.sin(angleB) * (a / Math.sin(angleA));

                    // display the answer
                    sineTextOutput.setText(R.string.sine_text_b);
                    sineAnswerOutput.setText(String.valueOf(b));
                }

                // if the user wants to calculate an angle
            } else if (aAngleInput.getText().toString().equals("")
                    || bAngleInput.getText().toString().equals("")) {

                // get side variable values
                double a = Double.parseDouble(aString);
                double b = Double.parseDouble(bString);

                // if user wants to calculate angle A
                if (aAngleInput.getText().toString().equals("")) {

                    // get angle B value and convert to radians
                    double bAngle = Double.parseDouble(bAngleString);
                    bAngle = Math.toRadians(bAngle);

                    // calculate angle A using the Sine Rule and convert to degrees
                    double aAngle = Math.toDegrees(Math.asin(a * Math.sin(bAngle) / b));

                    // display result to user
                    sineTextOutput.setText(R.string.sine_text_aAngle);
                    sineAnswerOutput.setText(String.valueOf(aAngle));
```

```
                    // if the user wants to calculate side B
                } else {

                    // get value of angle A and convert to radians
                    double aAngle = Double.parseDouble(aAngleString);
                    aAngle = Math.toRadians(aAngle);

                    // calculate angle B and convert it to degrees
                    double bAngle = Math.toDegrees(Math.asin(b *  (Math.sin(aAngle) / a)));

                    // display result to user
                    sineTextOutput.setText(R.string.sine_text_bAngle);
                    sineAnswerOutput.setText(String.valueOf(bAngle));
                }

                // user has not entered all the required inputs
            } else {
                Toast.makeText(this,
                        "ERROR: input all required values", Toast.LENGTH_SHORT).show();
            }

            // prevent crash and inform user of error
        } catch (NumberFormatException e) {
            Toast.makeText(this,
                    "ERROR: input all required values", Toast.LENGTH_SHORT).show();
        }

    }

    // send user back to Equations
    public void handleBackButton(View view) {
        startActivity(new Intent(SineRule.this, Equations.class));
    }
}
```

5.1.9 *AreaTriangle.java*

```
package com.calculatorproject;

import android.content.Context;
import android.content.Intent;

import android.content.SharedPreferences;
import android.content.pm.ActivityInfo;
import android.os.Bundle;
import android.support.annotation.NonNull;
import android.support.v7.app.AppCompatActivity;
import android.text.Editable;
import android.text.TextWatcher;
import android.view.View;
import android.view.inputmethod.InputMethodManager;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import io.github.kexanie.library.MathView;

public class AreaTriangle extends AppCompatActivity {

    MathView areaTriangleDisplay;
    String aString;
    String bString;
    String cAngleString;
    EditText aInput;
    EditText bInput;
    EditText cAngleInput;
    TextView areaTriangleAnswer;
    TextView areaTriangleText;
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.area_triangle);

    // initialise variables
    aString = "a";
    bString = "b";
    cAngleString = "C";

    // get MathView display
    areaTriangleDisplay = findViewById(R.id.area_triangle_display);

    // update display
    updateDisplay();

    // initialise EditTexts
    aInput = findViewById(R.id.area_triangle_a_input);
    bInput = findViewById(R.id.area_triangle_b_input);
    cAngleInput = findViewById(R.id.area_triangle_c_angle_input);

    // initialise output TextViews
    areaTriangleText = findViewById(R.id.area_triangle_output_text);
    areaTriangleAnswer = findViewById(R.id.area_triangle_output_answer);

    // set TextWatchers to update display after user input
    TextWatcher aTextWatcher = new TextWatcher() {
        @Override
        public void beforeTextChanged(CharSequence s, int start, int count, int after) {}

        @Override
        public void onTextChanged(CharSequence s, int start, int before, int count) {}

        @Override
        public void afterTextChanged(Editable s) {
            // get input
            aString = s.toString();

            // display new input to user
            updateDisplay();
        }
    };

    TextWatcher bTextWatcher = new TextWatcher() {
        @Override
        public void beforeTextChanged(CharSequence s, int start, int count, int after) {}

        @Override
        public void onTextChanged(CharSequence s, int start, int before, int count) {}

        @Override
        public void afterTextChanged(Editable s) {
            // get input
            bString = s.toString();

            // display new input to user
            updateDisplay();
        }
    };

    TextWatcher cAngleTextWatcher = new TextWatcher() {
        @Override
        public void beforeTextChanged(CharSequence s, int start, int count, int after) {}

        @Override
        public void onTextChanged(CharSequence s, int start, int before, int count) {}

        @Override
        public void afterTextChanged(Editable s) {
            // get input
```

```
                cAngleString = s.toString();

                // display new input to user
                updateDisplay();
            }
        };

        // add TextWatchers
        aInput.addTextChangedListener(aTextWatcher);
        bInput.addTextChangedListener(bTextWatcher);
        cAngleInput.addTextChangedListener(cAngleTextWatcher);

        // gets SharedPreferences
        SharedPreferences mPrefs = this.getSharedPreferences("calculator",
Context.MODE_PRIVATE);

        // orientation is locked depending on user's choice. Defaults to portrait if no choice
is
        // made in Settings class (as true corresponds to portrait)
        if (mPrefs.getBoolean("isPortrait", true)) {

            setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
        } else {

            setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
        }
    }

    public void updateDisplay() {

        // create new TeX code string
        String areaTriangle = String.format(
                "$$\\color{white}{\\frac{1}{2} \\times %1$s \\times %2$s \\times sin(%3$s)}$$",
                aString, bString, cAngleString);

        // render new TeX equation
        areaTriangleDisplay.setText(areaTriangle);
    }

    public void calculateAreaOfATriangle(View view) {

        // hide keyboard on button click
        InputMethodManager inputManager = (InputMethodManager)
                getSystemService(Context.INPUT_METHOD_SERVICE);
        assert inputManager != null;
        inputManager.hideSoftInputFromWindow(getCurrentFocus().getWindowToken(),
                InputMethodManager.HIDE_NOT_ALWAYS);

        // prevent crashes from NumberFormatException (when user hasn't entered required input)
        try {

            double a = Double.parseDouble(aString);
            double b = Double.parseDouble(bString);
            double cAngle = Double.parseDouble(cAngleString);
            cAngle = Math.toRadians(cAngle);

            double area = 0.5 * a * b * Math.sin(cAngle);


            areaTriangleText.setText(R.string.area_triangle_text);
            areaTriangleAnswer.setText(String.valueOf(area));

        } catch (NumberFormatException e) {

            Toast.makeText(this,
                    "ERROR: input all required values", Toast.LENGTH_SHORT).show();
        }
    }

    // send user back to Equations
    public void handleBackButton(View view) {
```

```
        startActivity(new Intent(AreaTriangle.this, Equations.class));
    }
}
```

5.1.10 *Conversions.java*

```
package com.calculatorproject;

import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.pm.ActivityInfo;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.view.View;

public class Conversions extends AppCompatActivity {

    RecyclerView mRecyclerView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.conversions);

        // initialise RecyclerView
        mRecyclerView = findViewById(R.id.conversions_recycler);
        ConversionsRAdapter adapter = new ConversionsRAdapter(this);

        mRecyclerView.setAdapter(adapter);
        mRecyclerView.setLayoutManager(new LinearLayoutManager(this));

        // gets SharedPreferences
        SharedPreferences mPrefs = this.getSharedPreferences("calculator",
Context.MODE_PRIVATE);

        // orientation is locked depending on user's choice. Defaults to portrait if no choice
is
        // made in Settings class (as true corresponds to portrait)
        if (mPrefs.getBoolean("isPortrait", true)) {

            setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
        } else {

            setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
        }
    }

    public void handleBackButton(View view) {

        // send user back to main activity
        startActivity(new Intent(Conversions.this, Calculator.class));
    }
}
```

5.1.11 *ConversionsRAdapter.java*

```
package com.calculatorproject;

import android.app.Dialog;
import android.content.Context;
import android.support.annotation.NonNull;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
```

```
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.EditText;
import android.widget.RelativeLayout;
import android.widget.TextView;
import android.widget.Toast;


import java.util.ArrayList;

public class ConversionsRAdapter extends RecyclerView.Adapter<ConversionsRAdapter.ViewHolder> {

    private ArrayList<String> mContentArray;
    private RecyclerView mRecyclerView;
    private Context mContext;
    private View.OnClickListener mOnClickListener = new View.OnClickListener() {

        @Override
        public void onClick(View v) {

            // get position for switch-case statement
            int position = mRecyclerView.getChildLayoutPosition(v);


            switch (position) {

                case 0:

                    // create dialog variable
                    final Dialog dialogDenToBin = new Dialog(v.getContext());

                    // inflate dialog layout
                    dialogDenToBin.setContentView(R.layout.conversion_dialog);

                    // initialise  buttons
                    Button dialogBack = dialogDenToBin.findViewById(R.id.dialog_cancel);
                    Button dialogCalculate = dialogDenToBin.findViewById(R.id.dialog_calculate);

                    // dismiss dialog when user clicks cancel button
                    dialogBack.setOnClickListener(new View.OnClickListener() {
                        @Override
                        public void onClick(View v) {
                            dialogDenToBin.dismiss();
                        }
                    });


                    dialogCalculate.setOnClickListener(new View.OnClickListener() {

                        @Override
                        public void onClick(View v) {

                            try {

                                // initialise variables
                                EditText denaryEditText =
dialogDenToBin.findViewById(R.id.conversion_input_1);
                                EditText binaryEditText =
dialogDenToBin.findViewById(R.id.conversion_input_2);

                                String denary = denaryEditText.getText().toString();
                                String binary = binaryEditText.getText().toString();

                                if (denary.equals("")) {

                                    // convert binary to denary and display to user
                                    denary = String.valueOf(Integer.parseInt(binary, 2));
                                    denaryEditText.setText(denary);
                                } else if (binary.equals("")) {
```

```
                                // convert denary to binary and display to user
                                binary = String.valueOf(Integer.toBinaryString(Integer.
                                        valueOf(denary)));
                                binaryEditText.setText(binary);
                            } else {

                                // alert user to their error
                                Toast.makeText(v.getContext(),
                                        "Leave one entry variable blank.",
                                        Toast.LENGTH_SHORT).show();
                            }

                            // prevent crashes from invalid input
                        } catch (NumberFormatException e) {
                            e.printStackTrace();

                            // inform user of reason of the error
                            Toast.makeText(v.getContext(), "ERROR: invalid input",
                                    Toast.LENGTH_SHORT).show();
                        }
                    }
                });

                // display dialog
                dialogDenToBin.show();
                break;

            case 1:

                // create dialog variable
                final Dialog dialogDenToHex = new Dialog(v.getContext());

                // inflate dialog layout
                dialogDenToHex.setContentView(R.layout.conversion_dialog);

                // initialise  buttons
                Button dialogDenToHexBack = dialogDenToHex.findViewById(R.id.dialog_cancel);
                Button dialogDenToHexCalculate =
dialogDenToHex.findViewById(R.id.dialog_calculate);

                // set different title
                TextView title2 = dialogDenToHex.findViewById(R.id.input_title_2);
                title2.setText(v.getResources().getString(R.string.hexadecimal));

                // dismiss dialog when user clicks cancel button
                dialogDenToHexBack.setOnClickListener(new View.OnClickListener() {
                    @Override
                    public void onClick(View v) {
                        dialogDenToHex.dismiss();
                    }
                });

                dialogDenToHexCalculate.setOnClickListener(new View.OnClickListener() {

                    @Override
                    public void onClick(View v) {

                        try {

                            EditText denaryEditText =
dialogDenToHex.findViewById(R.id.conversion_input_1);
                            EditText hexEditText =
dialogDenToHex.findViewById(R.id.conversion_input_2);

                            String denary = denaryEditText.getText().toString();
                            String hex = hexEditText.getText().toString();

                            if (denary.equals("")) {

                                // convert and display hex to denary
                                denary = String.valueOf(Integer.parseInt(hex, 16));
```

```
                                denaryEditText.setText(denary);

                            } else if (hex.equals("")) {

                                // convert and display denary to hex
                                hex =
String.valueOf(Integer.toHexString(Integer.valueOf(denary)));
                                hexEditText.setText(hex);

                            } else {

                                // display error message to user
                                Toast.makeText(v.getContext(),
                                        "Leave one entry variable blank.",
                                        Toast.LENGTH_SHORT).show();
                            }

                        // prevent crashes from invalid input
                        } catch (NumberFormatException e) {
                            e.printStackTrace();

                            // inform user of reason of the error
                            Toast.makeText(v.getContext(), "ERROR: invalid input",
                                    Toast.LENGTH_SHORT).show();
                        }
                    }
                });

                // display dialog;
                dialogDenToHex.show();
                break;

            case 2:

                // create dialog variable
                final Dialog dialogBinToHex = new Dialog(v.getContext());

                // inflate dialog layout
                dialogBinToHex.setContentView(R.layout.conversion_dialog);

                // initialise  buttons
                Button dialogBinToHexBack = dialogBinToHex.findViewById(R.id.dialog_cancel);
                Button dialogBinToHexCalculate =
dialogBinToHex.findViewById(R.id.dialog_calculate);

                // set different titles
                TextView titleBin = dialogBinToHex.findViewById(R.id.input_title_1);
                TextView titleHex = dialogBinToHex.findViewById(R.id.input_title_2);

                titleBin.setText(v.getResources().getString(R.string.binary));
                titleHex.setText(v.getResources().getString(R.string.hexadecimal));

                // dismiss dialog when user clicks cancel button
                dialogBinToHexBack.setOnClickListener(new View.OnClickListener() {
                    @Override
                    public void onClick(View v) {
                        dialogBinToHex.dismiss();
                    }
                });

                dialogBinToHexCalculate.setOnClickListener(new View.OnClickListener() {
                    @Override
                    public void onClick(View v) {

                        try {


                            EditText binaryEditText =
dialogBinToHex.findViewById(R.id.conversion_input_1);
                            EditText hexEditText =
dialogBinToHex.findViewById(R.id.conversion_input_2);
```

```
                                String binary = binaryEditText.getText().toString();
                                String hex = hexEditText.getText().toString();

                                if (binary.equals("")) {

                                    // convert hex to binary and display
                                    binary = Integer.toBinaryString(Integer.parseInt(hex, 16));
                                    binaryEditText.setText(binary);

                                } else if (hex.equals("")) {

                                    // convert and display binary to hex
                                    hex = Integer.toString(Integer
                                            .parseInt(binary, 2), 16);

                                    hexEditText.setText(hex);

                                } else {

                                    // display error message to user
                                    Toast.makeText(v.getContext(),
                                            "Leave one entry variable blank.",
                                            Toast.LENGTH_SHORT).show();
                                }

                                // prevent crashes from invalid input
                            } catch (NumberFormatException e) {
                                e.printStackTrace();

                                // inform user of reason of the error
                                Toast.makeText(v.getContext(), "ERROR: invalid input",
                                        Toast.LENGTH_SHORT).show();
                            }
                        }
                    });

                    // display dialog to user
                    dialogBinToHex.show();
                    break;
                }

            }
        };

        public static class ViewHolder extends RecyclerView.ViewHolder {

            public RelativeLayout mRow;
            public TextView mContent;

            public ViewHolder(View itemView) {
                super(itemView);

                // set member variables
                mRow = itemView.findViewById(R.id.default_recycler_row);
                mContent = itemView.findViewById(R.id.default_recycler_content);
            }
        }

        public ConversionsRAdapter(Context context) {

            mContentArray = new ArrayList<>();

            // add content to array to be later used to populate the RecyclerView
            mContentArray.add("Denary <-> Binary");
            mContentArray.add("Denary <-> Hexadecimal");
            mContentArray.add("Binary <-> Hexadecimal");
        }

        @NonNull
        @Override
```

```
    public ConversionsRAdapter.ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int
viewType) {

        // inflates each row of the RecyclerView
        LayoutInflater inflater = LayoutInflater.from(parent.getContext());
        View view = inflater.inflate(R.layout.recycler_default_row, parent, false);

        // sets the onClickListener on each row
        view.setOnClickListener(mOnClickListener);

        return new ViewHolder(view);
    }

    @Override
    public void onBindViewHolder(@NonNull ConversionsRAdapter.ViewHolder holder, int position) {

        // set text for each element of the RecyclerView
        TextView contentTextView = holder.mContent;
        contentTextView.setText(mContentArray.get(position));
    }

    @Override
    public int getItemCount() {

        // returns size of RecyclerView
        return mContentArray.size();
    }

    @Override
    public void onAttachedToRecyclerView(@NonNull RecyclerView recyclerView) {
        super.onAttachedToRecyclerView(recyclerView);

        // set member variable to RecyclerView so it can be used to find the position
        mRecyclerView = recyclerView;
    }
}
```

5.1.12 *Settings.java*

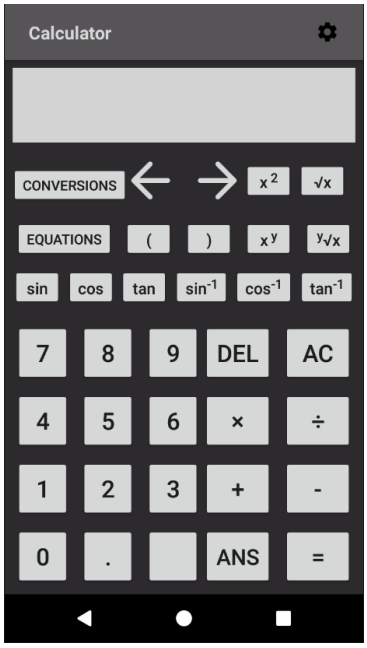
```
package com.calculatorproject;

import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.pm.ActivityInfo;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.SwitchCompat;
import android.view.View;
import android.widget.CompoundButton;
import android.widget.NumberPicker;
import android.widget.TextView;

public class Settings extends AppCompatActivity {

    private SharedPreferences mPrefs;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.settings);

        mPrefs = this.getSharedPreferences("calculator", Context.MODE_PRIVATE);

        NumberPicker decimalPicker = findViewById(R.id.decimal_picker);

        // add values to the NumberPicker
        final String[] pickerValues = {"1", "2", "3", "4", "5", "6", "7", "8", "9", "10"};
```

```
        decimalPicker.setMinValue(0);
        decimalPicker.setMaxValue(pickerValues.length - 1);
        decimalPicker.setDisplayedValues(pickerValues);
        decimalPicker.setWrapSelectorWheel(true);

        // set current value
        decimalPicker.setValue(mPrefs.getInt("decimalPoints", 1));

        decimalPicker.setOnValueChangedListener(new NumberPicker.OnValueChangeListener() {
            @Override
            public void onValueChange(NumberPicker picker, int oldVal, int newVal) {

                decimalPicker.setValue(newVal);

                // update preferences with new value
                mPrefs.edit()
                        .putInt("decimalPoints", newVal)
                        .apply();
            }
        });

        SwitchCompat orientationSwitch = findViewById(R.id.orientation_switch);
        TextView orientationLabel = findViewById(R.id.orientation_label);

        // gets portrait setting
        Boolean isPortrait = mPrefs.getBoolean("isPortrait", true);

        // changes the label to portrait or landscape depending on isPortrait variable
        orientationLabel
                // uses a ternary operator to set the text to either of two values depending on
the
                // value of the boolean conditional. In this case, true corresponds to portrait
                // and false to landscape.
                .setText(isPortrait ? getString(R.string.portrait) :
getString(R.string.landscape));

        // sets switch to corresponding state
        orientationSwitch.setChecked(isPortrait);

        orientationSwitch.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {

                if (isChecked) {

                    // alters label
                    orientationLabel.setText(getString(R.string.portrait));

                    // saves user input in SharedPreferences
                    mPrefs.edit()
                            .putBoolean("isPortrait", true)
                            .apply();

                    // changes orientation
                    setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
                } else {

                    // alters label
                    orientationLabel.setText(getString(R.string.landscape));

                    // saves user input in SharedPreferences
                    mPrefs.edit()
                            .putBoolean("isPortrait", false)
                            .apply();

                    // changes orientation
                    setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
                }
            }
        });
```

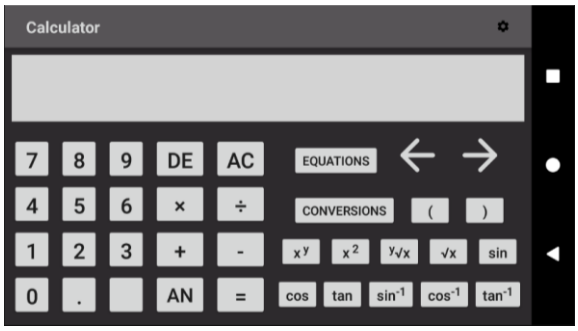
```
        // get views
        SwitchCompat feedbackSwitch = findViewById(R.id.haptic_feedback_switch);
        Boolean feedbackIsChecked = mPrefs.getBoolean("feedbackIsChecked", true);

        // set state of switch from user input. Default is checked.
        feedbackSwitch.setChecked(feedbackIsChecked);

        feedbackSwitch.setOnCheckedChangeListener((buttonView, isChecked) -> {

            mPrefs.edit()
                    .putBoolean("feedbackIsChecked", isChecked)
                    .apply();

        });

        // gets SharedPreferences
        SharedPreferences mPrefs = this.getSharedPreferences("calculator",
Context.MODE_PRIVATE);

        // orientation is locked depending on user's choice. Defaults to portrait if no choice
is
        // made in Settings class (as true corresponds to portrait)
        if (mPrefs.getBoolean("isPortrait", true)) {

            setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
        } else {

            setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
        }
    }


    public void onClickBackButton(View view) {

        // send user back to main calculator activity
        startActivity(new Intent(Settings.this, Calculator.class));
    }
}
```

5.2 – App Design

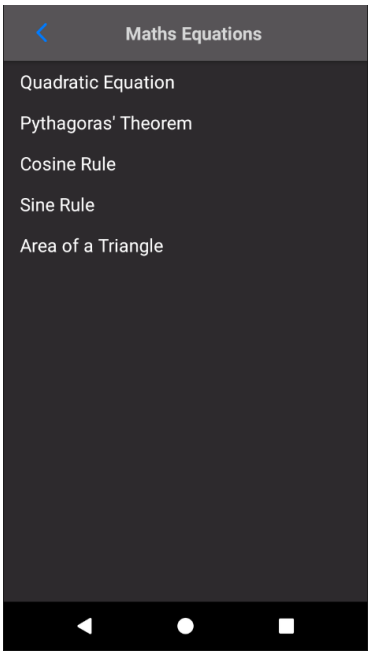
5.2.1.1 *calculator.xml (portrait)*



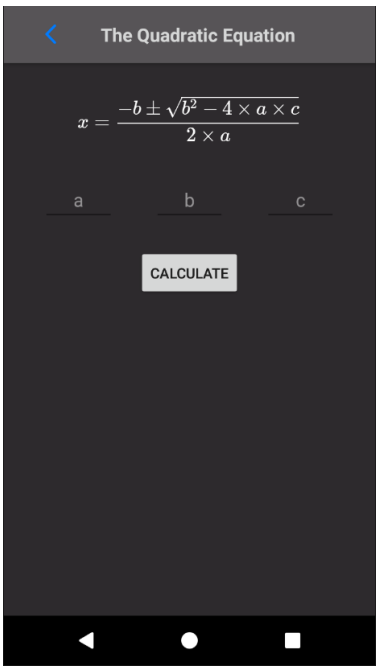
5.2.1.2 *calculator.xml (landscape)*



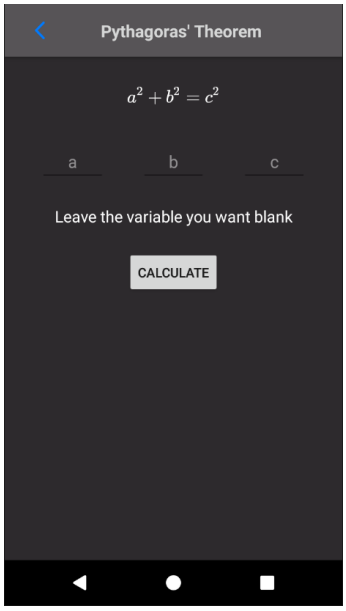
5.2.2 *equations.xml*



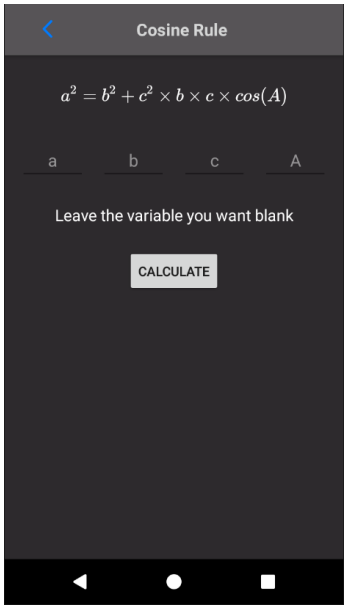
5.2.3 quadratic_equation.xml



5.2.4 pythagoras_theorem.xml

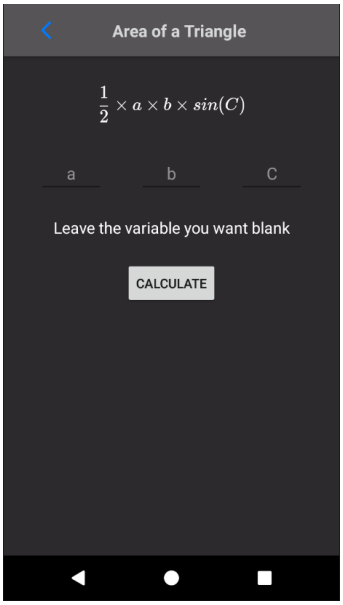


5.2.5 *cosine_rule.xml*

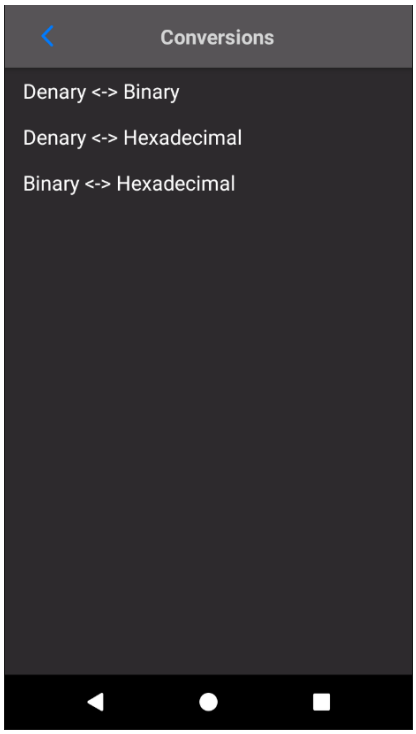


5.2.6 sine_rule.xml

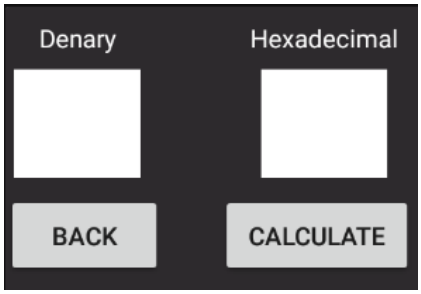
5.2.7 area_triangle.xml



5.2.8 *conversions.xml*



5.2.9 conversions_dialog.xml



5.2.10 settings.xml