Technical Solution

Contents

Technical Solution Overview

The Technical Solution section of the document will include the program code used to complete the objectives that have been set and signed off by my client. It will be split into different segments with each one showing the code used to complete an objective. Beta Testing will be completed in a separate section directly after the technical solution. A clear title with the objective(s) that are being completed will be included for every section of code. These will be shown with screenshots, and the full program code will be written as text within the document at the end. Each piece of code will have a unique reference to make the testing of it clearer. These will be referenced accordingly with numbers (such as 1.3) as I have used numbers with letters (such as 1c) in the testing references. This is to avoid confusion between which aspect of the document I am discussing. The 'Imported Modules' and 'Algorithms Used' subsections were made after I had completed the technical solution to provide an overview of the project's solution.

Imported Modules

I have created a table of all the Python modules that I imported to complete this project. Each module listed has the justification for using it and an example of its use.

Module Name	Justification
Kivy	The Kivy module provides all of the features to create a GUI including its own language (.kv) for design elements. This includes buttons, text inputs, labels, shapes, and a vast range of colours which is needed to for me to meet my objectives, especially objective 1. Kivy itself is an incredibly massive library which is why I only imported a subset of its code to improve the efficiency of the application when it is run.
Turtle	The Turtle module is a graphics module which allows me to make effective and high-quality games, with movement tracking and keyboard binding. This was especially important for objective 4 to create the Maths Invaders Game and will be used in the future if I plan to add more games to the application.
Math	My entire application is based on A-Level Mathematics which means many advanced functions that are not already included with Python will be needed. An example is the Square Root function which I used throughout my Revision Sections for Objective 7 and used it in my Game for Objective 4.
NumPy	For me to successfully complete my revision sections for obj 7, I needed to produce different user-defined graphs which required me to use the NumPy features such as the NumPy array that I used to plot a user-defined triangle on a graph.
MatPlotLib	This module allowed me to display the graphs that were plotted in a clear format. It also provided powerful tools such as zooming and saving the graph as an image. It was vital for objective 7
HashLib	For me to ensure that the accounts and their data are stored securely, I decided to store the hashed versions of the passwords rather than a plaintext version. This improves security as a Hash is usually a one-way cryptographic function and therefore is incredibly hard to crack. This was used for Objective 2. I did not make my own hashing algorithm as this project is to produce a mathematics application and not research into cryptographic hashing and techniques for doing so. If I did create my own, it would severely impact my other objectives, and I wouldn't have been able to meet all the client's objectives as he requested due to the time limitations.

Sqlite3	SQLite3 is an integral part of the entire application as the database is the foundations of the project and each objective. This was used to create, read, update and delete tables and data within the database. This module is directly linked to the completion of objective 5 but helps to meet all the other objectives too.
Random	The random module was used in several parts of the code, most notably in the topic tests for Objective 3. This allowed me to randomly choose a question from a list which was produced based on which topic the user wanted to get tested on. It was also crucial in the Statistics Revision section which maps to objective 7.
Tabulate	The Tabulate module allowed me to produce coherent reports and tables to output to the user based on the database. This was used to take in a list of data which was produced by my functions, and headers. It then tabulated that data for me to output. The Tabulate module was used for Objective 6 in outputting the Progress Tables and for Objective 8 to output the scheduled meetings or the timetables for users.
Datetime	The datetime module was used in Objective 8 to figure out the date of the next scheduled meeting and outputted this to the user. This clarified the scheduled meeting as it meant that along with the day of the week and the period, the student and teacher also knew the exact date of the meeting.
smtplib	The SMTP library allowed me to connect to an SMTP email server and send email notifications which was an additional requirement added into Objective 8 by my client.

Algorithms Used

I have included a table below which is an overview of my some of the algorithms used in the Technical Solution. **It will *not* include all the algorithms/data structures that I have used**. However, all algorithms and data structures that I have used will be shown in the Technical Solution throughout each of the sections with screenshots and corresponding tests. For the Evidence column, the numbers relate to the Code Screenshot: an entry into the evidence column of *3.1* will, therefore, be referring to *Code Screenshot 3.1*. Not all possible evidence references will be included in this table, but it will be clear throughout the Technical Solution section which algorithm is being mentioned using captions below the screenshots.

Section(s)	Data Structure/Algorithm	Justification	Evidence
Database	A relational database with several interlinked tables	The program needs to store data about a user. This involves different types or reasons for the data but needs to be linked, for example to the user table, to fully meet my objectives such as a progress tracking feature which will cover several tables in the database.	Used throughout the program for every objective (excluding objective 1)
	Data Type Checking Functions	This was used in conjunction with the database tables' own constraints to ensure any user input was compatible with the database design, i.e. having an integer or string in a specific field.	1.11
Database and Progress Tracking	SQL Aggregate Scripts	For a complete overview in progress tracking for a class, I needed to find the SUMS or AVG of several records for a given field.	7.8
	Parametrised SQL Scripts with Table Joins	Similar to the above reason, using these powerful SQL scripts allows me to generate extensive reports and tables about students on their progress.	7.7, 7.11, 7.12

GUI	Object-Oriented Programming	For a complete application, especially to fully meet objective 1, my GUI had to be completed fully. To do this, I used OOP with each screen being represented by its own class so that I can define functions which are related to that class, and therefore that screen. This made the structure of the program much easier to work with and overall is more efficient than other options.	Used throughout program
Revision Section – Calculus	Stack and Stack Operations	This was required to allow the user to undo one of their actions on the graph which was requested to make it easier to make comparisons between several graphs. As the students found this useful, it was vital to have in my program as they will be the end users along with teachers such as my client	4.6, 4.10
	Recursive Factorial Function	The factorial function is heavily relied upon in the Maclaurin Expansion which is a lesson in my revision materials, and it allows me to make that area more interactive which is a big part of objective 4 and 7. It was also important in the statistics section for me to produce the probability distribution functions.	4.2, 4.5, 4.19

Revision Section - Calculus	Advanced Maths – Numerical Integration and Differentiation	The core of calculus lies with finding the area under a curve or the gradient at a point. To develop these skills for maths students, I created an interactive revision section for the user-defined graphs. This allows them to practice, with an equation of their choice, to find the area or gradient and then use my functions to check their answer. Both functions use the first principles method to calculate their values. This is an integral part of objectives 4 and 7.	4.9
Revision Section – Statistics	Advanced Maths – Binomial and Normal Distribution	In the Mathematics A-level specification, probability distributions is a large part of the statistics course. Having these functions were vital in supporting students to understand them, with unlimited scope for practice as the user can type their own values to calculate.	4.19, 4.20
Revision Section, Progress Tracking, Scheduler	List Comprehension	This was used for a number of different purposes. However, the core reason for each was the improved efficiency of generating a list with iteration in a single line. It also allowed me to produce complex ranges to remove specific values such as in my Scheduler	4.7, 7.8, 8.10

Maths Invader Game	Merge Sort	One major part of the objective criteria for objective 4 was to have an efficient leaderboard sorting algorithm. I researched further and found that MergeSort had a worst-case time complexity of O(*nlog(n)*). This logarithmic time complexity was much better than the quadratic time complexity of Bubble Sort and Selection Sort which is why I chose to use it.	6.3, 6.4

Relational Database Setup

The first step for my technical solution is to set up my relational database which maps to the completion of ***Objective 5***. This will form the foundations of the application. Each screenshot/piece of code shown in this section will have a reference ***1.x*** where ***x*** is an integer. From the design section, I have made an addition to each of the SQL scripts which is the use of the 'IF NOT EXISTS' condition for each table creation. This ensures there are no duplicate table creations, overwriting of data or errors when the code is run.

```
#Mathematics Database
#Importing sqlite3 module which is how I can manage the database
import sqlite3
#I always need to explicitly connect to the database to register changes#
connection = sqlite3.connect("Mathematics.db")
#Using a cursor in order to fetch results when needed
crsr = connection.cursor()
#Using scripts that were made in the Design Section with a few additions

#Creating User Table
crsr.execute('''CREATE TABLE IF NOT EXISTS User (
    Email VARCHAR(60) NOT NULL UNIQUE,
    FirstName VARCHAR(15) NOT NULL,
    LastName VARCHAR(15) NOT NULL,
    HashedPassword CHAR(64) NOT NULL,
    Student_Status BOOLEAN NOT NULL,
    PRIMARY KEY (Email));
    ''')
#Creating Topic Table
crsr.execute('''CREATE TABLE IF NOT EXISTS Topic (
    QuestionID INT NOT NULL UNIQUE,
    Question VARCHAR(255) NOT NULL,
    Answer VARCHAR(50) NOT NULL,
    Marks INT NOT NULL,
    Difficulty INT NOT NULL,
    PRIMARY KEY (QuestionID));
    ''')
```

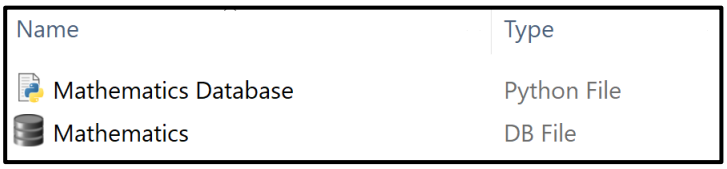
Code Screenshot 1.1 *– Creating a new 'Mathematics' database and connecting to it. Using Python SQLITE3 to create the User and Topic Tables*

```
#Creating Progress Table
crsr.execute('''CREATE TABLE IF NOT EXISTS Progress (
    ProgressID INT NOT NULL UNIQUE,
    DiffCorrect INT NOT NULL DEFAULT 0,
    DiffTotal INT NOT NULL DEFAULT 0,
    IntCorrect INT NOT NULL DEFAULT 0,
    IntTotal INT NOT NULL DEFAULT 0,
    StatCorrect INT NOT NULL DEFAULT 0,
    StatTotal INT NOT NULL DEFAULT 0,
    TrigCorrect INT NOT NULL DEFAULT 0,
    TrigTotal INT NOT NULL DEFAULT 0,
    AvgDifficulty FLOAT NOT NULL DEFAULT 1,
    Email VARCHAR(60),
    PRIMARY KEY(ProgressID),
    FOREIGN KEY(Email) REFERENCES User(Email));
    ''')
#Creating Timetable Table to store when the user is free in a specific school period#
crsr.execute('''CREATE TABLE IF NOT EXISTS TimeTable (
    TimeID Integer NOT NULL,
    DayNumber INT NOT NULL,
    Period_1 BOOLEAN NOT NULL,
    Period_2 BOOLEAN NOT NULL,
    Period_3 BOOLEAN NOT NULL,
    Period_4 BOOLEAN NOT NULL,
    Period_5 BOOLEAN NOT NULL,
    Email VARCHAR(60) NOT NULL,
    FOREIGN KEY (Email) References User(Email),
    PRIMARY KEY (TimeID, DayNumber));
    ''')

#Creating the Scheduled Table
crsr.execute('''CREATE TABLE IF NOT EXISTS Scheduled (
    ScheduleID INT NOT NULL UNIQUE,
    ScheduleDay INT NOT NULL,
    SchedulePeriod INT NOT NULL,
    Reason VARCHAR(255) NOT NULL,
    Email VARCHAR(60) NOT NULL,
    FOREIGN KEY(Email) REFERENCES User(Email),
    PRIMARY KEY (ScheduleID));
    ''')
```

```
#Creating the MathsInvader table to store data related to the game
crsr.execute('''CREATE TABLE IF NOT EXISTS MathsInvader (
    GameID INT NOT NULL UNIQUE,
    Email VARCHAR(60) NOT NULL,
    ProgressID INT NOT NULL,
    GameScore INT NOT NULL DEFAULT 0,
    MathScore INT NOT NULL DEFAULT 0,
    FOREIGN KEY(Email) REFERENCES User(Email),
    FOREIGN KEY(ProgressID) REFERENCES Progress(ProgressID),
    PRIMARY KEY (GameID));
    ''')

#To ensure the code above is executed, I have to use the commit function
connection.commit()
#This closes the connection to register all changes made to the database
connection.close()
```

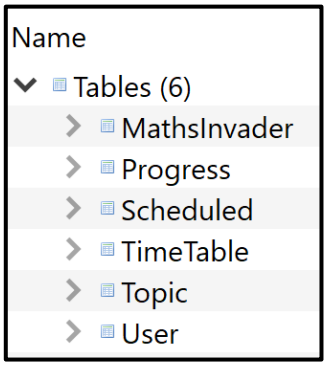
Code Screenshots 1.2 – Program code to complete creating remaining tables of 'Progress', 'Timetable', 'Scheduled' and 'MathsInvader' along with the code to ensure all code has been executed successfully and database has registered updates.

Upon executing the code, I checked if it was successful in firstly creating the database, and then creating the tables as I had defined with the schema above. I checked this using a database browser: software which allows you to check the structure of a database and data that has been inputted into any tables.



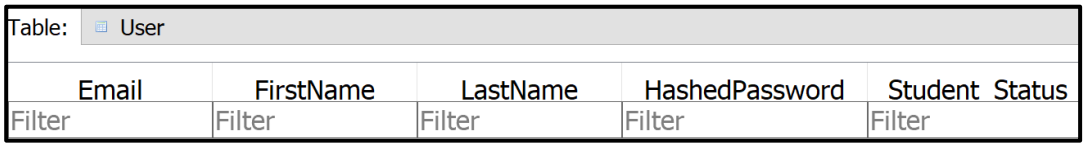
*Code Screenshot **1.3** – Shows the database file (DB file) has been created successfully*

The above screenshot (***1.3***) shows the successful creation of the database which I have named 'Mathematics' as that is what the application is based around. The database file has been created in the same folder as the saved Python file 'Mathematics Database' which contains the lines of code from ***Code Screenshots 1.1*** and ***1.2***.



*Code Screenshot **1.4** – Shows all tables have been created successfully in my Mathematics database*

Code Screenshot 1.4 shows that all coded tables have been created successfully in the database. If any additional table is required for the future, I can easily add it on using the Python module SQLITE3.



*Code Screenshot **1.5** – Shows one of the tables (User) with each field defined successfully from the executed code.*

Finally, ***Code Screenshot 1.5*** shows one of the tables that were created along with the fields. This is the User table. Each table has been successfully created as intended, and they will now undergo alpha testing. This is to ensure the database is ready for any types of data changes and processes that will be run at any point during my overall program. I will use my pre-designed '***Test Table 1 – Database and Tables***' which was made in the Design Section.

Alpha Testing – Database and Tables

The code to test these inputs will follow the same format, and to prevent redundancy, I will show the foundations of the code below. The only changes made will be the 'MyTable' which will be replaced by the Table Name in the test; 'The Columns' which will be replaced by the field names in the 'Input (Field)' section of the test and 'The Values' which will be replaced by the input values described in the test table. A screenshot of the output will be placed in the Actual Output section. The comments section will be colour coded: green means the expected output matches the actual output and red means the expected output does not match the actual output.

```
#Importing sqlite3 module which is how I can manage the database
import sqlite3
#Making connection with database
connection = sqlite3.connect("Mathematics.db")
#Using a cursor in order to fetch results when needed
crsr = connection.cursor()

#General Inputs for Test 1.x #
crsr.execute('''INSERT INTO MyTable (The Columns) VALUES (The Values); ''')

#To ensure the code above is executed, I have to use the commit function
connection.commit()
#This closes the connection to register all changes made to the database
connection.close()
```

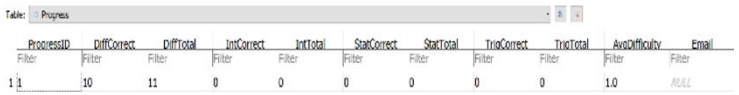
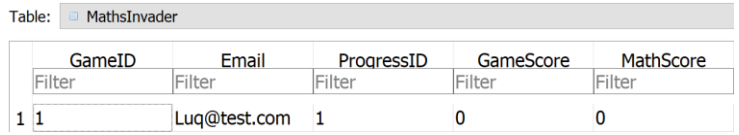
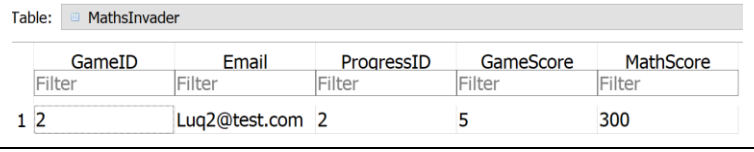
*Code Screenshot **1.6** – Generalised format for Alpha Testing of database and tables.*

Test Ref	Table Name	Input *(Field)*	Expected Output	*Actual Output*	*Comment*
1a	User	**Luqman** *(FirstName)*	The table will reject the input as there are other fields within the table that have been given a NOT NULL constraint.	`crsr.execute('''INSERT INTO User (FirstName) VALUES ('Luqman'); ''')` `sqlite3.IntegrityError: NOT NULL constraint failed: User.Email`	Works as expected
1b	User	**luq@test.com** *(Email)*, **Luqman** *(FirstName)*, **Liaquat** *(LastName)*, **ba7816…** *(HashedPassword)*, **TRUE** *(Student_Status)*	The table should update successfully (when the hashed password is written fully)	Table: User Email: luq@test.com; FirstName: Testing; LastName: Tests; HashedPasswor: ba7816bf8f01…; Student_Status: TRUE	Works as expected

1c	User	**luq@test.com** *(Email)*, **Testing** *(FirstName)*, **Tests** *(LastName)*, **4EFB32F218…** *(HashedPassword)*, **TRUE** *(Student_Status)*	This input should be rejected as the email is repeated from the last test, and therefore breaks the UNIQUE constraint.	`VALUES ('luq@test.com', 'Testing', 'Tests', '123456bf8f01cfea414140de5dae2`
				`3b00361a396177a9cb410ff61f20015a1', 'TRUE'); ''')`
				`sqlite3.IntegrityError: UNIQUE constraint failed: User.Email`
					Works as expected

1d	Topic	**100** *(QuestionID)*, **Differentiate $5x^2$** *(Question)*, **10x** *(Answer)*, **1** *(Marks)*, **1** *(Difficulty)*	This input should be accepted, and the table will be updated successfully	Table: Topic QuestionID: 100; Question: Differentiate 5…; Answer: 10x; Marks: 1; Difficulty: 1	Works as expected
1e	Topic	**200** *(QuestionID)*, **Integrate $6x^2$** *(Question)*, **$2x^3$** *(Answer)*, **One** *(Marks)*, **1** *(Difficulty)*	The *Marks* field has an Integer data type, but a string has been inputted which should cause an error and rejection.	Table: Topic QuestionID: 200; Question: Integrate 6x^2; Answer: 2x^3; Marks: One; Difficulty: 1	Not Working as intended – See amendment section

1f	Timetable	**1** *(TimeID)*, **Luq@test.com** *(Email)*, **5** *(DayNumber)*, **TRUE** *(Period1)*, **TRUE** *(Period2)*, **TRUE** *(Period3)*, **TRUE** *(Period4)*, **FALSE** *(Period5)*	All data entered is expected to cause no errors, and therefore the table will update successfully.	Table: TimeTable TimeID | DayNumber | Period 1 | Period 2 | Period 3 | Period 4 | Period 5 | Email 1 | 1 | 5 | TRUE | TRUE | TRUE | TRUE | FALSE | Luq@test.com	Works as expected
1g	Timetable	**2** *(TimeID)*, **Luq@test.com** *(Email)*, **2** *(DayNumber)*, **Maths** *(Period1)*, **Maths** *(Period2)*, **Physics** *(Period3)*, **Computing** *(Period4)*, **Computing** *(Period5)*	This input will not be accepted as each 'Period' field has a Boolean data type. The inputs use the names of the subjects which are strings and therefore will cause an error.	Table: TimeTable TimeID | DayNumber | Period 1 | Period 2 | Period 3 | Period 4 | Period 5 | Email 1 | 2 | 2 | Maths | Maths | Physics | Computing | Computing | Luq@test.com	Not Working as intended– See amendment section
1h	Scheduled	**1** *(ScheduledID)*, **12** *(ScheduleDay)*, **10** *(SchedulePeriod)*, **Failed** *(Reason)*, **Luq@test.com** *(Email)*	As with a previous table test, there should be no SQL error here but the schedule day and schedule period fields should not have numbers above 5. A new constraint will need to be added.	Table: Scheduled ScheduleID | ScheduleDay | SchedulePeriod | Reason | Email 1 | 1 | 12 | 10 | Failed | Luq@test.com	Working as expected – See amendment section for additions made
1i	Scheduled	**2** *(ScheduledID)*, **3** *(ScheduleDay)*, **1** *(SchedulePeriod)*, **Low Marks** *(Reason)*, **Luq@test.com** *(Email)*	This input should cause no errors which means the table should update succesfully	Table: Scheduled ScheduleID | ScheduleDay | SchedulePeriod | Reason | Email 1 | 2 | 3 | 1 | Low Marks | Luq@test.com	Works as expected

1j	Progress	**1** *(ProgressID)*, **10** *(DiffCorrect)*, ***11*** *(DiffTotal)*	These inputs will update the table successfully. However, a NOT NULL constraint needs to be added to the email field.		Working as expected – See amendment section for additions made
1k	Progress	**2** *(ProgressID)*, **Luq2@test.com** *(Email)*	This should update the table without any errors, as I have included default values for the remaining fields.		Works as expected
1l	MathsInvader	**1** *(GameID)*, **Luq@test.com** *(Email)*, **1** *(ProgressID)*	This will create the new record into the table with the fields not mentioned defaulting to 0.		Works as expected
1m	MathsInvader	**2** *(GameID)*, **Luq2@test.com** *(Email)* **2** *(ProgressID)*, **5** *(GameScore)*, **300** *(MathScore)*	This set of inputs should result in no errors.		Works as expected

Amendments

After further research on some of the unexpected outcomes from the above tests, I have found that the SQLITE3 data types are slightly different from the MySQL data types which I am used to. The container of an item of data does not have a data type attached to it, but, if specified, the item of data itself has a data type that is associated with it.[9] Therefore, any type of data will be able to used inside a field even if it's not the right data type, unless a Check constraint is used.[10] The repeated tests are shown with the same reference as the tests above but with a capital letter used instead.

Test 1E

I have added in CHECK constraints on several tables to ensure they match with my criteria of Objective 5. A specific example is shown below:

```
Marks INT NOT NULL CHECK(TYPEOF(Marks) = 'integer')
```

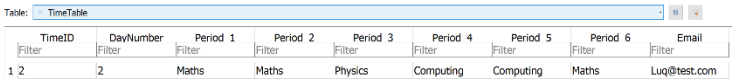
Code Screenshot 1.7 – Check constraint for static data typing

1E	Topic	**200** *(QuestionID)*, **Integrate 6x^2** *(Question)*, **2x^3** *(Answer)*, **One** *(Marks)*, **1** *(Difficulty)*	The *Marks* field has an Integer data type, but a string has been inputted which should cause an error and rejection.	`VALUES ('200', 'Integrate 6x^2', '2x^3', 'One', '1'); ''')` `sqlite3.IntegrityError: CHECK constraint failed: Topic`	Works as expected

I will also program specific functions which check the type of data when a user is required to make an input as this will prevent the program from breaking unnecessarily when a table constraint fails.

Test 1G

As I have decided to have the actual subject names and NULL values, rather than a BOOLEAN data type, for the Period fields, this test has slightly changed. My client also pointed out that many students have a period 6 in the school, so I have added that field as well in the Timetable table.

1G	Timetable	**2** *(TimeID)*, **Luq@test.com** *(Email)*, **2** *(DayNumber)*, **Maths** *(Period_1)*, **Maths** *(Period_2)*, **Physics** *(Period_3)*, **Computing** *(Period_4)*, **Computing** *(Period_5)*, **Maths** *(Period_6)*	This input should be accepted as the data types for the periods have been changed to TEXT.		Works as expected

Test 1I

I have added a constraint to prevent the ScheduleDay field from having a value over 5, and the SchedulePeriod field to have a value over 6.

```
ScheduleDay INT NOT NULL CHECK(TYPEOF(ScheduleDay) = 'integer' AND ScheduleDay<=5),
SchedulePeriod INT NOT NULL CHECK(TYPEOF(SchedulePeriod) = 'integer' AND SchedulePeriod<=6)
```

Code Screenshot 1.8 – Additional check constraints used for Scheduled Table

1H	Scheduled	**1** *(ScheduledID)*, **12** *(ScheduleDay)*, **10** *(SchedulePeriod)*, **Failed** *(Reason)*, **Luq@test.com** *(Email)*	This should cause an integrity error as the check constraint will fail.	`VALUES (1, 12, 10, 'Failed', 'Luq@test.com'); ''')` `sqlite3.IntegrityError: CHECK constraint failed: Scheduled`	Works as expected

Test 1K

1J	Progress	**1** *(ProgressID)*, **10** *(DiffCorrect)*, ***11*** *(DiffTotal)*	Email now has a NOT NULL constraint, so this will cause an error	`VALUES (1, 10, 11); ''')` `sqlite3.IntegrityError: NOT NULL constraint failed: Progress.Email`	Works as expected

Final Code

After thoroughly testing the database initialisation program I have made, I can confirm that all tests have been successful. As a result, this code fully satisfies the criteria (***A***, ***B*** and the description) set out in ***Objective 5*** meaning that ***Objective 5*** has now been completed. The final code with all the amendments for the completion of this objective is shown below.

```
#Mathematics Database
#Importing sqlite3 module which is how I can manage the database
import sqlite3
#I always need to explicitly connect to the database to register changes#
connection = sqlite3.connect("Mathematics.db")
#Using a cursor in order to fetch results when needed
crsr = connection.cursor()
#Using scripts that were made in the Design Section with a few additions

#Creating User Table
#Email, FirstName and LastName data types will be checked by seperate functions
crsr.execute('''CREATE TABLE IF NOT EXISTS User (
    Email VARCHAR(60) NOT NULL UNIQUE,
    FirstName VARCHAR(15) NOT NULL,
    LastName VARCHAR(15) NOT NULL,
    HashedPassword CHAR(64) NOT NULL,
    Student_Status BOOLEAN NOT NULL,
    PRIMARY KEY (Email)); ''')
#Creating Topic Table
crsr.execute('''CREATE TABLE IF NOT EXISTS Topic (
    QuestionID INT NOT NULL UNIQUE,
    Question VARCHAR(255) NOT NULL CHECK(TYPEOF(Question) = 'text'),
    Answer VARCHAR(50) NOT NULL CHECK(TYPEOF(Answer) = 'text'),
    Marks INT NOT NULL CHECK(TYPEOF(Marks) = 'integer'),
    Difficulty INT NOT NULL CHECK(TYPEOF(Difficulty) = 'real'),
    PRIMARY KEY (QuestionID));
    ''')
#Creating Progress Table
crsr.execute('''CREATE TABLE IF NOT EXISTS Progress (
    ProgressID INT NOT NULL UNIQUE,
    DiffCorrect INT NOT NULL DEFAULT 0,
    DiffTotal INT NOT NULL DEFAULT 0,
    IntCorrect INT NOT NULL DEFAULT 0,
    IntTotal INT NOT NULL DEFAULT 0,
    StatCorrect INT NOT NULL DEFAULT 0,
    StatTotal INT NOT NULL DEFAULT 0,
    TrigCorrect INT NOT NULL DEFAULT 0,
    TrigTotal INT NOT NULL DEFAULT 0,
    AvgDifficulty FLOAT NOT NULL DEFAULT 1,
    Email VARCHAR(60) NOT NULL,
    PRIMARY KEY(ProgressID),
    FOREIGN KEY(Email) REFERENCES User(Email));
    ''')
```

Code Screenshot 1.9 – Amended code for creating User, Topic and Progress Tables

```
#Creating Timetable Table to store when the user is free in a specific school period#
#Removed NOT NULL cosntraints on periods and have TEXT data type to provide more information
#to a user when using the program, and allows null to be taken as 'available' for the scheduler
crsr.execute('''CREATE TABLE IF NOT EXISTS TimeTable (
    TimeID Integer NOT NULL,
    DayNumber INT NOT NULL CHECK(TYPEOF(DayNumber) = 'integer' and 0<DayNumber<=5),
    Period_1 TEXT,
    Period_2 TEXT,
    Period_3 TEXT,
    Period_4 TEXT,
    Period_5 TEXT,
    Period_6 TEXT,
    Email VARCHAR(60) NOT NULL,
    FOREIGN KEY (Email) References User(Email),
    PRIMARY KEY (TimeID, DayNumber));
    ''')

#Creating the Scheduled Table
crsr.execute('''CREATE TABLE IF NOT EXISTS Scheduled (
    ScheduleID INT NOT NULL UNIQUE,
    ScheduleDay INT NOT NULL CHECK(TYPEOF(ScheduleDay) = 'integer' AND ScheduleDay<=5),
    SchedulePeriod INT NOT NULL CHECK(TYPEOF(SchedulePeriod) = 'integer' AND SchedulePeriod<=6),
    Reason VARCHAR(255) NOT NULL,
    Email VARCHAR(60) NOT NULL,
    FOREIGN KEY(Email) REFERENCES User(Email),
    PRIMARY KEY (ScheduleID));
    ''')
#Creating the MathsInvader table to store data related to the game
crsr.execute('''CREATE TABLE IF NOT EXISTS MathsInvader (
    GameID INT NOT NULL UNIQUE,
    Email VARCHAR(60) NOT NULL,
    ProgressID INT NOT NULL,
    GameScore INT NOT NULL DEFAULT 0,
    MathScore INT NOT NULL DEFAULT 0,
    FOREIGN KEY(Email) REFERENCES User(Email),
    FOREIGN KEY(ProgressID) REFERENCES Progress(ProgressID),
    PRIMARY KEY (GameID));
    ''')

#To ensure the code above is executed, I have to use the commit function
connection.commit()
#This closes the connection to register all changes made to the database
connection.close()
```

Code Screenshot 1.10 – Amended code for Timetable, Scheduled and MathsInvader Tables

```
#Checks if the argument is an integer
def isInteger(UserInput):
    #uses try and except functionality
    try:
        #Attempts to convert input into an integer
        IntCheck = int(UserInput)
        #If its successful then it returns True
        return True
    #If it fails, then it returns False
    except ValueError:
        return False

#Checks if the argument is a string
def isString(UserInput):
    #uses try and except functionality
    try:
        #Attempts to convert input into a string
        StrCheck = str(UserInput)
        #If its successful then it returns True
        return True
    #If it fails, then it returns False
    except ValueError:
        return False
```

Code Screenshot 1.11 – My datatype checking functions to be used throughout my main program to check user inputs with implementation of error handling statements.

Account System

To differentiate between several users and to give each user a personalised experience in revision and progress tracking, I must create an account system. This will be completed with respect to the criteria set out in ***Objective 2*** (shown below):

A) *Usernames (Emails) must be unique; each user has their own personal account with a username and password which they can access their own data only (except for teachers).*

B) *When logging in or registering, the GUI I make should ensure that the password is hidden (by asterisks for example) to improve security.*

C) *The password should be stored in the database as a secure hashed value: no plaintext storage of passwords further improves the security of accounts and reduces the potential of unwanted access.*

The current GUI system shown in this section (Account System) is only the start of the GUI and may improve when **objective 1** is being attempted in the next part. These foundations for the GUI will allow the account system to be tested and check if it meets all the pre-set criteria. All code and testing of the GUI will be shown in the next section *"**GUI**"* for simplicity.

I have split the account system into two parts: "*Registration*" and "*Logging In*". All code screenshot references in this section will be of the form: ***2.x*** where **x** is an integer.

I have shown the imported modules and scripts used in this part of the program below:

```
#As kivy is massive, it is more efficient to import only a subset
#of the library's scripts
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.textinput import TextInput
from kivy.app import App
from kivy.uix.screenmanager import ScreenManager, Screen, SlideTransition
from kivy.properties import ObjectProperty, StringProperty
from kivy.uix.textinput import TextInput
from kivy.uix.popup import Popup
from kivy.uix.label import Label
from kivy.uix.behaviors import ButtonBehavior
from kivy.uix.image import Image
#Importing the hash library and sqlite3 modules
import hashlib
import sqlite3
```

Code Screenshot 2.1 – Importing required libraries and scripts

Registration

This registration section deals with the code that allows a user to create a new account using the registration screen in my application. It will cover both functions and GUI elements of the registration process that I have made.

```
#Defining my own general version of the popup tool kivy uses
#This is to reduce code repitition and improve efficiency
def PoppingUp(PopName,Text):
    #Defines a new popup with title and text given by arguments of my function
    GeneralPop = Popup(title=PopName,
                content=Label(text=Text),
                size_hint=(None, None),size=(600, 200))
    #Opens this popup on the screen
    GeneralPop.open()
```

Code Screenshot 2.2 – My general defined popup function to be used for informing users of errors

Should read ‘repetition’ in the second comment of code screenshot 2.2 above instead of ‘repitition’ (made a spelling error)

I decided to put the next screenshots on one page for clarity.

```
#Creating the register screen as its own class and inheriting
#from the Screen and BoxLayout Superclasses
class RegisterScreen(Screen, BoxLayout):
    #using a constructor method with variable keyword arguments
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
    #User Registration method for new account
    def CreateAcc(self):
        #creating a connection with the database
        connection = sqlite3.connect("Mathematics.db")
        #Using a cursor in order to fetch results when needed
        crsr = connection.cursor()
        #Queries the emails from the User table in the database to check what is not available
        crsr.execute(''' SELECT Email from User ''')
        #creates an empty list to store emails
        taken = []
        #Iterates through the results of the query
        for row in crsr.fetchall():
            #appends data to 'taken' list as a string
            taken.append((row[0]))
        #Stores all entered text from the GUI in variables to be used
        CreateUser = self.ids.email_field.text
        CreatePass = self.ids.NewPwd_field.text
        CheckPass = self.ids.RePwd_field.text
        NewFname = self.ids.Fname_field.text
        NewLname = self.ids.Lname_field.text
        NewStatus = self.ids.StudentS_field.text
        #Checks that there are no empty fields
        if CreateUser == '' or CreatePass == '' or CheckPass == '' or NewFname == '' or NewLname == '' or NewStatus == '':
            #if an empty field is found, a pop up is used to inform the user to retry
            PoppingUp('Empty Field','One or more fields are empty! Please fill them all in.')
        else:
            #If no empty fields found, the email is checked to see if it is already taken
            if CreateUser in taken:
                #if email is taken then a pop up informing the user to retry is shown
                PoppingUp('Existing Username','This username is already taken! Please try another one.')
            else:
                #if email is unique, it is stored in a new variable to be used
                NewEmail = CreateUser
                #This conditional statement will check which type of user it is (student or teacher)
                if NewStatus.lower() == "student":
                    NewStatus = True
                else:
                    NewStatus = False
                #Checks if the two entered passwords match each other
                if CreatePass != CheckPass:
                    #if they don't match then a pop is used
                    PoppingUp('No Match', 'Passwords do not match! Please try again.')

                else:
                    #Uses sha256 hashing algorithm to ensure the password cannot be easily acquired
                    NewHashP = hashlib.sha256(CreatePass.encode()).hexdigest()
                    #Inserts all new user data, after meeting all required conditions, in the User table
                    crsr.execute(''' INSERT INTO User(Email, FirstName, LastName, HashedPassword, Student_Status)
                    VALUES (?,?,?,?,?)''', (NewEmail, NewFname, NewLname, NewHashP, NewStatus))
                    #To ensure the code above is executed, I have to use the commit function
                    connection.commit()
                    #This closes the connection to register all changes made to the database
                    connection.close()
                    #A pop is used to inform the user that the account has been successfully created
                    PoppingUp('Account Created', "Account Successfully Created!")
```

Code Screenshot 2.3 – Constructing class for registration screen and creating function for making user accounts

In conjunction with Python, I also used the Kv language of Kivy for the implementation, design, and some functions of the GUI including buttons and labels. This is shown below:

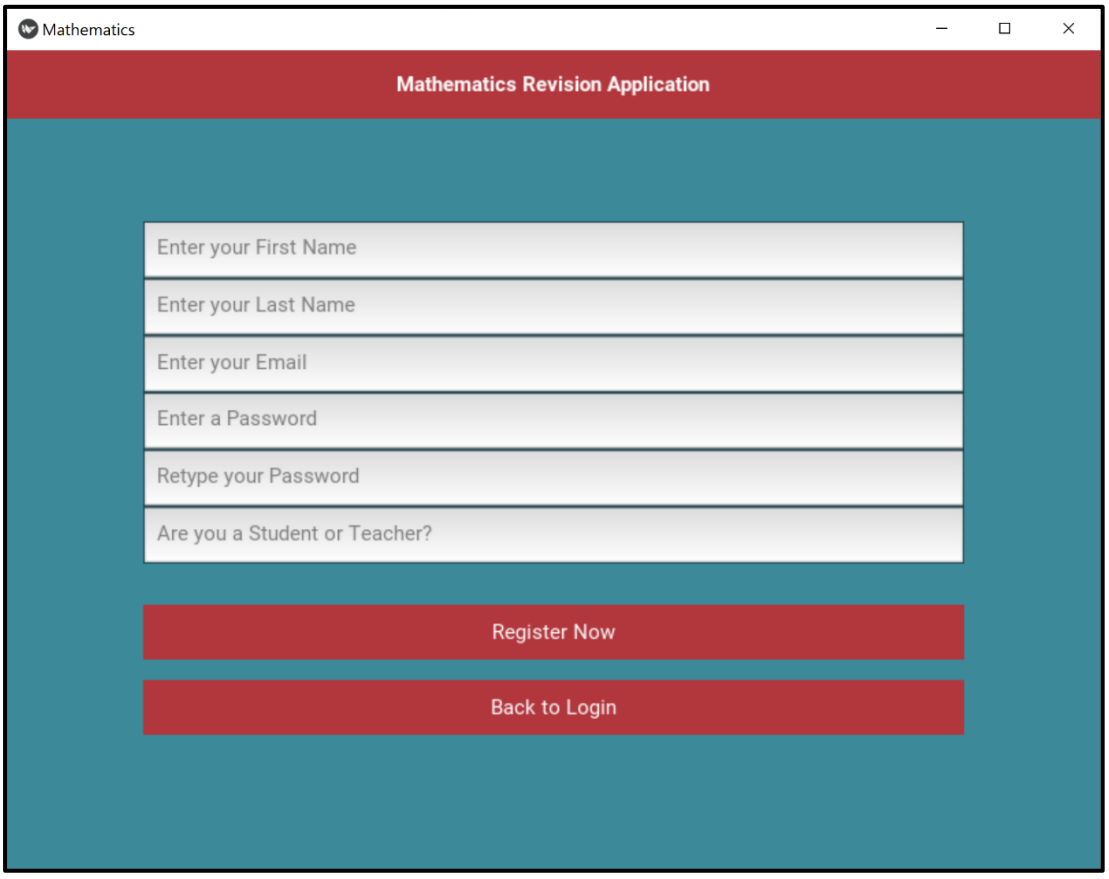
```
<RegisterScreen>:
    id: register_screen
    orientation: "vertical"
    spacing: 10
    space_x: self.size[0]/3
    canvas.before:
        Color:
            rgba: (0.392,0.902,1, .6)
        Rectangle:
            size: self.size
            pos: self.pos

    BoxLayout:
        size_hint_y: None
        height: 50
        pos: 0, 550
        canvas.before:
            Color:
                rgba: (1, 0, 0, 0.6)
            Rectangle:
                size: self.size
                pos: self.pos
        Label:
            text: "Mathematics Revision Application"
            bold: True
            size_hint_x: .9
```

```
    BoxLayout:
        orientation: 'vertical'
        padding: 100, 100
        BoxLayout:
            orientation: "vertical"
            size_hint_y: None
            height: 250
            TextInput:
                padding: 10
                id: Fname_field
                hint_text: "Enter your First Name"
                multiline: False
                focus: True
                on_text_validate: Lname_field.focus = True
            TextInput:
                id: Lname_field
                padding: 10
                hint_text: "Enter your Last Name"
                multiline: False
                on_text_validate: email_field.focus = True
            TextInput:
                id: email_field
                padding: 10
                hint_text: "Enter your Email"
                multiline: False
                on_text_validate: NewPwd_field.focus = True
            TextInput:
                id: NewPwd_field
                padding: 10
                hint_text: "Enter a Password"
                multiline: False
                on_text_validate: RePwd_field.focus = True
            TextInput:
                id: RePwd_field
                padding: 10
                hint_text: "Retype your Password"
                multiline: False
                on_text_validate: StudentS_field.focus = True
            TextInput:
                id: StudentS_field
                padding: 10
                hint_text: "Are you a Student or Teacher?"
                multiline: False
```

Code Screenshot 2.4 – Kivy File configuration for registration screen and class

```
        Label:
            id: sp
            size_hint_y: None
            height: 30

        Button:
            text: "Register Now"
            size_hint_y: None
            pos: (0,0)
            height: 40
            background_color: (1,0,0, 0.6)
            background_normal: ''
            on_release: root.CreateAcc()

        Label:
            id: sp
            size_hint_y: None
            height: 15

        Button:
            text: "Back to Login"
            size_hint_y: None
            pos: (0,0)
            padding: 0,25
            height: 40
            background_color: (1,0,0, 0.6)
            background_normal: ''
            on_release: root.manager.current = 'Login'
```

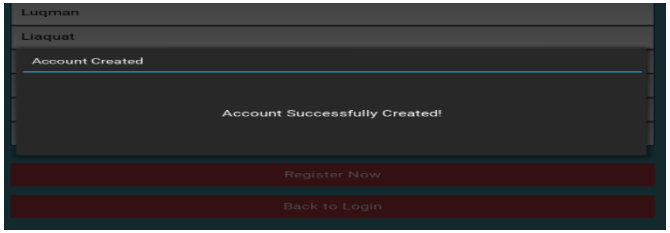
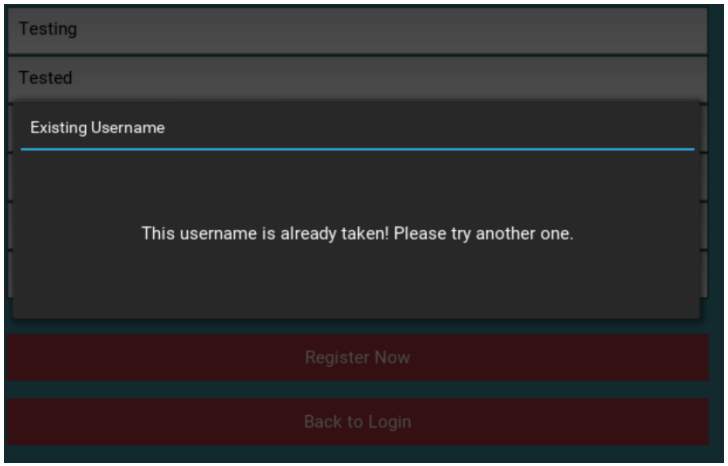
Code Screenshot 2.5 – Continuation from Code Screenshot 2.4

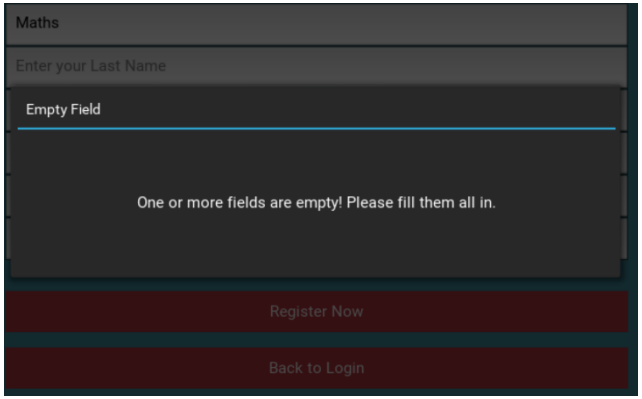
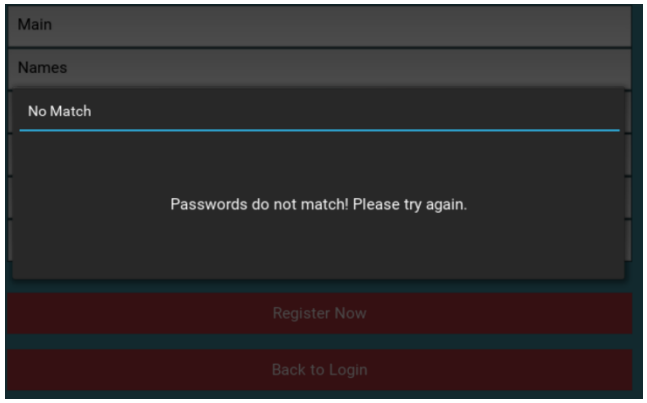


Code Screenshot 2.6 – GUI Registration Screen after running Python file with Kivy file configuration.

Alpha Testing – Registration

As I have changed some elements of the program since the design section, mainly the GUI, I have made a few changes to the input of the alpha tests.

Test Reference	Section	Input	Expected Output	Actual Output	Comments
2a	Register	**"Luqman"** *(Enter First Name)* **"Liaquat"** *(Enter Last Name)* "**Luq@test.com**" *(Enter Email)* **"password"** *(Enter Password)* **"password"** *(Retype Password)* **"student"** *(Teacher or Student)*	This should successfully create a new account, store the user details in the database and notifies the user through a popup.	 Table: User Email FirstName LastName HashedPasswor Student Status Filter Filter Filter Filter Filter 1 Luq@test.com Luqman Liaquat 5e884898da28... 1	**Works As Expected**
2b	Register	**"Testing"** *(Enter First Name)* **"Tested"** *(Enter Last Name)* "**Luq@test.com**" *(Enter Email)* **"password"** *(Enter Password)* **"password"** *(Retype Password)* **"student"** *(Teacher or Student)*	As this username as already been created, the program should notify the user, through a popup, to try a different one.		**Works As Expected**

2c	Register	**"Maths"** *(Enter First Name)* "**Testing1@test.com**" *(Enter Email)* **"alpha"** *(Enter Password)* **"alpha"** *(Retype Password)* **"teacher"** *(Teacher or Student)*	The last name field has been left empty which means the user will be notified to fill in the field.		**Works As Expected**
2d	Register	**"Main"** *(Enter First Name)* **"Names"** *(Enter Last Name)* "**Alpha@test.com**" *(Enter Email)* **"Password"** *(Enter Password)* **"NotPassword"** *(Retype Password)* **"Teacher"** *(Teacher or Student)*	The password does not match with the retyped password, so the program should ask the user to retry.		**Works As Expected**

Amendments

I realised that the registration screen also requires a hidden password field to fully meet the requirements of objective 2B which I did not include initially. This has been amended, using the Password: True functionality of a GUI label. All other aspects of the code have met their requirements and so do not need amending. Here is an example of a completed registration form (as it could not be shown in the table above):

Code Screenshot 2.7 – Completed Registration Form on my GUI

Upon realisation of requiring more complex SQL queries, I decided to create an updated data search function which allows me to pass an SQL query as an argument and for it to iterate through the database and construct a list with the data (rather than the raw tuples that are received from a usual SQLite script). This was created during the **MATHS TOPIC TEST Section** but was moved here as it was the most logical place for it as I used it within the registration function.

```
#A more useful DataSearch Function where I can define my own query
#and convert the recieved list of tuples into a list of mutable data items
def DataSearch(Query):
    #connecting to database
    DataConnect('connect')
    #Creating an executable query which i can define to select data
    crsr.execute(Query)
    #creating an empty list
    StoredList = []
    #Iterates through the results of the query
    for row in crsr.fetchall():
    #appends data to 'StoredList' list as a string
        StoredList.append((row[0]))
    #The list of data is returned
    return StoredList
```

Code Screenshot 2.8 – New DataSearch function to make querying data from my database easier within other functions and methods.

```
#Sets new ids for any of the table to ensure there is no repeated ids
def IncrementID(Query):
    #A MAX SQL aggregate function will be used in conjunction with this
    #It will use my DataSearch function to find the highest ID already set in the desired table
    MaxList = DataSearch(Query)
    #The value of the id will be stored in a variable called MaxID
    MaxID = MaxList[0]
    #This condition will check if there is nothing in the database for that table in the ID column
    if MaxID == None or MaxID == '':
        #If there isn't, then it is a new entry and therefore the ID will be set to 1
        NewID = 1
    else:
        #Otherwise, it will add one onto the MaxID
        NewID = int(MaxID) + 1
    #The NewID is returned by the function
    return NewID
```

Code Screenshot 2.9 – creating function to assign the correct ID to a user for any table within the database that requires it.

```
#Creating a new ID for the progress table for this new user
NewID = IncrementID('''Select MAX(ProgressID) FROM Progress''')
#Inserting this into the progress table
crsr.execute('''INSERT INTO Progress(ProgressID, Email) VALUES(?,?)''',(NewID, NewEmail))
#closing the connection with database
DataConnect('disconnect')
#A pop is used to inform the user that the account has been successfully created
PoppingUp('Account Created', "Account Successfully Created!")
```

Code Screenshot 2.10 – Change made to registration function (near end of the function)

Logging In

This section will contain all the code and GUI elements for the login page including all the functions that I have made to work with it.

I have made some updates to the code for an improvement in the efficiency of the program. This will affect the code for the ***Registration*** section, but will not change any functionality of it at all: it is just a more efficient implementation of the code. I have created functions which reduce data redundancy as I have realised there are some blocks of code that will be continuously reused, and a procedure or function will make it much more efficient. All tests are still successful with this update.

Firstly, I have created a database connection and disconnection function which is used anytime the database is involved in the program (most of the time it is). I provided both the connecting and disconnecting code in one function with the option of which code is used depending on the parameter.

```
#Connecting or Disconnecting from database function
def DataConnect(process):
    if process == 'connect':
        #creating a connection with the database, it will be global as
        #other queries will use it, but the connection itself won't ever change
        global connection
        connection = sqlite3.connect("Mathematics.db")
        #Using a global cursor in order to fetch results from any query when needed
        global crsr
        crsr = connection.cursor()
    elif process == 'disconnect':
        #To ensure the SQL Query is executed
        connection.commit()
        #This closes the connection to register all changes made to the database
        connection.close()
```

Code Screenshot 2.11 – Database connecting and disconnecting function to reduce data redundancy

I have also created a generalised 'FindData' function which iterates through a query's results and stores the data in a list. As this type of code will be reused multiple times, I decided to make a function to reduce redundancy and improve efficiency. The code is shown below:

```
#My generalised funtion for returning data from a database in a list for comparisons.
def FindData(Column, Table):
        #connecting to database
        DataConnect('connect')
        #Creating sql query to select data
        crsr.execute(''' SELECT %s from %s ''' %(Column, Table))
        #creating an empty list
        StoredList = []
        #Iterates through the results of the query
        for row in crsr.fetchall():
            #appends data to 'StoredList' list as a string
            StoredList.append((row[0]))
        return StoredList
```

Code Screenshot 2.12 – Generalised function for finding data for a single column (such as emails) and storing it in a list.

These functions have replaced any code with their functionality within the registration section.

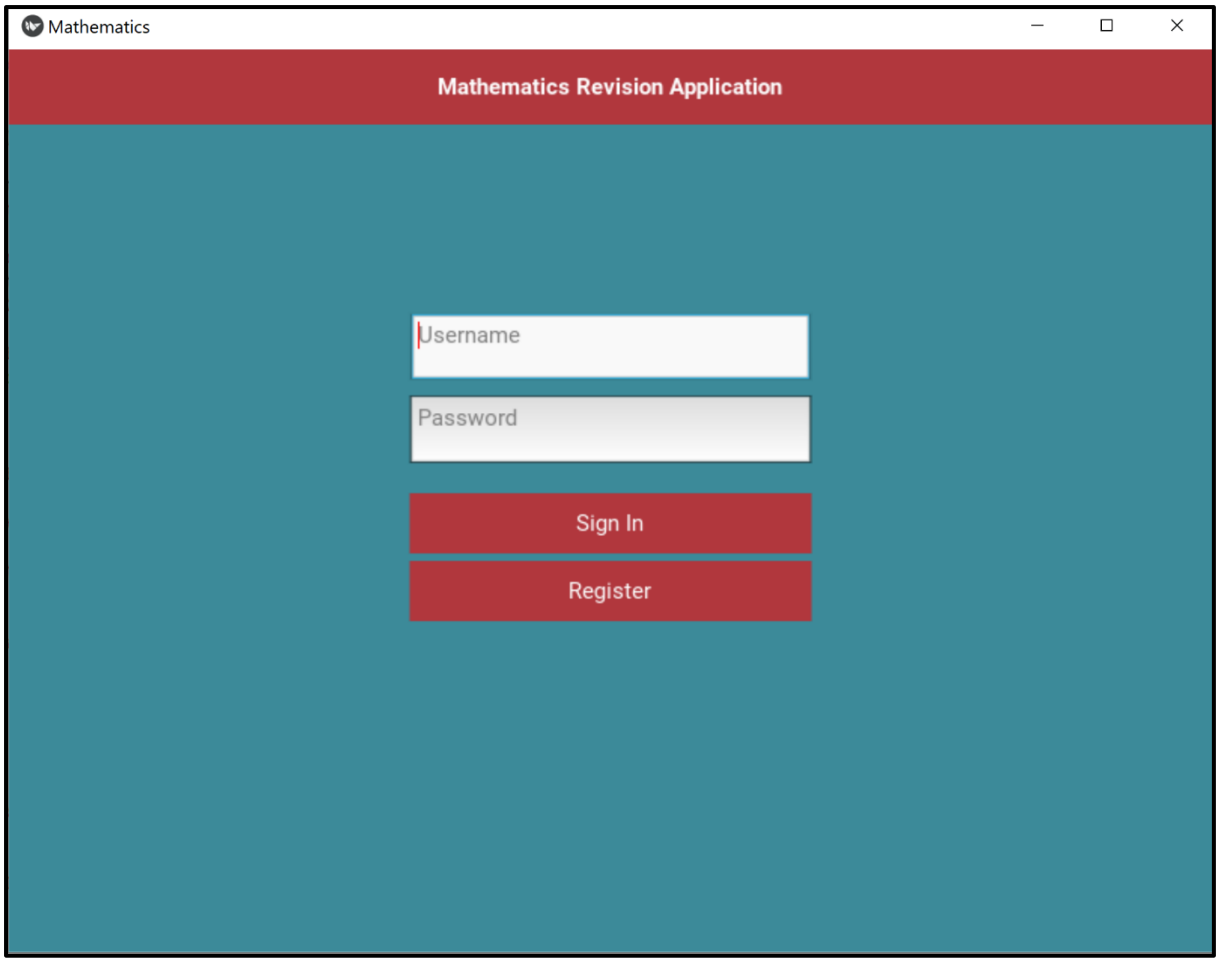
```
#Creating the log in screen as its own class and inheriting
#from the Screen and BoxLayout Superclasses
class LoginScreen(Screen, BoxLayout):
   #Creating subroutine linked to Log In button
   def LogIn(self):
      #creates a list of all emails
      StoredEmails = FindData('Email', 'User')
      #stores user input in Username field in a variable
      UserCheck = self.ids.username_field.text
      #Checks if the user input is in the list of Stored emails from the database
      if UserCheck in StoredEmails:
         #If the condition is met, the program
         PassCheck = self.ids.password_field.text
         #password entered is hashed with sha256 for verification checks
         PassCheck = hashlib.sha256(PassCheck.encode()).hexdigest()
         #Uses string formatting to query hashed password from database
         crsr.execute(''' SELECT HashedPassword FROM User WHERE Email=? ''', (UserCheck,))
         for row in crsr.fetchall():
            #Stores hashed value in different variable
            PassChecked = row[0]
         #compares hashed user input with hashed password in database
         if PassCheck == PassChecked:
            #if they match then they are presented with the home screen
            self.manager.current = 'Home'
         #Otherwise an error pops up telling them their password is incorrect
         else:
            PoppingUp('Password Error', 'Your password is incorrect')
      #if the username doesn't exist in the database, the user is informed.
      else:
         PoppingUp('Username Error','Username Not Found!')
```

Code Screenshot 2.13 – LogIn function

The corresponding **kv** code for the GUI features I have included in the program is shown on the next page.

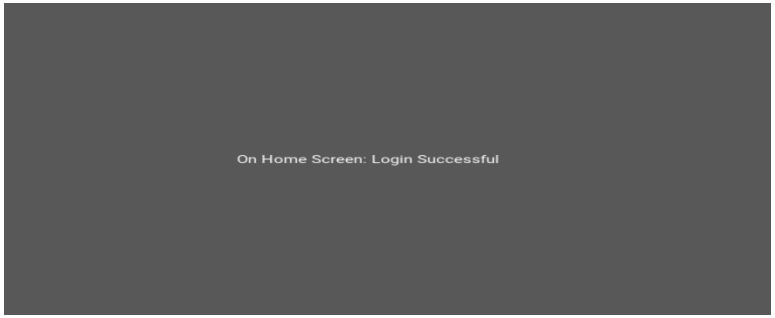
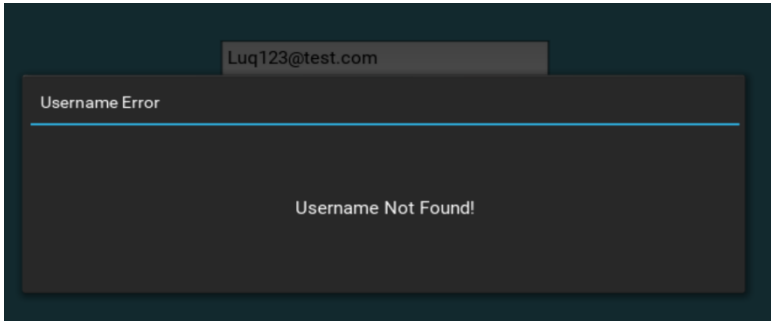
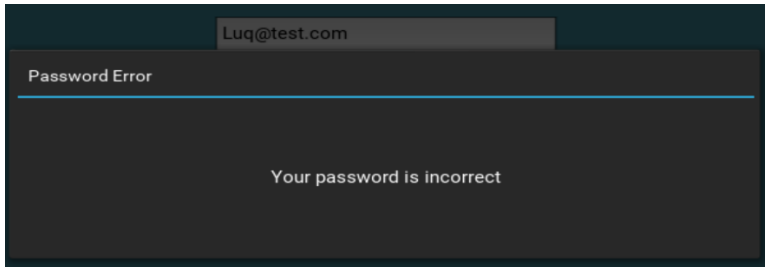
```
<LoginScreen>:
    id: login_screen
    orientation: "vertical"
    spacing: 10
    space_x: self.size[0]/3
    canvas.before:
        Color:
            rgba: (0.392,0.902,1, .6)
        Rectangle:
            size: self.size
            pos: self.pos

    BoxLayout:
        size_hint_y: None
        height: 50
        pos: 0, 550
        canvas.before:
            Color:
                rgba: (1, 0, 0, 0.6)
            Rectangle:
                size: self.size
                pos: self.pos
        Label:
            text: "Mathematics Revision Application"
            bold: True
            size_hint_x: .9


    BoxLayout:
        orientation: 'vertical'
        padding: login_screen.space_x, 200
        #spacing: 30
        BoxLayout:
            orientation: "vertical"
            spacing: 10
            size_hint_y: None
            height: 100
            TextInput:
                id: username_field
                hint_text: "Username"
                multiline: False
                focus: True
                on_text_validate: password_field.focus = True
            TextInput:
                id: password_field
                hint_text: "Password"
                multiline: False
                password: True
        Label:
            id: sp
            size_hint_y: None
            height: 20

        Button:
            text: "Sign In"
            size_hint_y: None
            height: 40
            background_color: (1,0,0, 0.6)
            background_normal: ''
            on_release: root.LogIn()

        Label:
            id: sp2
            size_hint_y: None
            height: 5

        Button:
            text: "Register"
            size_hint_y: None
            height: 40
            background_color: (1,0,0, 0.6)
            background_normal: ''
            on_release: root.manager.current = 'Register'

        Label:
            id: sp3
            size_hint_y: None
            height: 20
```

Code Screenshot 2.14 - Corresponding KV code used with GUI



Code Screenshot 2.15 – Login Screen, first screen the user will see when running the application

Alpha Testing – Logging In

Test Reference	Section	Input	Expected Output	Actual Output	Comments
2e	Login	"**Luq@test.com**" *(Enter Email),* "**password**" *(Enter password)* **Press LogIn**	As this user is already registered from previous tests, the program should successfully open the application's home page (home page design and functionality will be completed in the GUI section)		**Works As Expected**
2f	Login	"**Luq123@test.com**" *(Enter Email),* "**Password23243**" *(Enter password)* **Press LogIn**	As this user is not registered, the program should notify the user that their username was not found, through a popup		**Works As Expected**
2g	Login	"**Luq@test.com**" *(Enter Email),* "**Password23243**" *(Enter password)* **Press LogIn**	As this user is registered, but the password they have typed is incorrect, the user will be notified, through a popup.		**Works As Expected**

General GUI

To improve clarity in my program, I have separated the logical operations and functions of the main program from the design elements of the GUI which is stored in a **.kv** text file (as shown in the previous section). I found this makes the program more manageable and allows me to have a greater scope of options in how I would like to design the GUI, ensuring it meets with the Client's specifications.

I used the Kivy library to aid me in creating a well-designed GUI. As this was new to me, I had conducted further research into the library and how to use several features. I first used the Kivy documentation itself as that provided the basics in getting started and using the new tools[11]. I also used video tutorials on how to use the screen manager and implementing object-oriented programming into the GUI for easier management of the application[12,13].

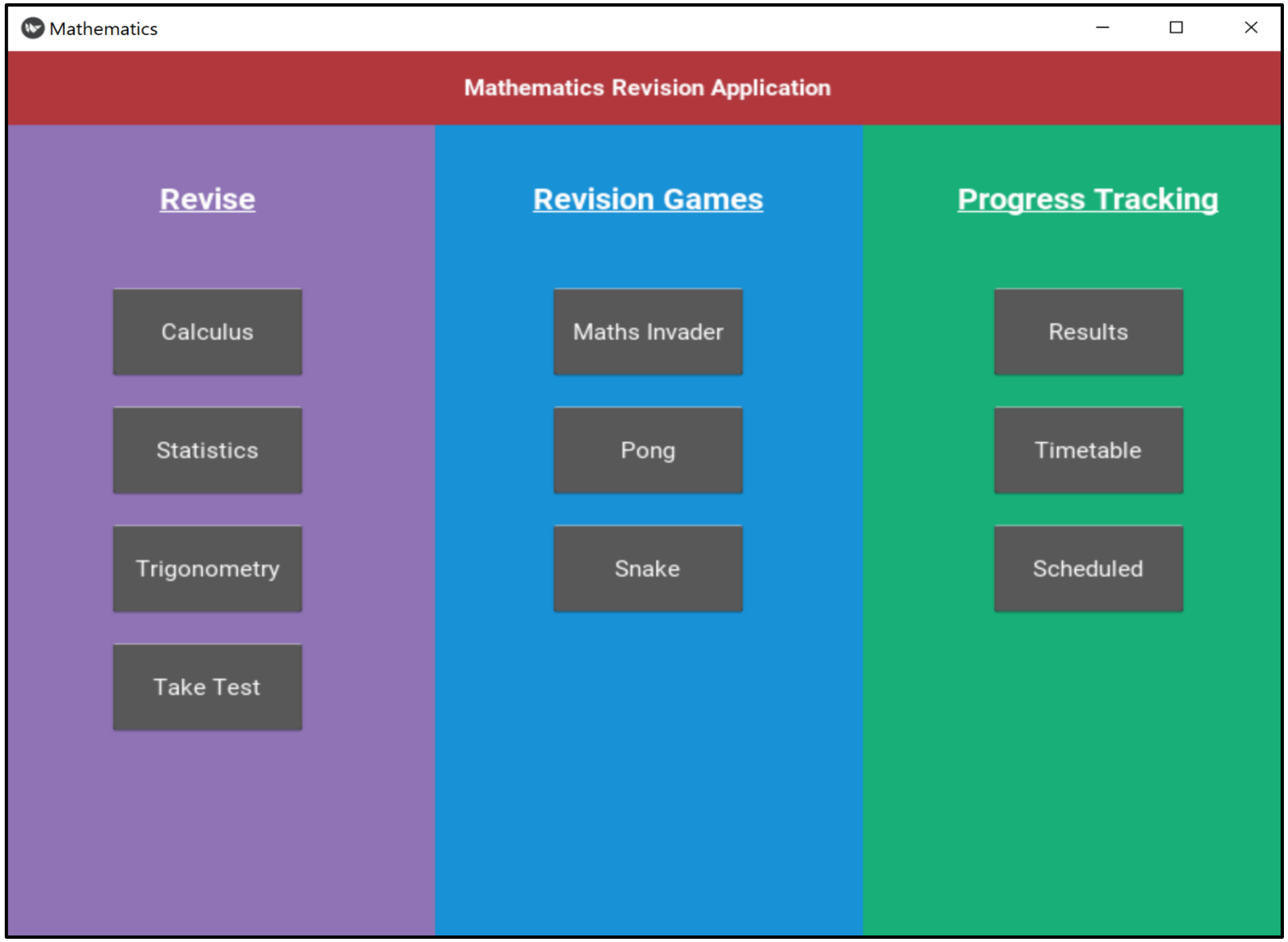
The object-oriented and class system provides me with the basis and blueprints to create the application where I can instantiate objects easily. The code from my kv file is then imported to work together with this system to create the GUI. This uses inheritance from some Kivy classes and using an arbitrary number of positional or keyword arguments through the *args and **kwargs parameters for each class. This allows me to define any range of functions with a certain number of arguments

```
#Creating the seperate screens as their own classes and inheriting
#from the Screen and respective layout Superclasses
class FeatureScreen(Screen, FloatLayout):
    #using a constructor method with variable keyword arguments
    def __init__(self, **kwargs):
        super().__init__(**kwargs)

class HomeScreen(Screen, BoxLayout):


class InvaderScreen(Screen, BoxLayout):
    #using a constructor method with variable keyword arguments
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
    #MathsInvaders.MathsInvaderGame()

class ProgressScreen(Screen, BoxLayout):
    #using a constructor method with variable keyword arguments
    def __init__(self, **kwargs):
        super().__init__(**kwargs)

class CalculusScreen(Screen, BoxLayout):
    #using a constructor method with variable keyword arguments
    def __init__(self, **kwargs):
        super().__init__(**kwargs)


#Creating a manager class which will handle the different screens
#of the application's GUI
class Manager(ScreenManager):
    login_screen = ObjectProperty(None)
    home_screen = ObjectProperty(None)
    register_screen = ObjectProperty(None)
    feature_screen = ObjectProperty(None)
    invader_screen = ObjectProperty(None)
    progress_screen = ObjectProperty(None)
    calculus_screen = ObjectProperty(None)


#This defines the overall application gui class
class Mathematics(App):
    #the builder method will return manager class as that handles the screens
    def build(self):
        return Manager()
```

Code Screenshot 3.1 – Structure of GUI using several classes, inheriting from super classes provided by Kivy, then importing my kv code to ensure it runs properly. Not all classes are shown in this screenshot.

```
<HomeScreen>:
    id: home_screen
    orientation: "vertical"
    spacing: 10
    space_x: self.size[0]/3

    BoxLayout:
        size_hint_y: None
        height: 50
        pos: 0, 550
        canvas.before:
            Color:
                rgba: (1, 0, 0, 0.6)
            Rectangle:
                size: self.size
                pos: self.pos
        Label:
            text: "Mathematics Revision Application"
            bold: True
            size_hint_x: .9

    BoxLayout:
        size_hint_y: 0.333
        pos: 0, 0
        canvas.before:
            Color:
                rgba: (0.78,0.39,0.78, .6)
            Rectangle:
                size: 267,550
                pos: 0,0
    FloatLayout:
        size: self.size
        Label:
            markup: True
            text: "[size=20][u][b]Revise[/b][/u][/size]"
            pos: (-275,200)
        Button:
            text: "Calculus"
            pos: (65, 380)
            size_hint: (0.15,0.10)
            on_release: root.manager.current = 'Calculus'
        Button:
            text: "Statistics"
            pos: (65, 300)
            size_hint: (0.15,0.10)
        Button:
            text: "Trigonometry"
```

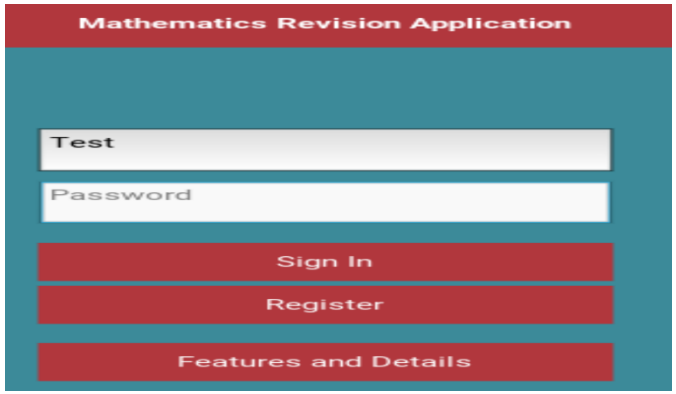
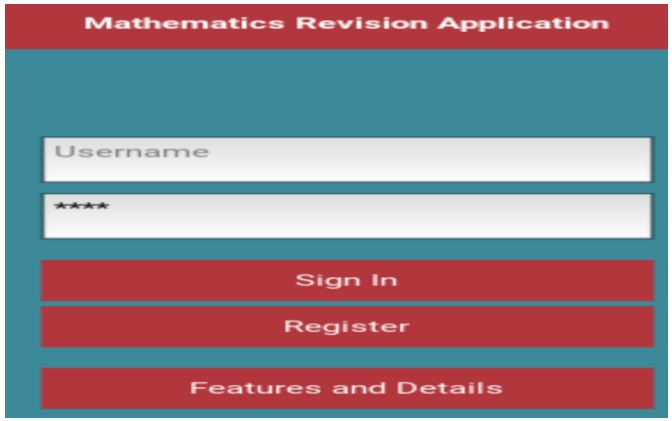
Code Screenshot 3.2 – My Home Screen kv code (some parts left out as there was repetition to create other buttons

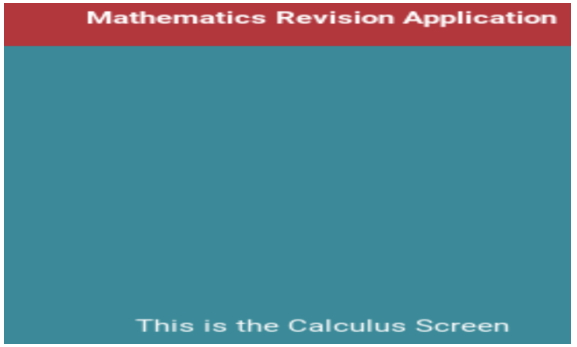
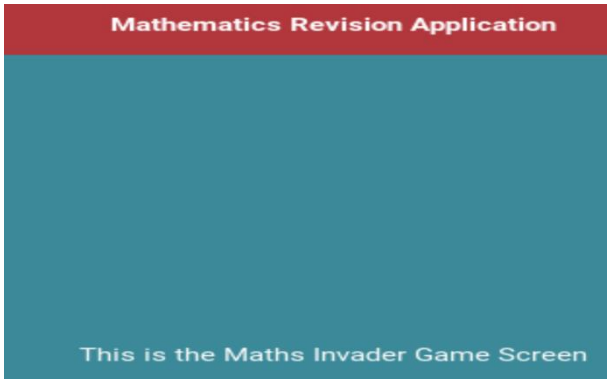


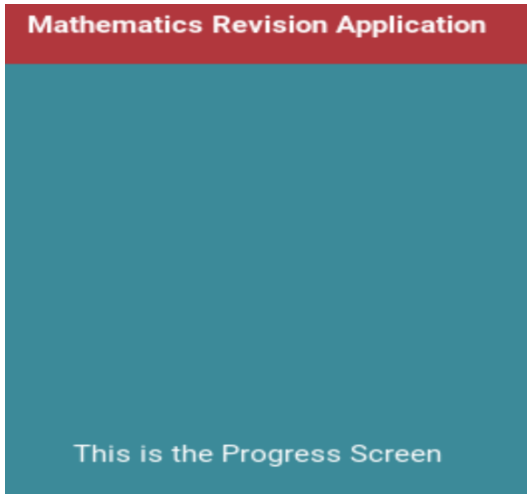
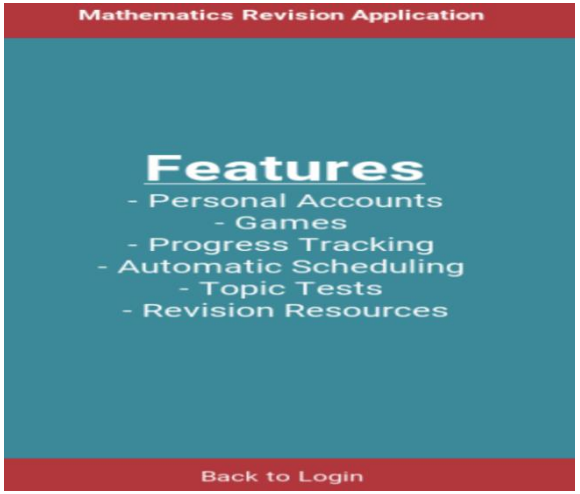
Code Screenshot 3.3 – Home screen compliant with Client's requests on background colouring

Alpha Testing – General GUI

Please note test 3i has changed after realising that the GUI itself has the ability to minimise and maximise and additional functionality does not need to be included

Test Reference	GUI Section	Input	Expected Output	Actual Outcome	Comments
3a	Log In Screen (Figure 4A)	Type "**Test**" into '*Username*' field	"**Test**" will be displayed within the Username Field		**Works As Expected**
3b	Log In Screen (Figure 4A)	Type "**Test**" into '*Password*' field	"********" will be displayed in the 'Password' field		**Works As Expected**

3c	Log In Screen (Figure 4A)	Press "**Login**" Button	The function defined for Logging in will run: LogIn()	Shown in test 2e	**Works As Expected**
3d	Log In Screen (Figure 4A)	Press "**Register**" Button	The user will be taken to the registration screen	Shown in tests 2a – 2d	**Works As Expected**
3e	Home Screen (Figure 4B)	Press "**Topic 1**" Button	Specific Topic 1 page will open.		**Works As Expected**
3f	Home Screen (Figure 4B)	Press "**MathsInvader**" Button	Maths Invader Game will be launched		**Works As Expected**

3g	Home Screen (Figure 4B)	Press "**Progress**" Button	User Progress Page will be opened		**Works As Expected**
3h	Log In Screen (Figure 4A)	Press "**Features and Details**"	The application will take the user to the Features and Details screen		**Works As Expected**

There are no amendments as all tests came back with a 'Works as Expected' Result.

Revision Sections

To be successful in ***part of objective 4*** and ***all of objective 7***, I need to create a revision section with useful interactive tools that can help students in at least three topics. These topics are Calculus, Trigonometry and Statistics. These topics cover four of the topics listed in my objective 7 description. The content, design and code for any procedures/functions I created will be shown will be shown on the next few pages for each topic.

The objective this section maps to the support section aspect of ***objective 4***:

"A support section for revision which includes at least one game to prevent boredom and increase student's memory of the topic."

It also maps to all of ***objective 7:***

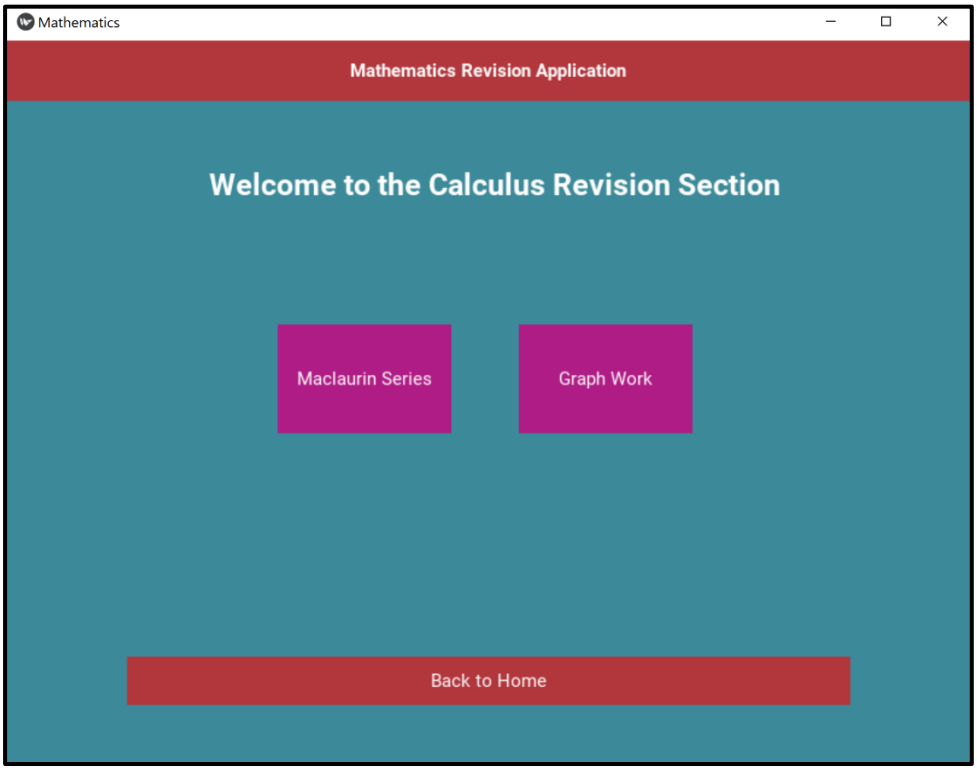
Have 3 major maths topics from differentiation, integration, Vectors, Data analysis (statistics) – the student can be shown how to find different values such as standard deviation when they input their own set of values.	Each topic should have its own set of revision materials to help the student revise.

Calculus

The calculus section will cover some revision material for both differentiation and integration of functions. Some important uses of calculus include gradients of functions; areas under curves and polynomial series expansions of functions. However, there is a multitude of applications of calculus that cannot be covered due to the limited time span of the project.

This section features two parts. The first part is working through the McLaurin Series Expansions with derivatives. This will allow a student to find the series expansions of a function they define. As this series expansion requires the ability to use factorials, I will also include a recursive factorial function for efficient calculations.

The second part is the graphing aspect where a user can use the tools in my program to plot a graph and practice working out the area or gradient in a given set of values. The graph can also be changed, and an undo button will be provided to the user. This will use a stack data structure (LIFO) to allow previous functions to be recalled and undo the current one. I learnt about stacks in my computer science A-level main theory lessons.



Code Screenshot 4.1 – Main Calculus Revision Screen

The above screenshot (*4.1*) shows the main calculus screen which currently has two purple coloured buttons that lead onto the actual revision screens. These buttons follow the colour scheme of the revision section as my client wanted. I decided to have this 'middle' screen instead of giving a direct link to the two individual calculus revision screens from the home page. This is because my decision will prepare my application for any expanding that could to be done, if time allows, such as the improvement of the revision resources once the program and logic have all been completed to the client's expectations.

The following two pages cover the series expansion aspect of the calculus revision section. This includes the functionality of using the factorial generators with the GUI.

```
class CalculusSeries(Screen, BoxLayout):
    #using a constructor method with variable keyword arguments
    #def __init__(self, **kwargs):
     #   super().__init__(**kwargs)
    #Defining the recursive factorial function
    def Factorial(n):
        #For recursion, I must establish my base case and I will cover all
        #non-negative integers as that is what the series uses, starting from 0
        if n == 0:
            #this is for returning 1 if n = 0
            return 1
        else:
            #Otherwise it calls itself recursively to find the nth factorial number
            return n * CalculusSeries.Factorial(n-1)
```

Code Screenshot 4.2 – Defining the CalculusSeries class and inheriting from important superclasses for GUI. Also defining my factorial function which is a recursive algorithm that will provide the nth factorial number for series expansion

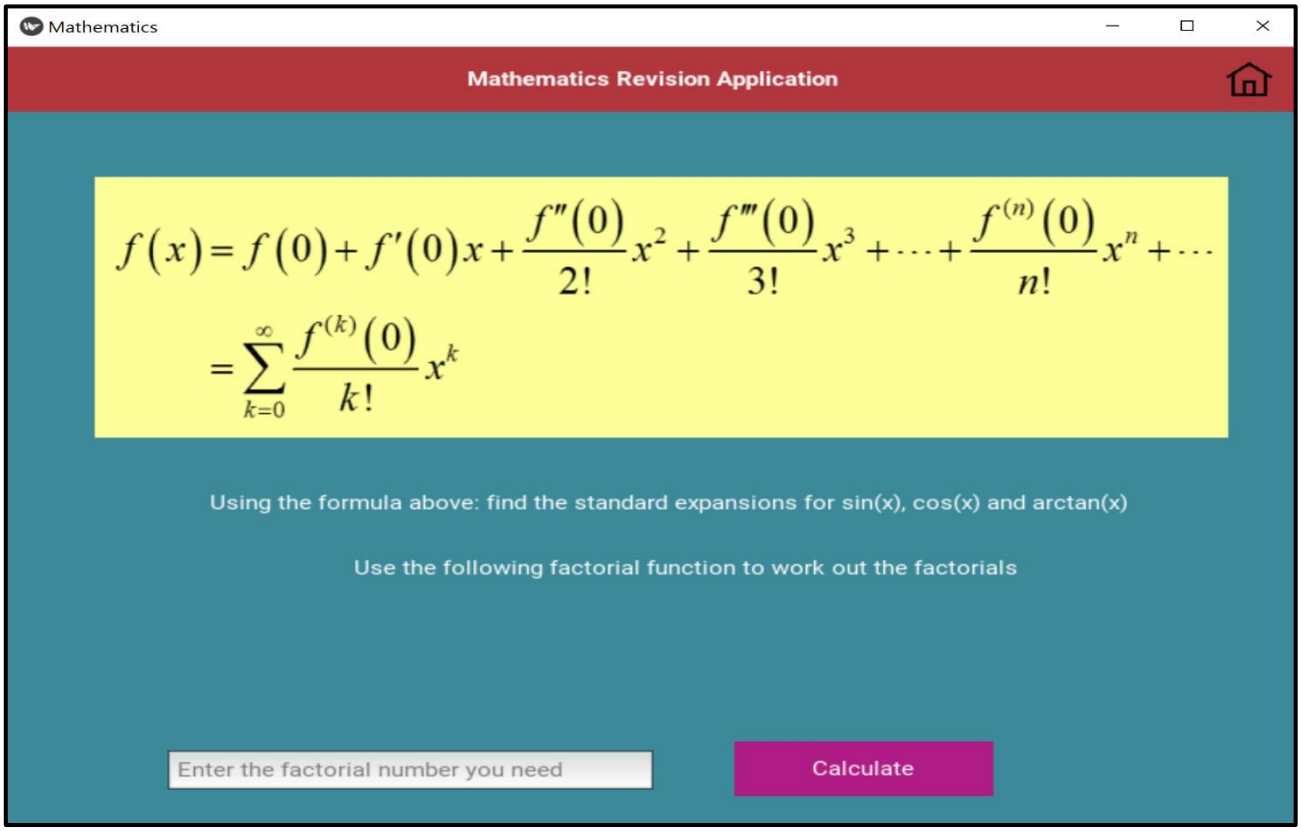
```
#defining factorial link to user interface function when calculate button is pressed
def giveFactorial(self):
    #taking in user input from the text input field
    InputNumber = self.ids.Factorial.text
    #checks if the number is an integer and the field is not empty
    if InputNumber != '' and isInteger(InputNumber):
        InputNumber = int(InputNumber)
        #applying my Factorial function to it and storing it in the variable
        FactorialNumber = CalculusSeries.Factorial(InputNumber)
        #Providing a popup to tell user what their answer was
        PoppingUp('Factorial', 'Your requested answer to %d Factorial is: %d' %(InputNumber, FactorialNumber))
    #otherwise a pop up shows that you need to type an integer into the field.
    else:
        PoppingUp('Error Factorial','Please Type an Integer!')
```

Code Screenshot 4.3 – Defining the output factorial method within the CalculusSeries class, using a combination of my previously defined data type checking function, PoppingUp function and Factorial function.

Corresponding KV code

```
<CalculusSeries>:
    id: calc1_screen
    MainWidget
    FloatLayout:
        HomeButton:
            pos: 743, 560
            on_release: root.manager.current = 'Home'

    BoxLayout:
        orientation: 'vertical'
        canvas:
            Rectangle:
                source: 'Maclaurin.gif'
                size: 700,200
                pos: 55,300
    FloatLayout:
        Label:
            text: "Using the formula above: find the standard expansions for sin(x), cos(x) and arctan(x)"
            pos: 10, -50

        Label:
            text: "Use the following factorial function to work out the factorials"
            pos: 20, -100

        Button:
            background_normal: ''
            background_color: 0.8,0,0.5,0.8
            pos: 450,25
            size_hint_x: 0.20
            size_hint_y: 0.07
            text:"Calculate"
            on_release: root.giveFactorial()
```

Code Screenshot 4.4 – Corresponding KV code with imported image formula to add into calculus revision section



Code Screenshot 4.5 – Revision screen with a practice question and the factorial calculator

The following four pages cover the **graphing** section of the Calculus revision section.

```
#Creating my stack data structure with its related methods as its own class
#This allows me to create a stack by instantiating it as an object of the class and
#always have these methods available to use with the stack
class Stack():
    #Constructor method creates a Python list and is used with object
    def __init__(self):
        self.elements = []
    #defining push method which adds a data element on the top of the stack
    def push(self, element):
        self.elements.append(element)
    #defining pop method which pops off the top element item
    def pop(self):
        return self.elements.pop()
    #defining method to return the contents of the stack when called
    def viewStack(self):
        return self.elements
    #Also have a method to check if the stack is empty
    def isEmpty(self):
        #checks if the stack equals an empty stack
        if self.elements == []:
            #if it is empty then true is returned by the method
            return True
        else:
            #otherwise false is returned by the method
            return False
```

Code Screenshot 4.6 – Creating a new class for my LIFO stack data structure with its methods to be used in the undo aspect of the graphing system. This code is placed at the top of the program before the classes for the user interface are created

```
class CalculusGraphs(Screen):
    #Using my Stack class to instantiate a stack data structure to store the details
    #of the graph and make this stack available to any function within this CalculusGraph Class
    global StoredEquations
    StoredEquations = Stack()
    #function to plot user defined graph
    def openGraph(UserEquation, x_Start, x_End):
        #Store important details of graph in a list
        UserGraphs = [UserEquation, x_Start, x_End]
        #pushes this list to be stored onto the stack
        StoredEquations.push(UserGraphs)
        #Checks if any input fields are empty using error handingling
        try:
            #Utilising List Comprehension techniques to iterate over the ranges
            #and create the list of x values that will be plotted
            x_values = [x for x in range(int(x_Start), int(x_End))]
            #using list comprehension again to produce the y values using the evaluated equation
            #and taking the inputs of the graph equation from an iteration over the x_values
            plt.plot(x_values, [eval(UserEquation) for x in x_values])
            #Bringing the graph forward to be visible to the user
            plt.show()
        #if a field is empty, a ValueError is thrown and this deals with it with a popup
        except ValueError:
            #Uses error pop up window
            PoppingUp('Empty Field', 'Please fill in all the fields before submitting')
```

Code Screenshot 4.7 – Creating CalculusGraphs class and defining function to open the graph based on user input into GUI text fields

Code Screenshots 4.7 to 4.10 all lie within the CalculusGraphs Class

```
#Takes in GUI user input for creating the graph including equation and domain
def AcquireDetails(self):
    #Defining global variable for the equation of the graph as it will be used
    #in the other functions
    global UserEquation
    UserEquation = self.ids.graph_equation.text
    #Takes in starting and end points of the graph
    x_Start = self.ids.equation_x1.text
    x_End = self.ids.equation_x2.text
    #Runs openGraph function to plot and open graph with user input
    CalculusGraphs.openGraph(UserEquation, x_Start, x_End)

#using equation after being inputted through openGraph function to be used again
def f(x):
    #returns the evaluated form of the user input to allow numerical manipulations
    return eval(UserEquation)
```

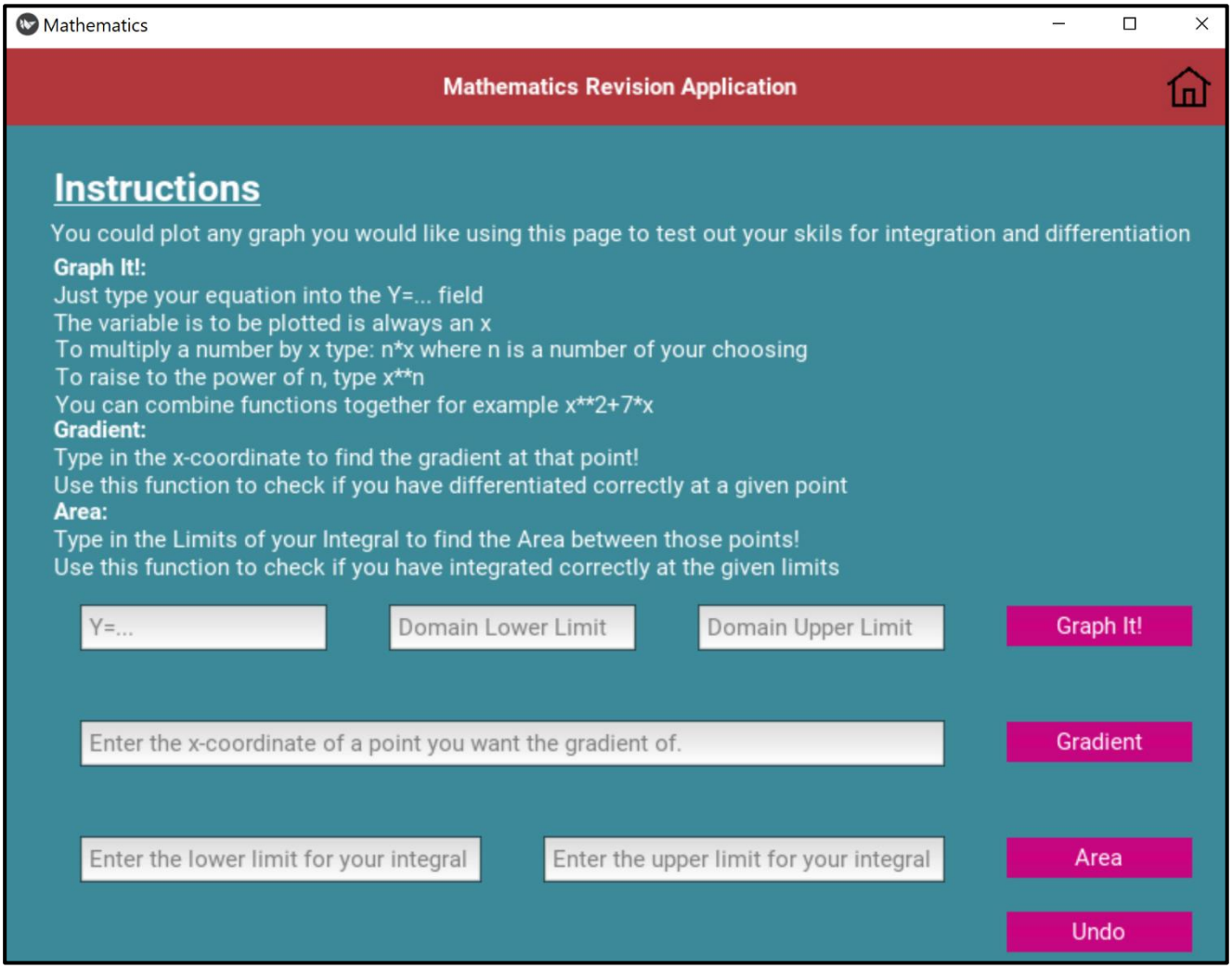
Code Screenshot 4.8 – functions to get user input from GUI to create the graph and function which stores the user defined equation

```
#Using the differentiation from first principles to calculate the gradient
#this nullifies the need to import a symbolic derivative calculator script which will
#be less efficient than my own defined function.
def FindGradient(self):
    #Takes in the point the user wants to find the gradient for
    GradientPoint = float(self.ids.gradient_point.text)
    #Mathematically known as delta x - small change in x (as dx tends to 0, derivative = gradient)
    dx = 1/10000
    #Change in the y function defined by the difference in the y values of x and
    #the minute change in x
    dy = CalculusGraphs.f(GradientPoint+dx) - CalculusGraphs.f(GradientPoint)
    #dy/dx is the Leibniz derivative notation which is well known in mathematics
    #Rounded to 2 decimal places for the user to make easier comparisons
    gradient = round(dy/dx, 2)
    #Outputs the gradient as a window popup
    PoppingUp('Gradient', 'Your requested gradient at the point x=%d is: %d' % (GradientPoint, gradient))

#Similar to above, this uses Integration from first principles to calculate the area
#under the graph between two points, and is more efficient than importing someone else's
#code as this is an important part of the Mathematics A Level that I know
def FindArea(self):
    #Converting user input into a float to calculate with
    Area_Start = float(self.ids.area_start.text)
    Area_End = float(self.ids.area_end.text)
    #This finds the range and divides by the number of rectangles that I choose to
    #I have chosen 10000 as this gives a very accurate answer to my chosen 2 decimal
    #places but does not slow down the program and make the user wait unnecessarily
    RecWidth = (float(Area_Start)-float(Area_End))/10000
    #Will use this as a cumulative area variable for the 'for loop'
    Area = 0
    #Loops over the number of rectangles
    for i in range(10000):
        #Finds the y values for the tiny changes in width
        RecHeight = CalculusGraphs.f(Area_Start + i * RecWidth)
        #Sums up all the areas of the rectangles under the graph to find total area
        Area += RecWidth * RecHeight
    PoppingUp('Area', 'Your requested Area under the curve between x=%d and x=%d is: %d' % (Area_Start, Area_End, Area))
```

Code Screenshot 4.9 – My differentiation and integration functions from first principles to efficiently calculate the gradient and area respectively

```
#Undo function utilising the stack
def Undo(self):
    #current equation is popped of stack
    StoredEquations.pop()
    #Checks if the stack is not empty after popping of list of details
    if not StoredEquations.isEmpty():
        #Pops of next list of details and stores it into a string
        UndoneGraph = StoredEquations.pop()
        #Storing the equation seperately so it can be used by the gradient and area functions
        #even after doing an undo
        UserEquation = UndoneGraph[0]
        #Calls openGraph function with order of details as arguments
        CalculusGraphs.openGraph(UserEquation, UndoneGraph[1],UndoneGraph[2])
    #If the stack is found to be empty, then the user cannot undo further
    else:
        #a pop up is used to notify the user
        PoppingUp('Empty!', 'There are no more equations to Undo!')
```

Code Screenshot 4.10 – The undo function, utilising the stack, allows a user to easily compare their graphs from previous inputs and generate a better understanding of how the area and gradients of a graph are affected by changing specific parameters



Code Screenshot 4.11 – Calculus Graphs Screen

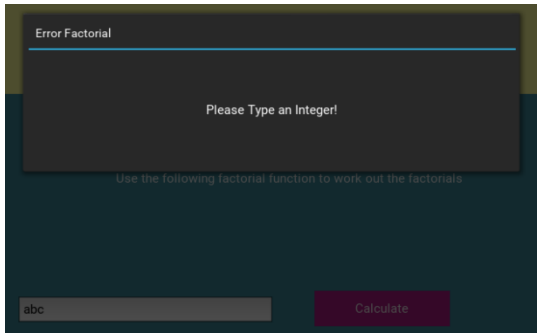
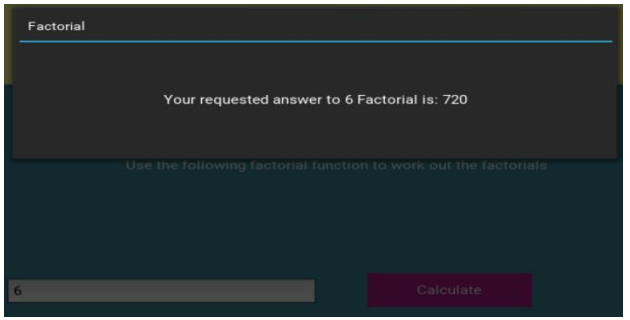
```
<CalculusGraphs>:
    id: calc2_screen
    MainWidget
    FloatLayout:
        HomeButton:
            pos: 743, 560
            on_release: root.manager.current = 'Home'
        Label:
            markup: True
            text: "[size=25][u][b]Instructions[/b][/u][/size]"
            pos: -300, 210
        Label:
            markup: True
            text: "You could plot any graph you would like using this page to test out your skils for integration and differentiation"
            pos: 0, 180
        Label:
            markup: True
            text: "[b]Graph It!: [/b]\nJust type your equation into the Y=... field\nThe variable is to be plotted is always an x"
            pos: -225, 140
        Label:
            text: "To multiply a number by x type: n*x where n is a number of your choosing\nTo raise to the power of n, type x**n\nYou can combine functions together for example x**2+7
            pos: -122, 87
        Label:
            markup: True
            text: "[b]Gradient: [/b] \nType in the x-coordinate to find the gradient at that point!\nUse this function to check if you have differentiated correctly at a given point"
            pos: -110, 35
        Label:
            markup: True
            text: "[b]Area: [/b] \nType in the Limits of your Integral to find the Area between those points!\nUse this function to check if you have integrated correctly at the given l
            pos: -112, -18
    FloatLayout:
        size: self.size
        Button:
            text:"Gradient"
            size_hint_x: 0.15
            size_hint_y: 0.01
            pos: 650,148
            background_normal: ''
            background_color: 0.8,0,0.5,0.8
            on_release: root.FindGradient()
```

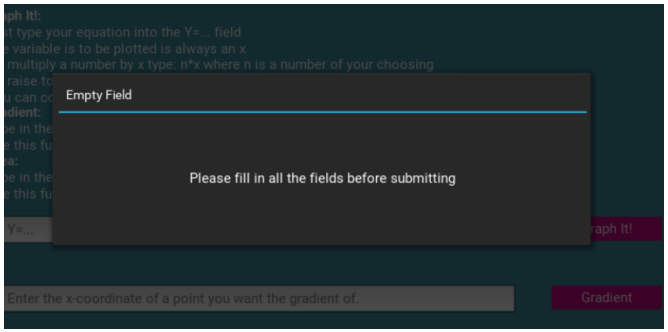
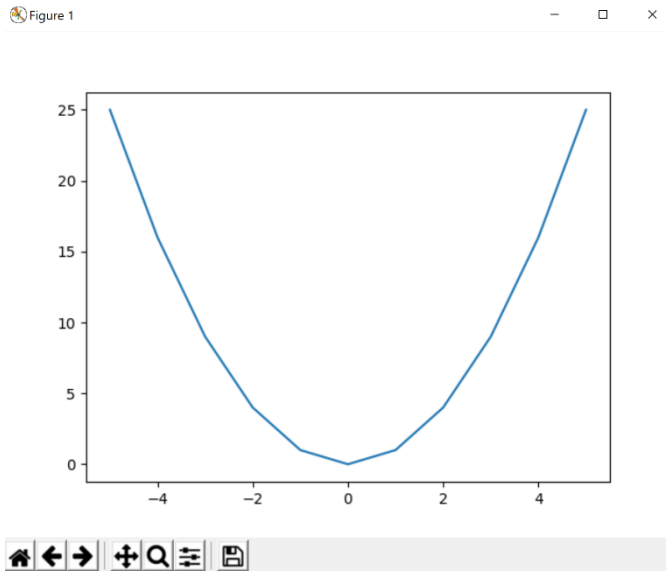
```
    Button:
        text:"Graph It!"
        size_hint_x: 0.15
        size_hint_y: 0.01
        pos: 650,223
        background_normal: ''
        background_color: 0.8,0,0.5,0.8
        on_release: root.AcquireDetails()
    TextInput:
        id: graph_equation
        hint_text: "Y=..."
        size_hint_x: 0.20
        size_hint_y: 0.05
        pos: 50, 210
    TextInput:
        id: equation_x1
        hint_text: "Domain Lower Limit"
        size_hint_x: 0.20
        size_hint_y: 0.05
        pos: 250, 210
    TextInput:
        id: equation_x2
        hint_text: "Domain Upper Limit"
        size_hint_x: 0.20
        size_hint_y: 0.05
        pos: 450, 210
    TextInput:
        id: gradient_point
        hint_text: "Enter the x-coordinate of a point you want the gradient of."
        size_hint_x: 0.7
        size_hint_y: 0.05
        pos: 50, 135
    TextInput:
        id: area_start
        hint_text: "Enter the lower limit for your integral"
        size_hint_x: 0.325
        size_hint_y: 0.05
        pos: 50, 60
    TextInput:
        id: area_end
        hint_text: "Enter the upper limit for your integral"
        size_hint_x: 0.325
        size_hint_y: 0.05
```

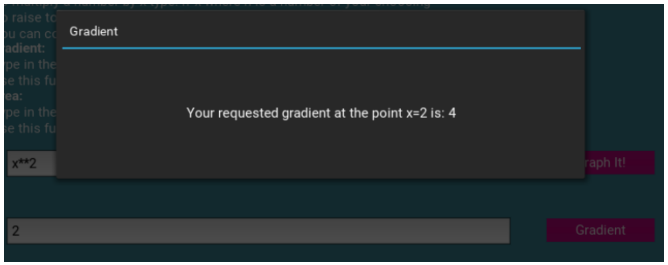
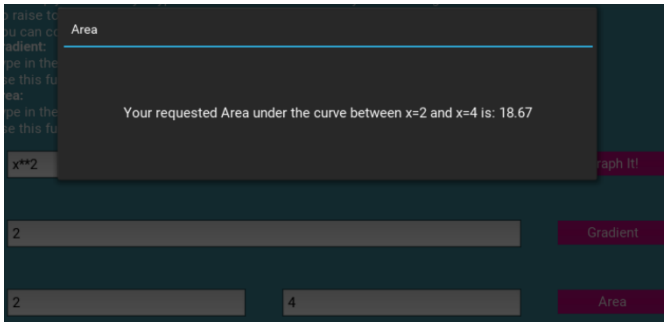
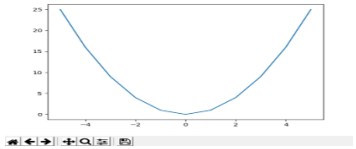
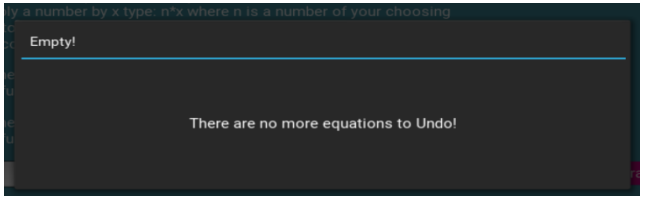
Code Screenshot 4.12 – Corresponding KV code for Calculus Graphs Screen

Alpha Testing – Calculus Revision Section

To confirm that my code has been successful in meeting the criteria of objective 7, I am completing an alpha test on the calculus section. Tests will also be completed on the other two revision sections following this part of the 'Revision Section'.

Test Reference	Section	Input	Expected Output	Actual Outcome	Comments
8a	Calculus Series	Type "**abc**" into 'Enter Factorial Field' then press **Calculate**	Pop up should tell the user to enter an integer		Works As Expected
8b	Calculus Series	Type "**6**" into 'Enter Factorial Field' then press **Calculate**	There should be a pop up which tells the user the answer of **6! = 720**		Works As Expected

8c	Calculus Graphs	Leave all fields blank, and press **Calculate**	Error Pop Up should tell the user to fill in all fields		Works As Expected
8d	Calculus Graphs	Type "**x**2**" into 'Enter Equation Field', "**-5**" in 'Lower Limit Domain Field' and "**5**" in 'Upper Limit Domain Field' then press **Calculate**	It should plot the curve $y = x^2$ which will be symmetrical in the line x=0 and be plotted between x = -5 and x = 5		Works As Expected

8e	Calculus Graphs	Enter "**2**" in 'Gradient Point' field then press **Gradient**	The gradient should be calculated from the x^2 function (from previous test) at x=2 and therefore pop up with an answer of **4**		Works As Expected
8f	Calculus Graphs	Enter "**2**" in 'Lower Limit Integral' field and "**4**" in 'Upper Limit Integral' field then press **Area**	The area should be calculated for the x^2 function between x=2 and x=4 and should give an answer of 18.67		Works As Expected
8g	Calculus Graphs	Do same inputs as **test 8d** then do same inputs as **test 8d** but using "**x**3**" then press **Undo**	The program should output the $y = x^2$ graph, then $y = x^3$ and when Undo is pressed, it will output the $y = x^2$ graph		Works As Expected
8h	Calculus Graphs	Press **Undo** twice again after **test 8g**	Pop up should tell the user that it can no longer undo, as there is nothing left on the stack		Works As Expected

Trigonometry

The trigonometry revision section will cover the understanding of using radians as an alternative angle measurement to degrees. A small explanation will be written on the screen. Most importantly, the user will have the ability to convert degrees into radians and vice versa using the functions that I have made. The user will also be able to provide three coordinates which will be used to plot a triangle. This will allow them to practice their skills in a visual way as well as with the numerical method provided.

```
#Defining Screen and functions for trigonometry revision
class Trigonometry(Screen, FloatLayout):
    #Creating function to plot the user defined triangle
    def PlotTriangle(self):
        #Taking in user coordinates to plot their triangle
        Co_One = self.ids.xy_1.text
        Co_Two = self.ids.xy_2.text
        Co_Three = self.ids.xy_3.text
        #Using error handling to prevent user from breaking program when typing incorrectly
        try:
            #Creating numpy array which stores the evaluated coordinates given by the user
            #numpy arrays are more optimised for this task which is why I decided to use it
            Tri = np.array([eval(Co_One), eval(Co_Two), eval(Co_Three)])
            #Creating a new figure which is going to be the triangle
            plt.figure()
            #Plotting triangle as a scatter graph with each point with size 50
            #using python number slicing to allow plot to be cyclic
            #this means the first point will act as starting and ending point to close the shape
            plt.scatter(Tri[:, 0], Tri[:, 1], s = 50, color = 'red')
            #Plotting the triangle and setting internal colour to be blue
            Triangle = plt.Polygon(Tri[:3,:], color='blue')
            plt.gca().add_patch(Triangle)
            #Bringing plot to front for user to see
            plt.show()
        #Prevents the two possible errors that can occur and pops up warning user
        except NameError or SyntaxError:
            PoppingUp('Co-ordinate Error', 'Please enter the co-ordinates in the correct format')
```

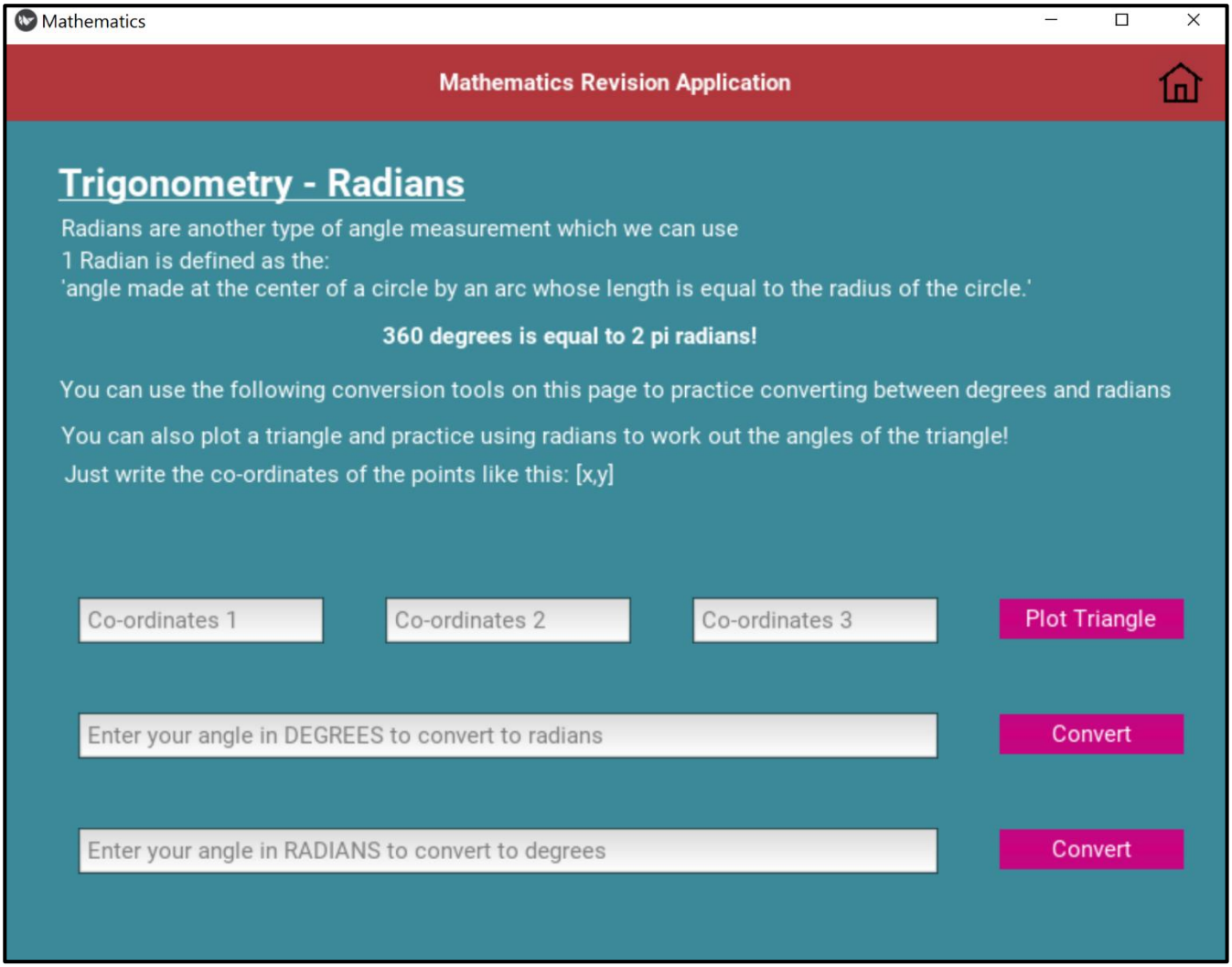
Code Screenshot 4.13 – Creating a new class for the trigonometry revision section and creating the function to plot the user defined triangle

```
#Creating function to convert degrees into radians
def ConvertDegrees(self):
    #Taking user input from text box
    user_degrees = self.ids.angle_degrees.text
    #Using error handling to ensure user inputs a number
    try:
        #converting the input into a float data type to be used in calculations
        user_degrees = float(user_degrees)
        #using the mathematical conversion to change degrees into radians
        convertedD = user_degrees *((math.pi)/180)
        #Outputting the answer as a pop up
        PoppingUp('Converted Radians', '%s degrees converted into radians is: %s' % (user_degrees, convertedD))
    #Checks for the ValueError
    except ValueError:
        #Pop up to notify user to input a number
        PoppingUp('Not Number', 'Please enter a number to be converted to radians!')
```

Code Screenshot 4.14 – My Degrees to Radians Conversion Function

```
#Creating function to convert radians into degrees
def ConvertRadians(self):
    #Taking user input from text box
    user_radians = self.ids.angle_radians.text
    #Using error handling code to make sure that the inputs a number
    try:
        #converting the input into a float data type to be used in calculations
        user_radians = float(user_radians)
        #using the inverse mathematical conversion to change radians into degrees
        convertedR = user_radians *(180/(math.pi))
        #Outputting the answer as a pop up
        PoppingUp('Converted Degrees', '%s radians converted into degrees is: %s' % (user_radians, convertedR))
    #Checks for a possible ValueError
    except ValueError:
        #A Pop up is used to notify the user to input a number
        PoppingUp('Not Number', 'Please enter a number to be converted to degrees!')
```

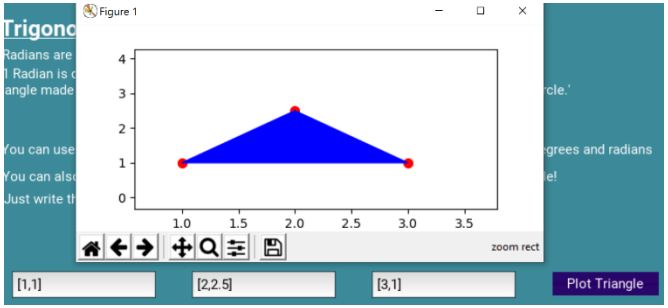
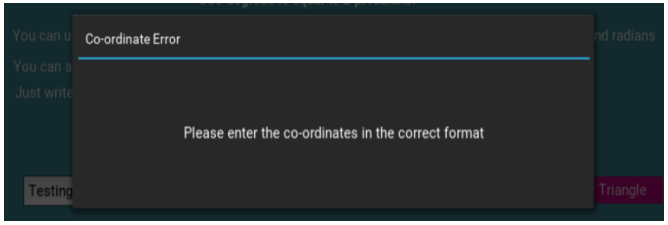
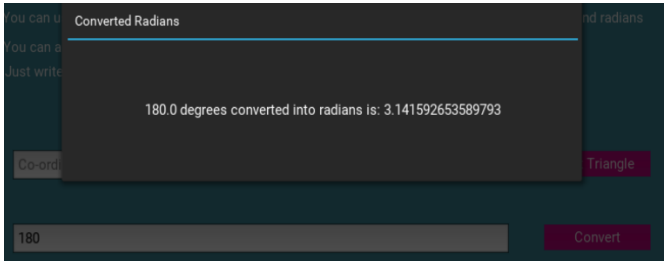
Code Screenshot 4.15 – My Radians to Degrees Conversion Function

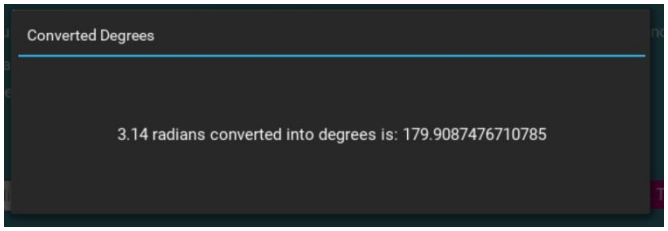
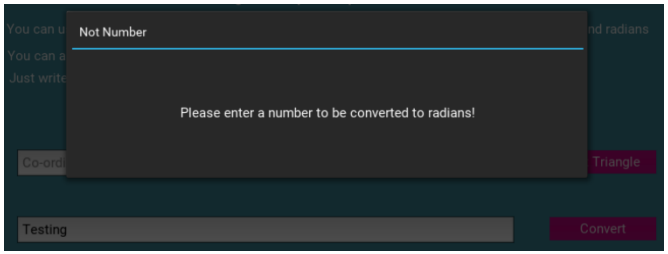


Code Screenshot 4.16 – Trigonometry Screen

Alpha Testing – Trigonometry Revision Section

These tests are taken from Test Table 8 from the design section.

Test Reference	Section	Input	Expected Output	Actual Outcome	Comments
8i	Trigonometry	Type "**[1,1]**" into 'Co-ordinates 1' Field Type "**[2,2.5]**" into 'Co-ordinates 2' Field Type "**[3,1]**" into 'Co-ordinates 3' Field then press **Plot Triangle**	The triangle defined by the user should be plotted and shown to the user		Works As Expected
8j	Trigonometry	Type "**Testing**" into 'Co-ordinates 1' Field Type "**Testing**" into 'Co-ordinates 2' Field Type "**Testing**" into 'Co-ordinates 3' Field then press **Plot Triangle**	As these are not coordinates, there should be a pop up which notifies the user to enter the coordinates in the given format		Works As Expected
8k	Trigonometry	Type "**180**" into 'Degrees' Field then press **Convert**	This should output the first few digits of pi as 180 degrees = pi		Works As Expected

8l	Trigonometry	Type "**3.14**" into 'Radians' Field then press **Convert**	I have typed an approximated value of pi so an answer approximately equal to 180 should be outputted.		Works As Expected
8m	Trigonometry	Type "**Testing**" into 'Radians' Field then press **Convert**	This is not a number, and therefore a pop up will inform the user to type in a number to convert to degrees.		Works As Expected

All tests have been successful for the trigonometry section, and the code is working as it is expected to.

Statistics Revision

The next and final topic that I will need to complete interactive material for is Statistics. This will allow me to complete the requirements of objective 7 fully. In this section, I will create functions which generate a random set of data that users can do statistical analysis on within the requirements of the Mathematics A Level. I will also create functions which allow the user to carry out these statistical calculations including a standard deviation function, variance and mean. To link statistical analysis of a data set to the probability of events occurring such as a specific value, I will also create functions for some probability distributions. These will be the Binomial Probability Distribution, Binomial Cumulative Distribution and the Normal Probability Distribution.

Both binomial distributions will be based entirely on the user's input of the required parameters. The Normal distribution function will utilise the already created mean and variance functions and apply the Central Limit Theorem to the generated data set of numbers. The user will then be able to test their knowledge by calculating the probability of a specific value being in the data set. They will be able to use my function to check their answer. This will provide excellent practice for the student as there are endless practice questions for them with the randomly generated data set.

```
class StatScreen(Screen, FloatLayout):
    #Creating a new attribute with a string property for this class so
    #there can be live updates to it when the user generates new set
    data_set = StringProperty()
    #using a constructor method with variable keyword arguments
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        #setting the data_set variable as an attribute of the objects in this class
        data_set = str()

    #Defining function to generate random set of ten numbers to be used in calculations
    def GenerateSet(self):
        #Creates new list
        random_set = []
        #Creates a loop to append 10 random numbers to the list
        for i in range (10):
            random_set.append(random.randint(0,100))
        #Sets the data_set variable as the random_set to update the screen live
        self.data_set = str(random_set)

    #Creating a function for the average of a list as this will be useful for
    #the other statistical functions
    def Average(self, numlist):
        total = 0
        #Adding up all the numbers
        for i in range(len(numlist)):
            total += numlist[i]
        #Dividing by the length of the list
        return total/len(numlist)
```

Code Screenshot 4.17 – Class for StatScreen, with init method, the function to randomly generate the data set and update it live on the screen and finally the Average function which will calculate the average of the list of numbers.

```
#Defining function to calculate the variance of the given set of numbers
def Variance(self, numlist):
    #Setting average of list using defined function
    average = self.Average(numlist)
    #setting variance to 0
    variance = 0
    #creating a for loop to iterate over the numbers in the list
    for i in range(len(numlist)):
        #Adding a cumulative sum of the difference between each value and the average
        #then squaring that
        variance += (numlist[i] - average)**2
    #the variance is the above calculation divided by the amount of numbers
    return variance/len(numlist)

#Function for when StatCalc Button is clicked
def StatCalc(self):
    #assigning the evaluated list of numbers to a variable
    DataList = eval(self.data_set)
    #Calculating the mean of the data set
    mean = self.Average(DataList)
    #Calculating the variance of the data set
    variance = round(self.Variance(DataList), 3)
    #for efficiency, rather than creating a seperate function for the standard deviation
    #I can use the fact that it is the square root of the value of the variance
    #This means the program can be more efficient as there is less redundant code
    deviation = round(math.sqrt(variance), 3)
    #A pop up provides all these statistical calculations for the user
    PoppingUp('Statistic Calculations', "Mean = %s, Standard Deviation = %s, Variance = %s" %(mean, deviation, variance))
```

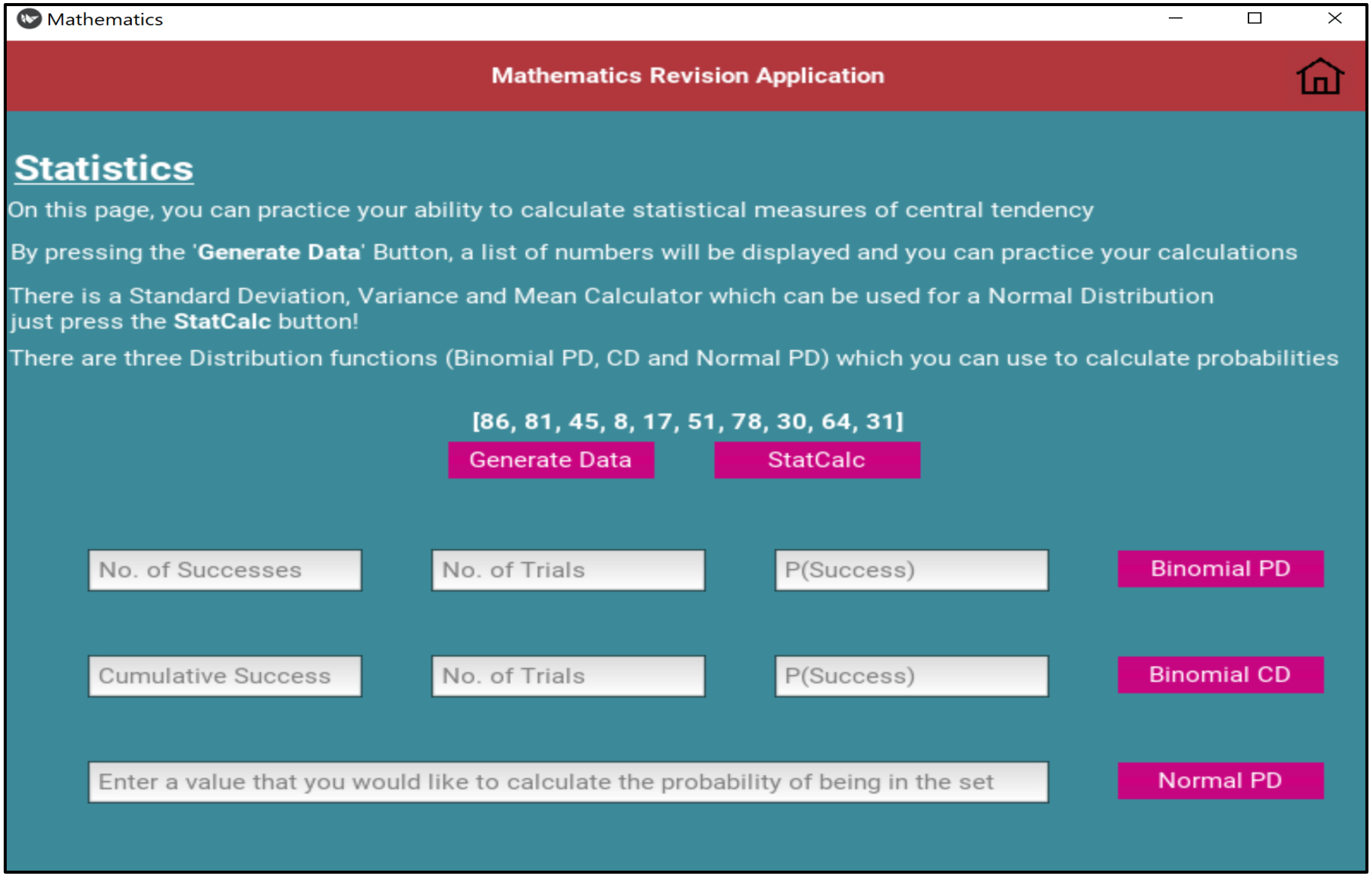
Code Screenshot 4.18 – Variance Function and Statistics Calculation Functions to output results to user

```
#Defining Binomial Probability Distribution Function
def BinomialDistribution(self, trialx, totaltrials, probability):
    #Using error handling to prevent user from making incorrect inputs
    try:
        #checking if probability is between 0 and 1, and that the trials given is below
        #the total trials that are given
        if probability > 0 and probability <=1 and trialx <= totaltrials:
            #Finding the binomial coefficient which uses combinatorics (N choose x)
            #This uses my recursive factorial function to calculate the required values
            BiCo1 = CalculusSeries.Factorial(totaltrials)
            BiCo2 = CalculusSeries.Factorial(trialx)
            BiCo3 = CalculusSeries.Factorial(totaltrials - trialx)
            #Working out the binomial coefficient from its calculated parts
            BiCoefficient = BiCo1/(BiCo2*BiCo3)
            #Calculating the probability of the 'successes' occuring
            p_successes = probability ** trialx
            #Calculating the probability of the remaining trials being 'failures'
            p_failures = (1-probability) ** (totaltrials - trialx)
            #Returning the binomial distribution generated probability
            return BiCoefficient * p_successes * p_failures
        else:
            #Notifying user if their inputs cannot be processed using the distribution
            #as they can not be used within the binomial distribution
            PoppingUp('Numbers', 'Values cannot be processed, check them carefully')
    #Notifying user that they have to check they have inputted the correct values
    except ValueError:
        PoppingUp('Numbers', 'Please check that you have inputted your values correctly')

#defining function to be used with the GUI and distribution button
def BinomialInterface(self):
    #assigning the user inputs to variables
    trialx = int(self.ids.binomial_x.text)
    totaltrials = int(self.ids.binomial_N.text)
    probability = float(self.ids.binomial_p.text)
    #Calculating the binomial probability from the distribution function
    BinomialProbability = self.BinomialDistribution(trialx, totaltrials, probability)
    #Displaying the calculated probability to the user
    PoppingUp('Binomial Distribution', 'Your requested probability is: %s' %(BinomialProbability))
```

Code Screenshot 4.19 – Binomial Distribution function using my factorial function and the accompanying Binomial Interface function to output the results of processed data to the user

```
#Defining Cumulative Binomial Probability Distribution Function
def CumulativeBinomial(self):
    #assigning the user inputs to variables
    trialx = int(self.ids.Cbinomial_x.text)
    totaltrials = int(self.ids.Cbinomial_N.text)
    probability = float(self.ids.Cbinomial_p.text)
    #setting cumulative probability to 0
    CumulativeProbability = 0
    #Adding up all probabilities for trials occuring from 0 up to the given number
    for i in range(0, trialx + 1):
        #running the binomial distribution function over all values in range and summing
        CumulativeProbability += self.BinomialDistribution(i, totaltrials, probability)
    #returning the cumulative probability
    PoppingUp('Binomial Cumulative Distribution', 'Your requested probability is: %s' %(CumulativeProbability))

#Using the central limit theorem, this will calculate the probability of the
#generated set containing a given value by the user
def NormalDistribution(self):
    #assigning the evaluated list of numbers to a variable
    DataList = eval(self.data_set)
    #Calculating the mean of the data set
    mean = self.Average(DataList)
    #Calculating the variance of the data set
    deviation = math.sqrt(self.Variance(DataList))
    #Using error handling to catch any invalid inputs by the user
    try:
        #assigning user input to variable and converting into a float data type
        normalx = float(self.ids.normal_x.text)
        #Calculating the Normal Coefficient (coefficient of the exponential in equation)
        NCoefficient = 1/(deviation*math.sqrt(2*math.pi))
        #Calculating both parts of the exponential's power
        power1 = -1 * (normalx - mean)**2
        power2 = 2*(deviation**2)
        #Calculating final probability from distribution using Euler's number e
        NormalProbability = NCoefficient * ((math.e)**(power1/power2))
        #Outputs the final probability to the user
        PoppingUp('Normal Distribution', 'Your requested probability is: %s' %(NormalProbability))
    #Checks for Value Error and informs user when error is caught
    except ValueError:
        PoppingUp('Numbers', 'Please check that you have inputted your value correctly')
```

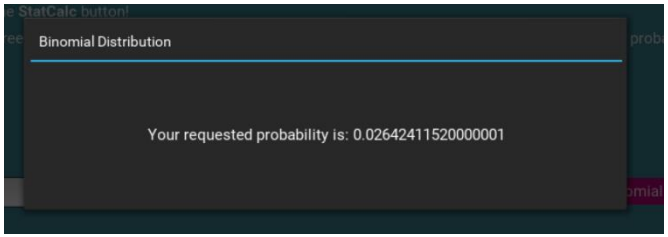
Code Screenshot 4.20 – Cumulative Binomial Distribution function and Normal Distribution function

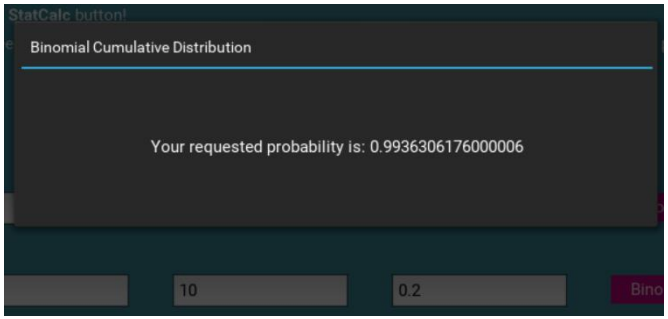
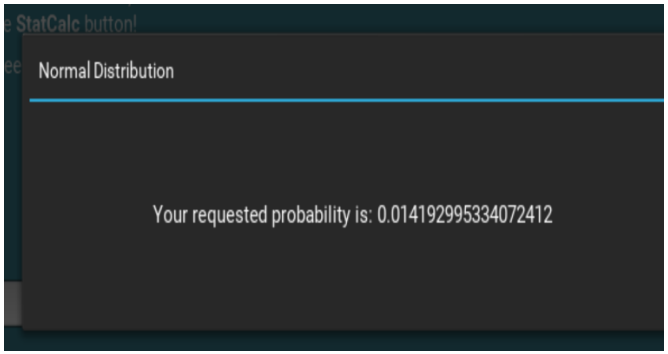
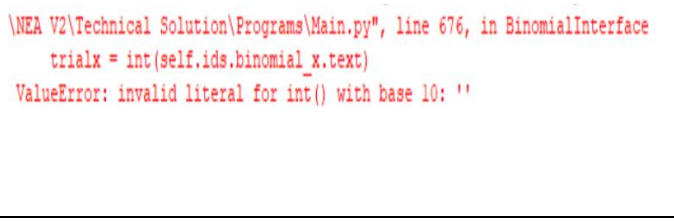


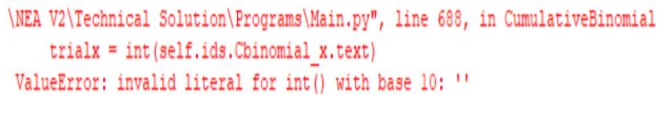
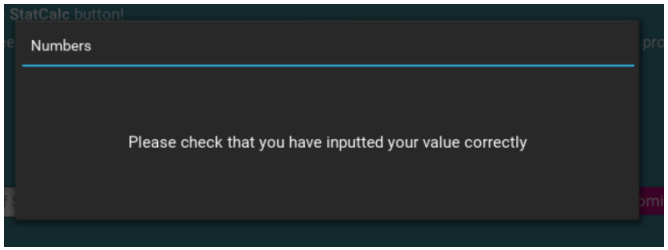
Code Screenshot 4.21 – Statistics Revision Screen

Alpha Testing – Statistics Revision Section

These tests were taken from test table 8.

Test Reference	Section	Input	Expected Output	Actual Outcome	Comments
8n	Statistics	Press **Generate Data** button	A set of 10 random integers should be displayed on the screen for the user to use in calculations.		Works As Expected
8o	Statistics	Press the **StatCalc** button	A popup displaying the mean, variance and standard deviation of the set of data should be displayed.		Works As Expected
8p	Statistics	Type "**5**" into 'No. of Successes' Field Type "**10**" into 'Number of Trials' Field Type "**0.2**" into 'P(Success)' Field then press **Binomial PD**	This should calculate and display **P(X = 5)** where X is the random variable that is binomially distributed with n = 10 and p = 0.2. From my calculations, this probability should be around 0.02642		Works As Expected

8q	Statistics	Type "**5**" into 'No. of Successes' Field Type "**10**" into 'Number of Trials' Field Type "**0.2**" into 'P(Success)' Field then press **Binomial CD**	This should calculate and display **P(X <= 5)** where X is the random variable that is binomially distributed with n = 10 and p = 0.2. From my calculations, this probability should be around 0.99363		Works As Expected
8r	Statistics	Type "**38**" into 'Normal Distribution' Field and then press **Normal PD** Random Set On Screen while carrying out test: [47, 80, 81, 59, 79 14, 31, 13, 49, 12]	This will use my algorithm for the normal distribution to calculate to the probability of 38 being in the randomly generated set – applying the central limit theorem. From my own calculations, the probability (from the set on screen at the time) should be around 0.01419		Works As Expected
8s	Statistics	Leave all fields empty and press **Binomial PD**	This should notify the user to enter a value in the required fields		Not Working as intended – See amendment section

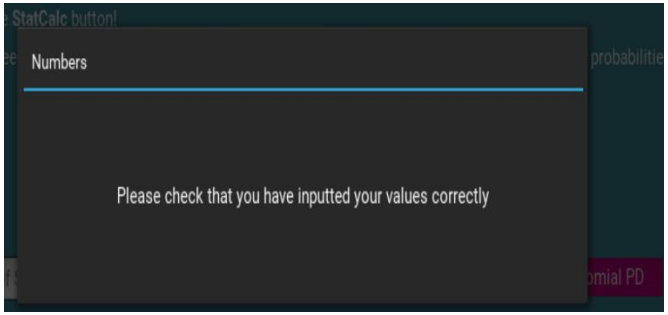
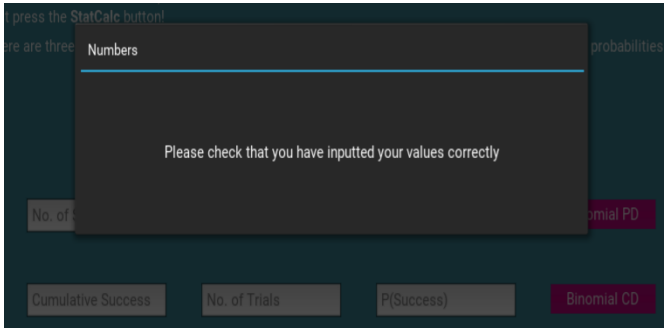
8t	Statistics	Leave all fields empty and press **Binomial CD**	This should notify the user to enter a value in the required fields		Not Working as intended – See amendment section
8u	Statistics	Leave all fields empty and press **Normal PD**	This should notify the user to enter a value in the required field		Works As Expected

Amendments

From my testing, I had realised that I didn't include exception handling for my Binomial Probability and Cumulative Distribution functions. I have now added them in and carried out the tests again which were successful.

```
#defining function to be used with the GUI and distribution button
def BinomialInterface(self):
    #Adding Error Handling
    try:
        #assigning the user inputs to variables
        trialx = int(self.ids.binomial_x.text)
        totaltrials = int(self.ids.binomial_N.text)
        probability = float(self.ids.binomial_p.text)
        #Calculating the binomial probability from the distribution function
        BinomialProbability = self.BinomialDistribution(trialx, totaltrials, probability)
        #Displaying the calculated probability to the user
        PoppingUp('Binomial Distribution', 'Your requested probability is: %s' %(BinomialProbability))
    #Notifying user that they have to check they have inputted the correct values
    except ValueError:
        PoppingUp('Numbers', 'Please check that you have inputted your values correctly')


#Defining Cumulative Binomial Probability Distribution Function
def CumulativeBinomial(self):
    #Adding Error Handling
    try:
        #assigning the user inputs to variables
        trialx = int(self.ids.Cbinomial_x.text)
        totaltrials = int(self.ids.Cbinomial_N.text)
        probability = float(self.ids.Cbinomial_p.text)
        #setting cumulative probability to 0
        CumulativeProbability = 0
        #Adding up all probabilities for trials occuring from 0 up to the given number
        for i in range(0, trialx + 1):
            #running the binomial distribution function over all values in range and summing
            CumulativeProbability += self.BinomialDistribution(i, totaltrials, probability)
        #returning the cumulative probability
        PoppingUp('Binomial Cumulative Distribution', 'Your requested probability is: %s' %(CumulativeProbability))
    #Notifying user that they have to check they have inputted the correct values
    except ValueError:
        PoppingUp('Numbers', 'Please check that you have inputted your values correctly')
```

Code Screenshot 4.22 – Amended code for Binomial Distribution functions, used in tests 8s and 8t

8s	Statistics	Leave all fields empty and press **Binomial PD**	This should notify the user to enter a value in the required fields		Works As Expected
8t	Statistics	Leave all fields empty and press **Binomial CD**	This should notify the user to enter a value in the required fields		Works As Expected

Mid-Technical Solution Client Feedback

As I am *about* halfway through the technical solution, I decided to send the document to my client to ensure that my work thus far matches their expectations. His reply is shown below:

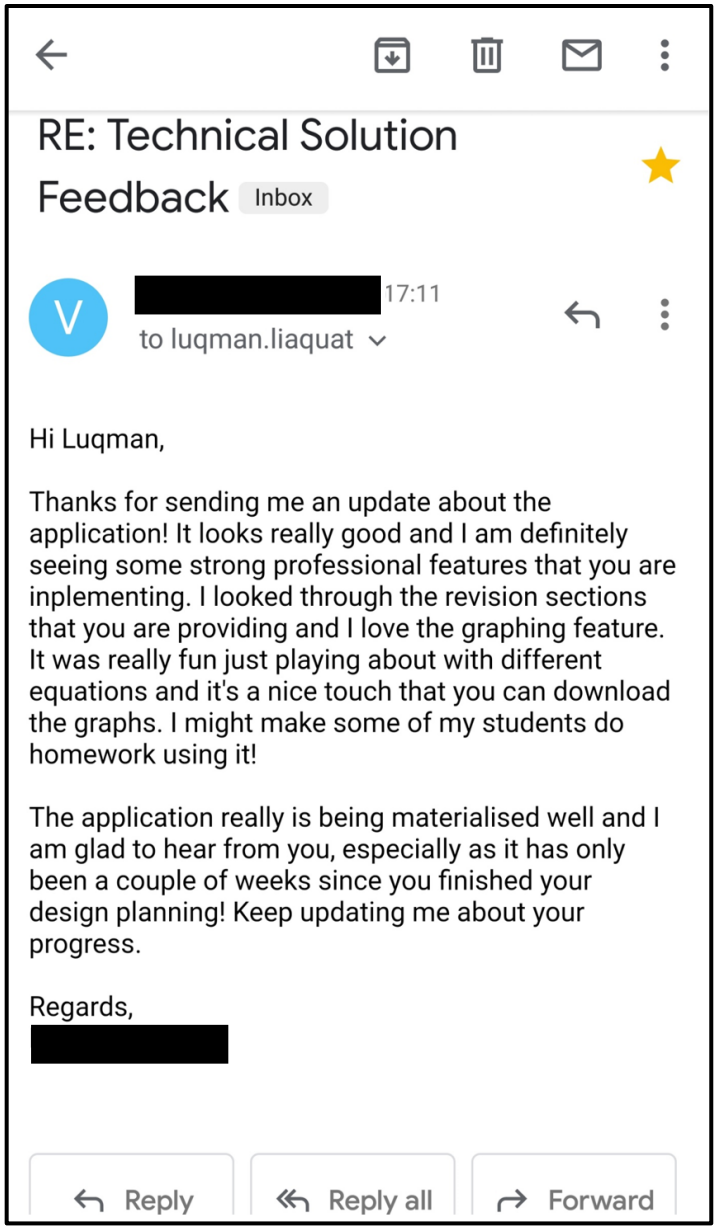


Figure 9 – Feedback on how I am doing so far in the technical solution from my client.

My client likes what I have done so far which means that I can continue working on the technical solution. I will ask for further feedback from my client and the end users (students) when I have completed the application. This will be the beta testing of the application.

Maths Topic Test

This section is for the maths testing aspect of the application. The user will be able to choose which topics they want to get tested on and then answer a series of questions on them. These questions will be taken from the database and will continuously change. It is important to note that there are currently only a few questions and as this project develops, more will be added.

```
#Topic test screen for users to practice their maths
class TopicTestScreen(Screen):
    #function to return the topics needed
    def WhichTopics(self):
        #Create a new list to check which topics will be in the test
        chosen_topics = []
        #Storing the status of each of the checkboxes in variables to be used
        CheckTrig = self.ids.trig_check.active
        CheckCalc = self.ids.calculus_check.active
        CheckStats = self.ids.stat_check.active
        #using boolean conditions to check status of each
        if not CheckTrig and not CheckCalc and not CheckStats:
            #if no check box is pressed then False is returned
            return False
        else:
        #checking through each checkbox to see if it is activated
            if CheckTrig:
                #if it is activated it will append the ids of the questions to the chosen topics list
                crsr.execute('''SELECT QuestionID FROM Topic WHERE QuestionID BETWEEN 500 AND 599''')
                for trig in crsr.fetchall():
                    chosen_topics.append(trig[0])
            #Same for the next two checkboxes
            if CheckCalc:
                crsr.execute('''SELECT QuestionID FROM Topic WHERE QuestionID BETWEEN 100 AND 299''')
                for calc in crsr.fetchall():
                    chosen_topics.append(calc[0])
            if CheckStats:
                crsr.execute('''SELECT QuestionID FROM Topic WHERE QuestionID BETWEEN 300 AND 399''')
                for stat in crsr.fetchall():
                    chosen_topics.append(stat[0])
            #The list of question ids are returned
            return chosen_topics
```

Code Screenshot 5.2 – Function to find which topics the user wants to use through checkboxes

```
    #Function to automatically choose the question based on user's topic choices
    def WhichQuestion(self):
            #Connects to Database
            DataConnect('connect')
            #Stores the topics taken from the WhichTopics Function
            QuestionsToChoose = TopicTestScreen.WhichTopics(self)
            #Chooses a random id from the list of questions that were taken
            NewQuestionID = random.choice(QuestionsToChoose)
            #Finds the associated question with that ID from database
            crsr.execute('''SELECT Question FROM Topic WHERE QuestionID=?''',(NewQuestionID,))
            #Stores that question in the MainQuestion variable
            for Ques in crsr.fetchall():
                MainQuestion = Ques[0]
            #Finds the associated answer with that ID from the database
            crsr.execute('''SELECT Answer FROM Topic WHERE QuestionID=?''',(NewQuestionID,))
            #Stores that question in the MainQuestion variable
            for Ans in crsr.fetchall():
                MainAnswer = Ans[0]
            #Disconnects from the database
            DataConnect('disconnect')
            #Returns the Question and Answer
            return MainQuestion, MainAnswer
```

Code Screenshot 5.2 – Function to work out which question should be used

```
#Function to obtain number of correct values and total values for each topic
#with incrementing total value to show a new question was attempted
def ProgressObtain(self):
    #Connecting to database
    DataConnect('Connect')
    #Assgining global variable from different function to a
    #local variable to prevent any side effects from occuring
    QuestionID = NewQuestionID
    #Checks if the question id which matches the differentiation ids
    if QuestionID >= 100 and QuestionID <=199:
        #Uses an sql query to find the total and correct values for differentiation topic
        crsr.execute('''SELECT DiffCorrect, DiffTotal FROM Progress WHERE Email =?''', (UserCheck,))
        #Assigns the integer values of the query return to variables
        for row in crsr.fetchall():
            DiffCorrect = int(row[0])
            DiffTotal = int(row[1])
        #Increments the total answered questions for topic by 1
        DiffTotal+=1
        return QuestionID, DiffCorrect, DiffTotal
    #Similar sets of code as above for other topics in program
    elif QuestionID >= 200 and QuestionID <=299:
        #Uses an sql query to find the total and correct values for the integration topic
        crsr.execute('''SELECT IntCorrect, IntTotal FROM Progress WHERE Email =?''', (UserCheck,))
        #Assigns the integer values of the query return to variables
        for row in crsr.fetchall():
            IntCorrect = int(row[0])
            IntTotal = int(row[1])
        #Increments the total answered questions for integration topic by 1
        IntTotal+=1
        return QuestionID, IntCorrect, IntTotal
    #data obtained for statistics topic
    elif QuestionID >= 300 and QuestionID <=399:
        #Uses an sql query to find the total and correct values for the Statistics topic
        crsr.execute('''SELECT StatCorrect, StatTotal FROM Progress WHERE Email =?''', (UserCheck,))
        #Assigns the integer values of the query return to variables
        for row in crsr.fetchall():
            StatCorrect = int(row[0])
            StatTotal = int(row[1])
        #Increments the total answered questions for tatistics topic by 1
        StatTotal+=1
        return QuestionID, StatCorrect, StatTotal
    #Code for Trigonometry topic
    elif QuestionID >= 500 and QuestionID <=599:
        #Uses an sql query to find the total and correct values for the integration topic
        crsr.execute('''SELECT TrigCorrect, TrigTotal FROM Progress WHERE Email =?''', (UserCheck,))
        #Assigns the integer values of the query return to variables
        for row in crsr.fetchall():
            TrigCorrect = int(row[0])
            TrigTotal = int(row[1])
        #Increments the total answered questions for integration topic by 1
        IntTotal+=1
        return QuestionID, TrigCorrect, TrigTotal
    #Disconnecting from the database
    DataConnect('disconnect')
```

Code Screenshot 5.3 – Progress Obtain function to obtain progress data for specific topic and make updates depending on the topic, such as incrementing total. These are returned to be used in other functions

```
#Function to update database and questions accordingly
#if the user's answer was correct
def ProgressUpdate(self, Outcome):
    #Connecting to database
    DataConnect('connect')
    #Assigning values acquired from progress obtain function
    QuestionID, Correct, Total = self.ProgressObtain()
    #if outcome was correct the Correct Variable is incremented by 1
    #otherwise it remains the same, this allows me to use one function to
    #carry out the tasks for updating the datbase when a correct or incorrect
    #answer is given
    if Outcome == 'Success':
        Correct+=1
    #Checks ids match differentiation topic ids
    if QuestionID >= 100 and QuestionID <=199:
        #Updating database to reflect these changes for that user
        #For the differentiation topic
        crsr.execute('''INSERT INTO DiffCorrect, DiffTotal
                    VALUES(?,?) WHERE Email=?''', ((Correct,), (Total,), (UserCheck,)))
    #Checking ids for integration topic
    elif QuestionID >= 200 and QuestionID <=299:
        #Updating database to reflect these changes for that user
        #For the Integration topic
        crsr.execute('''INSERT INTO IntCorrect, IntTotal
                    VALUES(?,?) WHERE Email=?''', ((Correct,), (Total,), (UserCheck,)))
    #Checking ids for Statistics topic
    elif QuestionID >= 300 and QuestionID <=399:
        #Updating database to reflect these changes for that user
        #For the Statistics topic
        crsr.execute('''INSERT INTO StatCorrect, StatTotal
                    VALUES(?,?) WHERE Email=?''', ((Correct,), (Total,), (UserCheck,)))
    #Checking ids for Trigonometry topic
    elif QuestionID >= 500 and QuestionID <=599:
        #Updating database to reflect these changes for that user
        #For the Trigonometry topic
        crsr.execute('''INSERT INTO TrigCorrect, TrigTotal
                    VALUES(?,?) WHERE Email=?''', ((Correct,), (Total,), (UserCheck,)))
    #Disconnecting from database and committing any commands that have been used
    DataConnect('disconnect')
```

Code Screenshot 5.4 – Function to make updates in database depending on the outcome of a user input in the test

```
#Creating a method to check the user's answer
def CheckAnswer(self):
    #First stores the user input in a variable
    CheckingAnswer = txt1.text
    #Connects to the database
    DataConnect('connect')
    #The answers are stored as a string with a list of answers inside
    #the eval converts it into a list and the answer can be searched through
    #That way many possible variations of the answer can be checked and it is NOT a
    #multiple choice test
    if CheckingAnswer in eval(MainA):
        #Stops the test to show user the result of their answer
        QuestionPopup.dismiss()
        #stops for 2 seconds
        time.sleep(2)
        #Pops up with correct answer statement
        PoppingUp('Answer', 'Answer correct!')
        #Running code to update database on correct answer
        self.ProgressUpdate('Success')

    else:
        #Similar to code above except it pops up with incorrect answer statement
        QuestionPopup.dismiss()
        #Stops for 2 seconds for question popup to be fully dismissed
        time.sleep(2)
        #Tells user their answer was incorrect
        PoppingUp('Answer', 'Answer Incorrect!')
        #Updates database for wrong answer
        self.ProgressUpdate('Fail')
```

Code Screenshot 5.5 – Function to check if the user's answer is correct and act accordingly

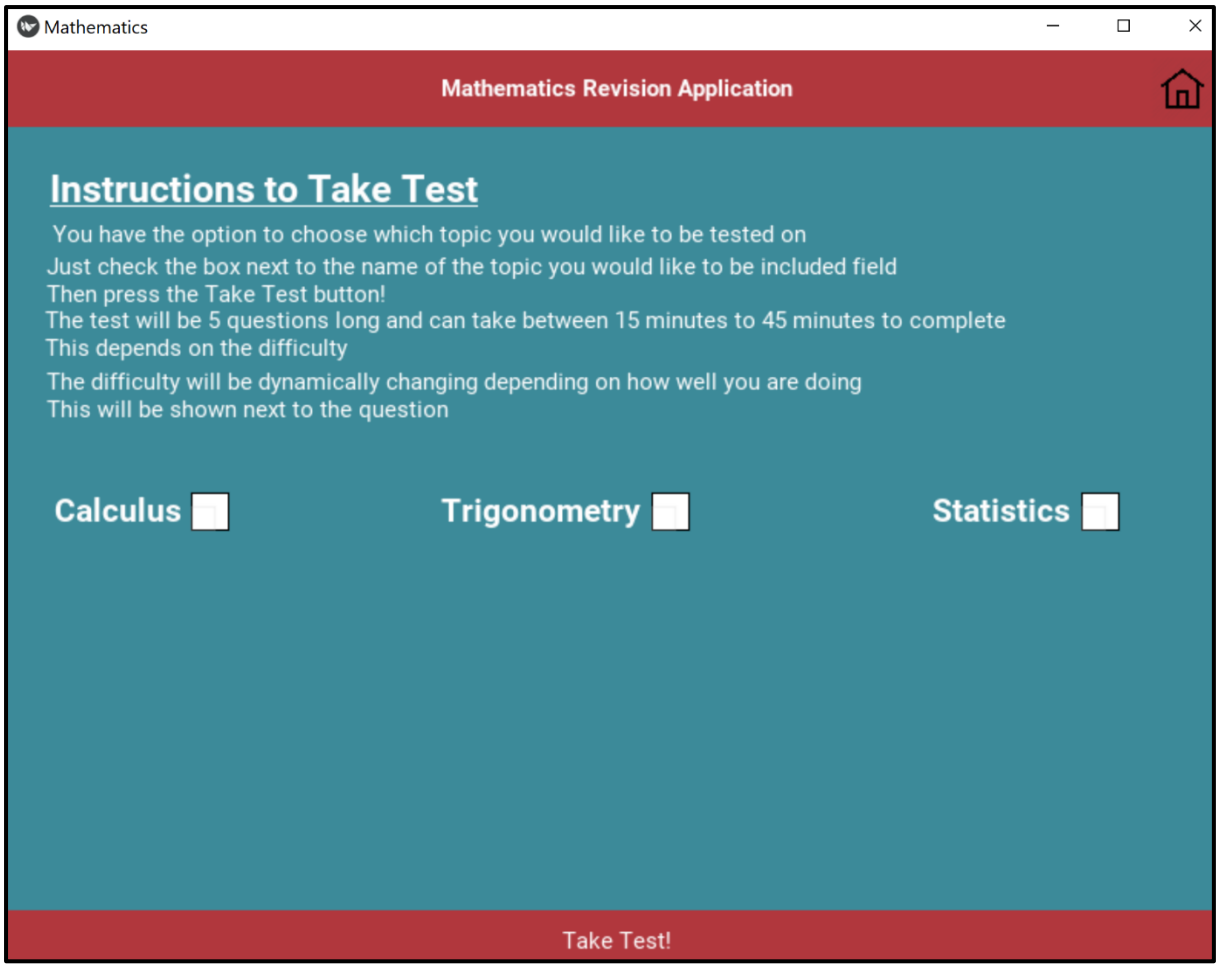
```
#Function to open up test for user with inputs and screen widgets
def StoreQA(self):
    #setting questions and answer as global class variables to be used with any method in this class
    global MainQ
    global MainA
    #Uses WhichQuestion function and stores the output question and answer in the two variables
    MainQ, MainA = TopicTestScreen.WhichQuestion(self)

    #Creating the widgets required for the test
    box = BoxLayout(orientation = 'vertical', padding = (10))
    #This is the question being outputted, taken from the database
    box.add_widget(Label(text = MainQ))
    #Creating the button to submit
    btn1 = kb.Button(text = 'Submit', background_color =(1,1,1,1), pos = (700,20), size_hint = (.1,.05))
    #Creating a global class variable which stores the user input for the question
    global txt1
    txt1 = TextInput(hint_text = 'Enter Your Answer', pos = (150,20), size_hint = (0.5, 0.08))
    #adding all of the widgets to the box layout
    box.add_widget(btn1)
    box.add_widget(txt1)
    #Setting up the Test using a popup without ability to dismiss by clicks
    global QuestionPopup
    QuestionPopup = Popup(title='Topic Test', title_size= (30),
                  title_align = 'center', content = box,
                  auto_dismiss = False)
    #Setting button bind to CheckAnswer method created already
    btn1.bind(on_press = TopicTestScreen.CheckAnswer)
    #Opens this popup on the screen
    QuestionPopup.open()
```

Code Screenshot 5.6 – Creating Test Screen and creating labels and buttons which map to my defined functions and variables

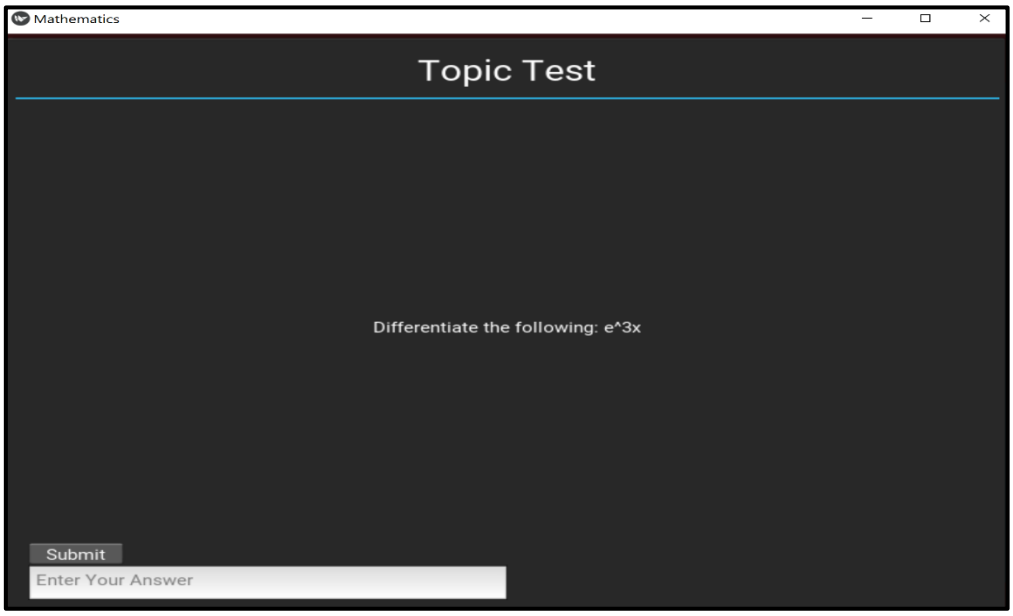
```
<TopicTestScreen>:
    id: topictest_screen
    MainWidget
    FloatLayout:
        HomeButton:
            pos: 743, 560
            on_release: root.manager.current = 'Home'
        Label:
            markup: True
            text: "[size=25][u][b]Instructions to Take Test[/b][/u][/size]"
            pos: -230, 210
        Label:
            markup: True
            text: "You have the option to choose which topic you would like to be tested on"
            pos: -122, 180
        Label:
            markup: True
            text: "Just check the box next to the name of the topic you would like to be included field\nThen press the Take Test button!"
            pos: -95, 150
        Label:
            markup: True
            text: "The test will be 5 questions long and can take between 15 minutes to 45 minutes to complete\nThis depends on the difficulty"
            pos: -60, 115
        Label:
            markup: True
            text: "The difficulty will be dynamically changing depending on how well you are doing\nThis will be shown next to the question"
            pos: -106, 75
    FloatLayout:
        CheckBox:
            id: calculus_check
            pos:-270,-5
            canvas.before:
                Color:
                    rgb: 0,0,0
                Rectangle:
                    pos:self.center_x-8, self.center_y-8
                    size:[25,25]
                Color:
                    rgb: 1,1,1
                Rectangle:
                    pos:self.center_x-7, self.center_y-7
```

```
                Color:
                    rgb: 0,0,0
                Rectangle:
                    pos:self.center_x-8, self.center_y-8
                    size:[25,25]
                Color:
                    rgb: 1,1,1
                Rectangle:
                    pos:self.center_x-7, self.center_y-7
                    size:[23,23]

        Label:
            markup:True
            text: "[size=21][b]Trigonometry[/b][/size]"
            pos: -50, 0

        CheckBox:
            id: stat_check
            pos:310,-5
            canvas.before:
                Color:
                    rgb: 0,0,0
                Rectangle:
                    pos:self.center_x-8, self.center_y-8
                    size:[25,25]
                Color:
                    rgb: 1,1,1
                Rectangle:
                    pos:self.center_x-7, self.center_y-7
                    size:[23,23]

        Label:
            markup:True
            text: "[size=21][b]Statistics[/b][/size]"
            pos: 250, 0


        Button:
            text: "Take Test!"
            size_hint_y: None
            padding: 0,25
            height: 40
            background_color: (1, 0, 0, 0.6)
            background_normal: ''
            on_release: root.StoreQA()
```

Code Screenshot 5.7 – Corresponding kv code, also changed the theme for my kv file

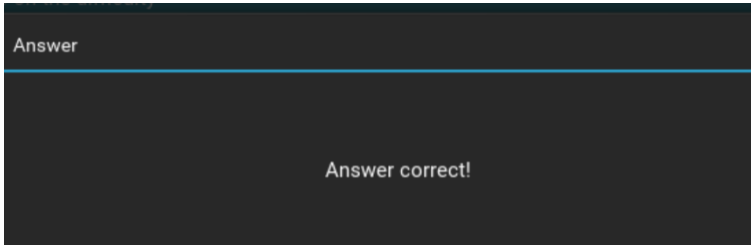
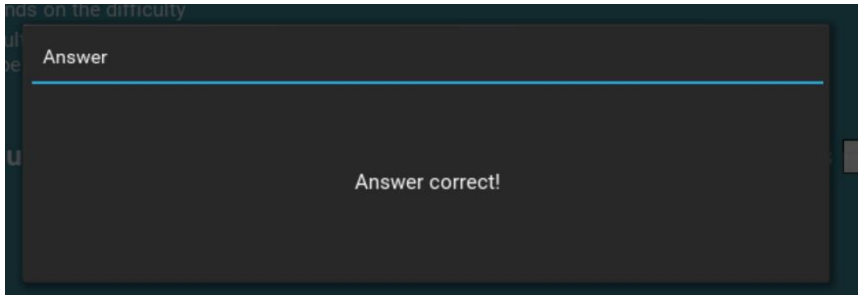
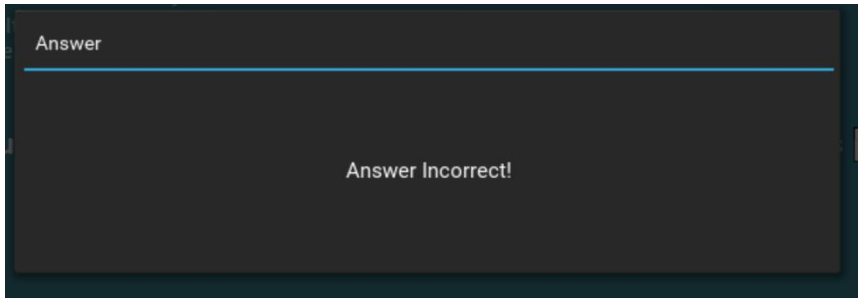


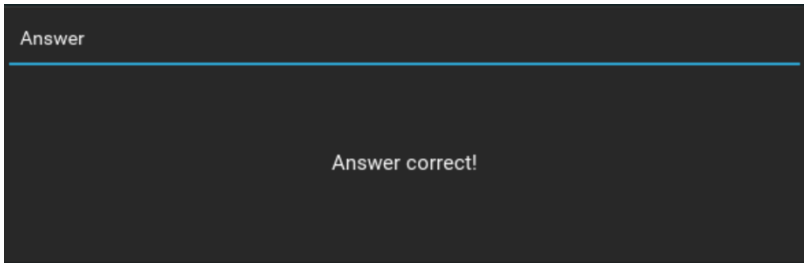
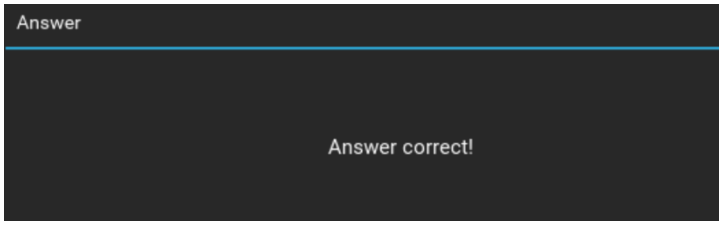
Code Screenshot 5.8 – Screen before test screen with checkboxes for the user to pick which topics they want to be tested on



Code Screenshot 5.9 – Test screen showing a question taken from the database

Alpha Testing – Maths Topics Test

Test Reference	Question	Input/User Attempt	Expected Output	Actual Outcome	Comments
4a	Differentiate $5x^2$	**10x**	The answer is correct and will result in the student gaining the relevant marks and update the progress tables.		Works As Expected
4b	Differentiate $5x^2$	**10X**	This answer will also be correct as the program will parse it and match it with the keywords in the list of the answers for this particular question		Works As Expected
4c	Differentiate $5x^2$	**432y**	As this answer is incorrect, the program will notify the user and update the progress table with this incorrect attempt.		Works As Expected

4d	Integrate $6x^2$	**2x^3**	As the user will not be able to use a superscript, a hat symbol will be used instead and should be accepted as a correct answer, giving the correct answer pop up.		Works As Expected
4e	When is the Normal Distribution a good approximation for the binomial distribution?	**N is large, and p is close to 0.5**	This answer is correct so the program should carry out the correct answer procedures as described in previous tests.		Works As Expected

After using the topic test several times and answering the questions with some intentionally incorrect answers, this was the final results for the progress table in the database. I did this as it will be useful in future testing when I am looking at my progress trackers.

	ProgressID	DiffCorrect	DiffTotal	IntCorrect	IntTotal	StatCorrect	StatTotal	TrigCorrect	TrigTotal	AvgDifficulty	Email
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	10	13	3	4	5	5	4	10	2.0	Luq@test.com
2	2	5	9	21	28	11	12	1	5	3.0	Luq2@test.com

Code Screenshot 5.10 – Progress Table after taking topic tests multiple times on different accounts

Maths Invaders Game

To implement the requested Space Invaders game, I first used a tutorial to get a foundation of how to use the turtle mod to produce the game[14]. As the tutorial was using Python 2 and my main program was on Python 3, I researched further into the Turtle module to ensure there were no compatibility issues and resolved them once I found a few[15]. I then built upon this to better suit the application. This entailed several updates to the original code I wrote with the foundations of the tutorial, including: design choices to map to my clients' requests; implementing a connection to my database with joins to store data related to the student; creating an efficient leader board sorting system to map to objective 4 for the overall scores by students and adding in the mathematics revision questions in between rounds. This section maps to ***Objective 4***.

```
'''First Maths Revision Game (MathsInvaders)'''
#Screen which has option to play game or view scores
#through buttons using kv
class MathsInvadersScreen(Screen):
    pass

#Screen to view game scores
class ViewLeaderboardScreen(Screen):
    #Creating a new leaderboard variable with a string property
    Leaderboard = StringProperty()
    #using a constructor method with variable keyword arguments
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        #using the timetable variable as an attribute of the class' objects
        Leaderboard = str()
```

Code Screenshot 6.1 – New classes and methods for the maths invader screens

Code Screenshot 6.2 – Main Screen for maths invaders to decide to play the game or view the Leader boards

```
#Merge sort leaderboard function for
def MergeSortBoard(self, LeaderSort):
    #checks if the list has more than 1 element to be sorted
    if len(LeaderSort)>1:
        #creates the midpoint using the integer division by 2
        mid=len(LeaderSort)//2
        #creates two lists by halving them
        left = LeaderSort[0:mid]
        right = LeaderSort[mid::]

        #recursively calls the function to split the list further
        MergeSort(left)
        MergeSort(right)
        #assigns 3 variables used for indexes to 0
        left_pointer = right_pointer = final_pointer = 0
        #checks that left and right pointers are less than the total number of elements
        while left_pointer < len(left) and right_pointer < len(right):
            #compares the term in the left to the term in the right list
            if left[left_pointer] < right[right_pointer]:
                #if the left[left_pointer] is smaller, it replaces the term in the inputted list
                LeaderSort[final_pointer]=left[left_pointer]
                #left pointer is incremented
                left_pointer+=1
            #otherwise the right list element is taken
            else:
                LeaderSort[final_pointer]=right[right_pointer]
                right_pointer+=1
            #final pointer is incremented by 1
            final_pointer+=1
        #if no more values in either of the LeaderSort lists then these loops are used
        #Finds and appends remaining values in the left list to final list
        while left_pointer < len(left):
            LeaderSort[final_pointer]=left[left_pointer]
            left_pointer+=1
            final_pointer+=1
        #Finds and appends remaining values in the left list to final list
        while right_pointer < len(right):
            LeaderSort[final_pointer]=right[right_pointer]
            right_pointer+=1
            final_pointer+=1
        #function values are returned in case of use further on for calculations
        return LeaderSort

    else:
        #if list has 1 element or none then its automatically returned as it can't be sorted further
        return LeaderSort
```

Code Screenshot 6.3 – Merge sort function to sort leaderboard, required to fully meet objective 4

```
#Defining view leaderboard function to allow the user to see all the game scores
def ViewLeaderboard(self):
    #Connecting to database
    DataConnect('connect')
    #Querying data from database and storing it in a variable to be ready for output
    GameLeaderboard = crsr.execute('''SELECT Email, GameScore, MathScore FROM MathsInvader''')
    #Runs merge sort algorithm to sort out the leaderboard, with mathscore being prioritised
    GameLeaderboard = self.MergeSortBoard(GameLeaderboard)
    #Creating the headings for the displayed table on screen
    GameHeadings = ['Email', 'Game Score', 'Maths Score']
    #Tabulates the data and updates the leadeboard variable on screen
    self.Leaderboard = tabulate(GameLeaderboard, headers = GameHeadings, tablefmt = "grid")
    #Disconnecting from database
    DataConnect('disconnect')
```

Code Screenshot 6.4 – Viewing leaderboard function to show the leaderboard on entry to the ViewLeaderboard Screen

```
#Main Game class and screen
class MathsInvadersGame(Screen):
    #using a constructor method with variable keyword arguments
    def __init__(self, **kwargs):
        super().__init__(**kwargs)

    #Defining function to update database when game is played
    def GameData(self, GameScore, MathScore):
        #Connecting to database
        DataConnect('connect')
        #Selecting progress id to match with user's email
        crsr.execute('''SELECT ProgressID From Progress WHERE Email=?''', (UserCheck,))
        #Storing progress id for that user in a variable
        for row in crsr.fetchall():
            ProgressID = Row[0]
        #Creating a new GameID for the game session using my IncrementID function
        GameID = IncrementID('''Select MAX(GameID) FROM MathsInvader''')
        #Inserting all information into the database
        crsr.execute('''INSERT INTO MathsInvader(GameID, Email, ProgressID, GameScore, MathScore
                    VALUES(?,?,?,?,?)''', (GameID, UserCheck, ProgressID, GameScore, MathScore))
        #Comitting all commands and disconnecting from database
        DataConnect('disconnect')

    #Creating a function to allow the user to move left and right
    def MoveLeft(self):
            #Finds user's x coordinate
            x = player.xcor()
            #Moves player the amount of pixels that the speed is set to
            x -= playerspeed
            if x < -280:
                    x = - 280
            #Shows player on that position on screen
            player.setx(x)

    #This function is the same as above but inversed the negative to move right instead
    def MoveRight(self):
            x = player.xcor()
            x += playerspeed
            if x > 280:
                    x = 280
            player.setx(x)
```

Code Screenshot 6.5 – Class for Maths Invaders game screen and methods to be used within the game for characters

```
#function for how the bullet is shot when spacebar is pressed
def FireBullet(self):
        #Declare bulletstate as a global variable as it will change outside the function
        global bulletstate
        #Checks if the bullet is ready to fire
        if bulletstate == "ready":
                #changes bullet state to fire mode
                bulletstate = "fire"
                #Move the bullet to the just above the player
                x = player.xcor()
                y = player.ycor() + 10
                bullet.setposition(x, y)
                bullet.showturtle()

#Function to check if a collision occurs
def isCollision(self, t1, t2):
        #Using the cartestian coordinates distance formula to work out how far
        #an alien is from the student spaceship
        distance = math.sqrt(math.pow(t1.xcor()-t2.xcor(),2)+math.pow(t1.ycor()-t2.ycor(),2))
        #Checks the distance is less than 15 as that is when the alien would be touching the user
        if distance < 15:
                #There is a collision so True is returned
                return True
        else:
                #There is not a collision so False is returned
                return False

#Setting game as a function to be run when button is pressed
def MathsInvadersGame(self):
    #Using turtle mod to create a screen for which the game can be played on
    #as kivy alone will not be able to track movement
    MathsInvaders = turtle.Screen()
    #Set the colour to black
    MathsInvaders.bgcolor("black")
    #Naming it as Maths Invaders, as it is related to a maths application
    MathsInvaders.title("Maths Invaders")
    #Found an online background image to match with the space invaders theme
    MathsInvaders.bgpic("MathsInvaders_Images/MathsInvaders - Background.gif")
    #Also used images for the aliens and the user controlled ship from online
    turtle.register_shape("MathsInvaders_Images/Alien.gif")
    turtle.register_shape("MathsInvaders_Images/Student.gif")
```

Code Screenshot 6.6 – Defining functions for collisions and firing a bullet along with the first part of the main game

```
#Creating the border around the game using turtle module
SpaceCreator = turtle.Turtle()
#Changed the speed from 0 to create an interesting animation when opened
SpaceCreator.speed(10)
#Setting the colour of the border
SpaceCreator.color("green")
SpaceCreator.penup()
SpaceCreator.setposition(-300,-300)
SpaceCreator.pendown()
#Creating the size of the border
SpaceCreator.pensize(10)
#Creating a 4 sided square border around main game area
for side in range(4):
        #pen moves forward for 600 pixels
        SpaceCreator.fd(600)
        #pen takes a 90 degree left turn
        SpaceCreator.lt(90)
#After the borders and the main space is setup the pen can be hidden
SpaceCreator.hideturtle()

#The user's Maths and Game Scores begins at 0
GameScore = 0
MathScore = 0
#Showing the game score on the top left corner for the user to see
score_pen = turtle.Turtle()
#Score shown instantly
score_pen.speed(0)
#Setting colour of score text
score_pen.color("white")
score_pen.penup()
#Setting at top left corner
score_pen.setposition(-290, 270)
#Creating a string showing it is the Game score
GameScoreText = "Game Score: %s" %GameScore
#Setting textual styles and font size to the Game Score indicator
score_pen.write(GameScoreText, False, align="left", font=("Arial", 15, "normal"))
#Pen is hidden once finished making score
score_pen.hideturtle()
```

Code Screenshot 6.7 – Creating the main design features of the game and the score section to display game score

```
#Similar code below but to draw maths score as well
#Showing the game score on the top left corner for the user to see
mscore_pen = turtle.Turtle()
#Maths Score shown instantly
mscore_pen.speed(0)
#Setting colour of score text
mscore_pen.color("red")
mscore_pen.penup()
#Setting at top left corner
mscore_pen.setposition(-140, 270)
#Creating a string showing it is the Game score
MathScoreText = "Maths Score: %s" %MathScore
#Setting textual styles and font size to the Maths Score indicator
mscore_pen.write(MathScoreText, False, align="left", font=("Arial", 15, "normal"))
#Pen is hidden once finished making score
mscore_pen.hideturtle()
```

Code Screenshot 6.8 – Similar code to display the math score at top with different colour to make it clear

```
#Creating the Student's Spaceship
player = turtle.Turtle()
#Using blue colour for spaceship
player.color("blue")
#using downloaded image to replace base image
player.shape("MathsInvaders_Images/Student.gif")
player.penup()
player.speed(0)
player.setposition(0, -250)
player.setheading(90)
#Setting how fast the player can move
playerspeed = 15

#Setting the amount of aliens that are on screen
number_of_aliens = 5
#Create an empty list for the aliens
aliens = []
#Add each alien on screen to the list using for loop
for i in range(number_of_aliens):
        #Create the alien
        aliens.append(turtle.Turtle())

#Creating the aliens themselves with required design
for alien in aliens:
        #setting alien colour to red
        alien.color("red")
        #using the downloaded image file as the alien
        alien.shape("MathsInvaders_Images/Alien.gif")
        alien.penup()
        alien.speed(0)
        #Aliens 'spawn' in at a random point at start using random integer
        #for both x and y coordinates, with the range within the borders of the game
        x = random.randint(-200, 200)
        y = random.randint(100, 250)
        #sets the starting position of the alien
        alien.setposition(x, y)
alienspeed = 5
```

Code Screenshot 6.9 – Creating player character and enemies using images to make them look better than simple squares

```
#Create the spaceship's bullet
bullet = turtle.Turtle()
bullet.color("blue")
#using a triangular shaped bullet
bullet.shape("triangle")
bullet.penup()
bullet.speed(0)
bullet.setheading(90)
bullet.shapesize(0.5, 0.5)
bullet.hideturtle()
#Setting the bullet's speed
bulletspeed = 20

#Define bullet state
#ready - ready to fire
#fire - bullet is firing
global bulletstate
bulletstate = "ready"

#Create keyboard bindings to allow student
#to control spaceship with respective keys
turtle.listen()
turtle.onkey(MoveLeft, "Left")
turtle.onkey(MoveRight, "Right")
turtle.onkey(FireBullet, "space")
```

Code Screenshot 6.10 – Creating spaceship bullets and keyboard bindings

```
#Main game loop
while True:
        #Allow all aliens to keep moving in the horizontal direction
        for alien in aliens:
                #Move the enemy
                x = alien.xcor()
                #uses alien speed to move the amount of pixels per second
                x += alienspeed
                alien.setx(x)
                #If aliens reach border, they move down a step
                if alien.xcor() > 280:
                        #This loops so all aliens are moved down
                        for a in aliens:
                                y = a.ycor()
                                y -= 40
                                a.sety(y)
                        #The direction is switched so they move towards opposite border
                        alienspeed *= -1
                #This performs the same actions as above but for the opposite border
                if alien.xcor() < -280:
                        #This loops so all aliens are moved down
                        for a in aliens:
                                y = a.ycor()
                                y -= 40
                                a.sety(y)
                        #The direction is switched so they move towards opposite border
                        alienspeed *= -1
```

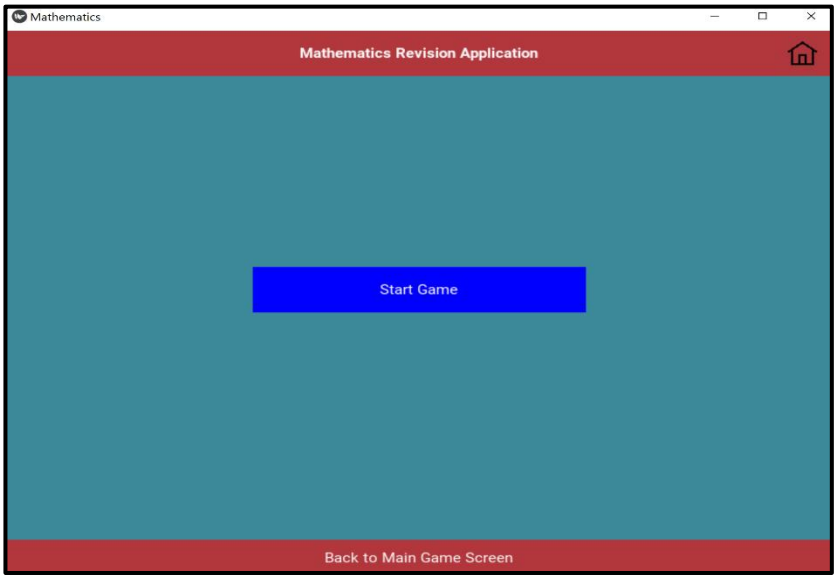
Code Screenshot 6.11 – Main game loop section, placing aliens on screen and defining their movement patterns

```
#This checks if a collision occurs between a bullet and alien using function
if self.isCollision(bullet, alien):
        #The bullet gets reset and its state changes to ready mode
        bullet.hideturtle()
        bulletstate = "ready"
        bullet.setposition(0, -400)
        #The enemy also gets reset so student can attempt to reach higher scores
        x = random.randint(-200, 200)
        y = random.randint(100, 250)
        alien.setposition(x, y)
        #The GameScore increases by 100 per success
        GameScore += 100
        GameScoreText = "Game Score: %s" %GameScore
        score_pen.clear()
        #Setting textual styles and font size to the Game Score indicator
        score_pen.write(GameScoreText, False, align="left", font=("Arial", 15, "normal"))
```

Code Screenshot 6.12 – process for when a player-controlled spaceship bullet collides with an alien

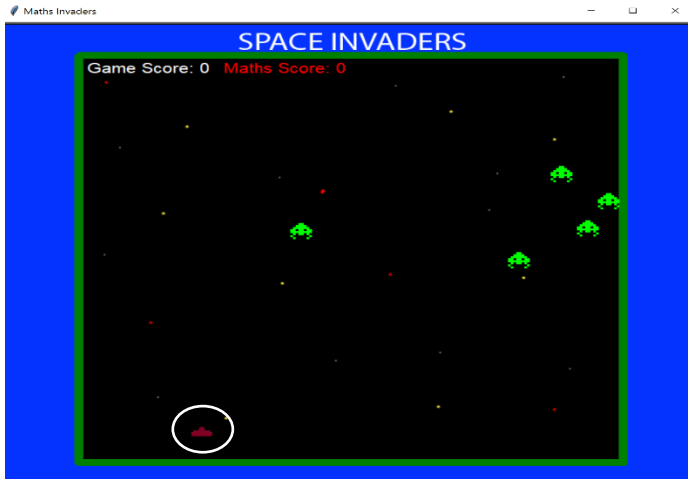
```
        #This checks a collision occurs between the student and alien
        if self.isCollision(player, alien):
                #The alien and player both get taken off the screen
                player.hideturtle()
                alien.hideturtle()
                #Iterates over 3 times to give player another chance
                #Opens up Topic Test Screen to allow user to gain another chance
                #by answering questions
                self.manager.current = 'TopicTest'
                #Works out topics for testing
                TopicTestScreen.WhichTopics()
                #Question pops up
                TopicTestScreen.StoreQA()
                #Answer is checked and updates progress database table
                TopicTestScreen.CheckAnswer()
                #Checks if user input was correct
                if txtl.text in eval(MainA):
                    #adds on 100 score to the math score
                    MathScore+=100
                #After Question is complete the database is updated
                self.GameData(GameScore, MathScore)
                #Once the chances have finished, the game ends and a message is popped up
                PoppingUp('End', 'You have no more lives remaining!')
                break

#This sets how the bullet will move up when in fire mode
if bulletstate == "fire":
        #sets y coordinate and moves according to bulletspeed variable
        y = bullet.ycor()
        y += bulletspeed
        bullet.sety(y)

#If the bullet has reached the top border
if bullet.ycor() > 275:
        #Bullet gets hidden from screen
        bullet.hideturtle()
        #Bullet set to fire mode so student can shoot again
        bulletstate = "ready"
```

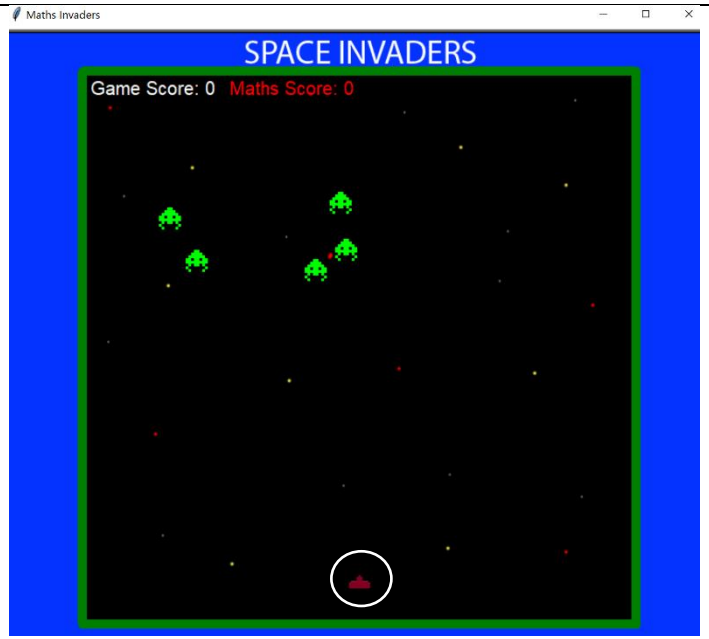
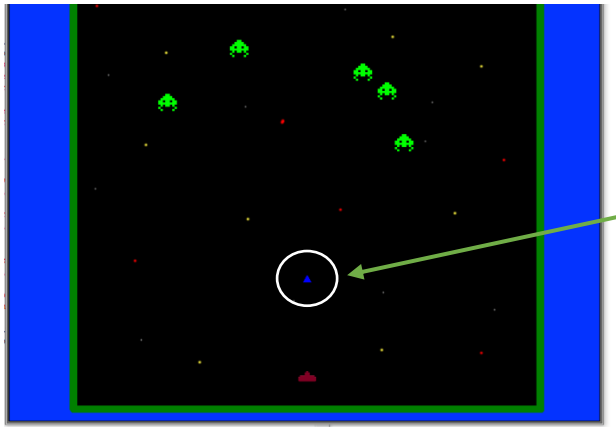
Code Screenshot 6.13 – Processes for collisions between alien and player which starts the maths topic test section to allow player to gain additional score along with the processes of defining bullet states such as when it leaves the screen

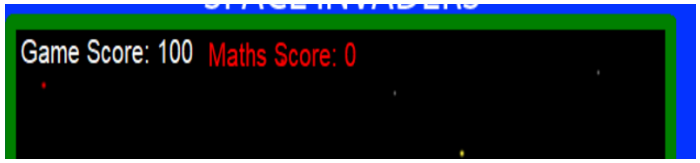
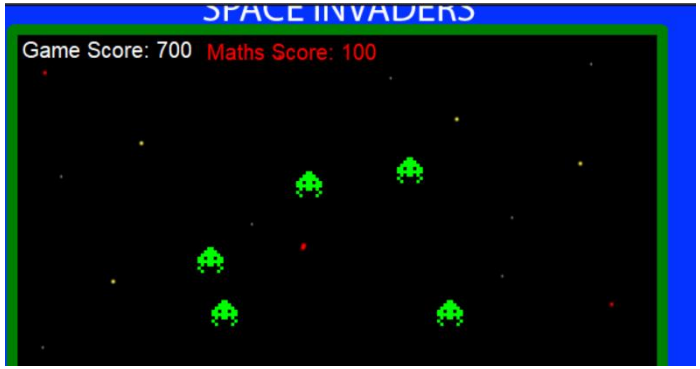
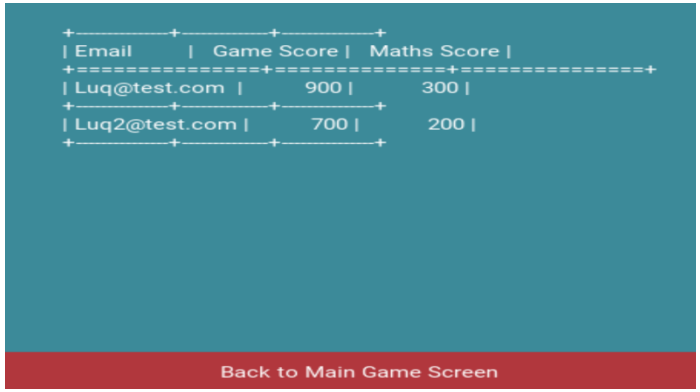


Code Screenshot 6.14 – Simple screen to allow user to start game by pressing button

Alpha Testing – Maths Invaders Game

Test Reference	Input	Expected Output	Actual Outcome	Comments
5a	**LEFT Arrow**	The user ship should move left. A w*hite circle has been placed around the ship showing it has moved*		**Works As Expected**
5b	**RIGHT Arrow**	The user ship should move right. *A white circle has been placed around the ship showing it has moved*		**Works As Expected**

5c	**UP Arrow** or **DOWN Arrow**	The game only uses left and right for the navigation of the paddle, so these arrow presses should yield no responses from the program, and the game continues *A white circle has been placed around the ship showing it has stayed central*		**Works As Expected**
5d	**SPACEBAR**	This should shoot the bullet from the ship *A white circle has been placed around the bullet*		**Works As Expected**

5f	**Collision Between Bullet and Alien**	This should increase the Game Score by 100		**Works As Expected**
5g	**Play Game and Answer Maths Question Correctly**	This should increase the Maths Score by 100		**Works As Expected**
5h	Press **View Leaderboard**	This should display the leaderboard for the games played		**Works As Expected**

Progress Tracker - Student

The Progress Tracker section is split into two sections to meet the objectives set by my client fully. It maps to the completion of ***Objective 6***. My Client wanted to have green buttons within this section's Progress Screens for the application to promote positivity in results which is why I have used that colour for the buttons. That also maps to ***Objective 1***. For testing, I created an account which I filled with data to test all aspects of the progress section.

```
#New class for the Progress Tracking Screen
class ProgressScreen(Screen):
    user_data = StringProperty()
    #using a constructor method with variable keyword arguments
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        user_data = str()

    #Defining function to acquire results across all topics for the user
    def GetAll():
        #connecting to database
        DataConnect('connect')
        #Retrieving required data using SQL queries and specifying the user's email to ensure only their data is shown
        crsr.execute('''SELECT DiffCorrect, DiffTotal, IntCorrect, IntTotal, StatCorrect, StatTotal, TrigCorrect, TrigTotal, AvgDifficulty
                        FROM Progress WHERE Email=?''', (UserCheck,))
        #Stores all data taken from the query in named variables to carry out calculations with
        for row in crsr.fetchall():
            diffCorrect = row[0]
            diffTotal = row[1]
            intCorrect = row[2]
            intTotal = row[3]
            statCorrect = row[4]
            statTotal = row[5]
            trigCorrect = row[6]
            trigTotal = row[7]
            AverageDif = row[8]

        #Calculating percentage correct for each of the topics that the program currently supports
        DiffPercent = int((diffCorrect/diffTotal) * 100)
        IntPercent = int((intCorrect/intTotal) * 100)
        StatPercent = int((statCorrect/statTotal) * 100)
        TrigPercent = int((trigCorrect/trigTotal) * 100)
        #returns all these values to be used in other functions
        return DiffPercent, IntPercent, StatPercent, TrigPercent, AverageDif
```

Code Screenshot 6.1 – New Class for Progress Screen and GetAll() function to retrieve requested data about user progress

```
#defining function to output overall data about user
def All(self):
    #Storing results from GetAll() function in variables to be used
    DiffPercent, IntPercent, StatPercent, TrigPercent, AverageDif = ProgressScreen.GetAll()
    #creating a results list which holds a tuple of all the data calculated to be outputted to the user
    results = [(str(DiffPercent)+'%', str(IntPercent)+'%', str(StatPercent)+'%', str(TrigPercent)+'%', AverageDif)]
    #creating the table headings as a tuple to be used for displaying the data to the user
    resultheadings = ('Differentiation', 'Integration', 'Statistics' , 'Trigonometry', 'Average Difficulty')
    #tabulating the data and updating the user_data text variable so the Progress Screen updates for the user
    self.user_data = tabulate(results, headers = resultheadings, tablefmt = "simple")
    #Disconnecting from the database
    DataConnect('disconnect')

#Defining function for outputting user's best topic with their percentage for that topic
def Best(self):
    #Assigning returned values from GetAll function to named variables
    DiffPercent, IntPercent, StatPercent, TrigPercent, AverageDif = ProgressScreen.GetAll()
    #creating a new list to store variables in
    CheckList = [DiffPercent, IntPercent, StatPercent, TrigPercent]
    #Using MAX function to find the user's highest correct percentage topic and then outputs it on the page
    #by updating the text variable
    if max(CheckList) == DiffPercent:
        self.user_data = "Differentiation is your best subject with: " + str(DiffPercent) + "% of answers being correct"
    elif max(CheckList) == IntPercent:
        self.user_data = "Integration is your best subject with: " + str(IntPercent) + "% of answers being correct"
    elif max(CheckList) == StatPercent:
        self.user_data = "Statistics is your best subject with: " + str(StatPercent) + "% of answers being correct"
    else:
        self.user_data = "Trigonometry is your best subject with: " + str(TrigPercent) + "% of answers being correct"

#This function is similar to the one above and uses the MIN list function instead
def Worst(self):
    DiffPercent, IntPercent, StatPercent, TrigPercent, AverageDif = ProgressScreen.GetAll()
    CheckList = [DiffPercent, IntPercent, StatPercent, TrigPercent]
    if min(CheckList) == DiffPercent:
        self.user_data = "Differentiation is your worst subject with: " + str(DiffPercent) + "% of answers being correct"
    elif min(CheckList) == IntPercent:
        self.user_data = "Integration is your worst subject with: " + str(IntPercent) + "% of answers being correct"
    elif min(CheckList) == StatPercent:
        self.user_data = "Statistics is your worst subject with: " + str(StatPercent) + "% of answers being correct"
    else:
        self.user_data = "Trigonometry is your worst subject with: " + str(TrigPercent) + "% of answers being correct"
```

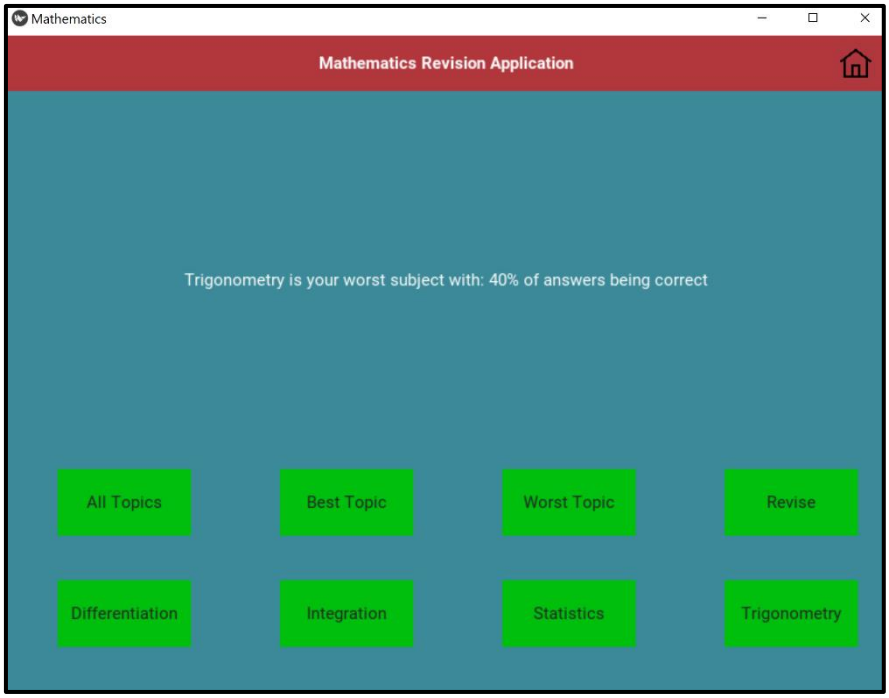
Code Screenshot 7.2 – Functions to query data, carry out required calculations and display vital progress information to the user

```
#As specific topic functions are very similar, and as some formatting requirements of SQLITE3,
#I have placed as much code into a seperate reusuable function as possible to minimise
#redundancy and improve the efficiency of the program
def General(Correct, Total):
    #Calculating percentage correct for each of the topics that the program currently supports
    Incorrect = Total - Correct
    Percent = int((Correct/Total) * 100)
    #creating a results list which holds a tuple of all the data calculated to be outputted to the user
    results = [(Correct, Incorrect, Total, str(Percent)+'%')]
    #creating the table headings as a tuple to be used for displaying the data to the user
    resultheadings = ('Total Correct', 'Total Incorrect', 'Total Answered' , 'Success Rate')
    #tabulating the data and storing it in a variable so the Progress Screen can update for the user
    Final = tabulate(results, headers = resultheadings, tablefmt = "simple")
    #Disconnecting from the database
    DataConnect('disconnect')
    #returning the tabulated version of the data
    return Final


#The next 4 functions use similar techniques to find the user data for a specfic topic
#display information to the user on the screen.
def Differentiation(self):
    #connecting to database
    DataConnect('connect')
    #Using query to get the differentiation parts of the progress table
    crsr.execute('''SELECT DiffCorrect, DiffTotal
                FROM Progress WHERE Email=? ''', (UserCheck,))
    #Stores all data taken from the query in named variables to carry out calculations with
    for row in crsr.fetchall():
        Correct = int(row[0])
        Total = int(row[1])
    #Updates user data text property to display on the screen using my General Function defined above
    self.user_data = ProgressScreen.General(Correct, Total)
```

Code Screenshot 7.3 – Generalised function to carry out calculations, tabulate data and return a final output for user and function on displaying all data on user about differentiation topic

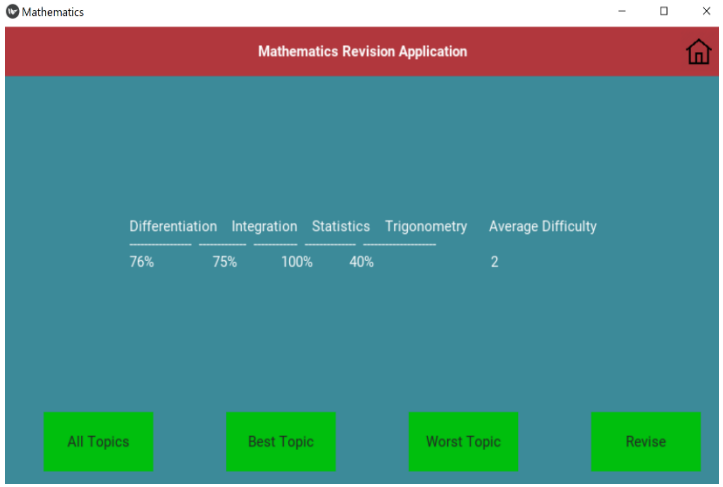
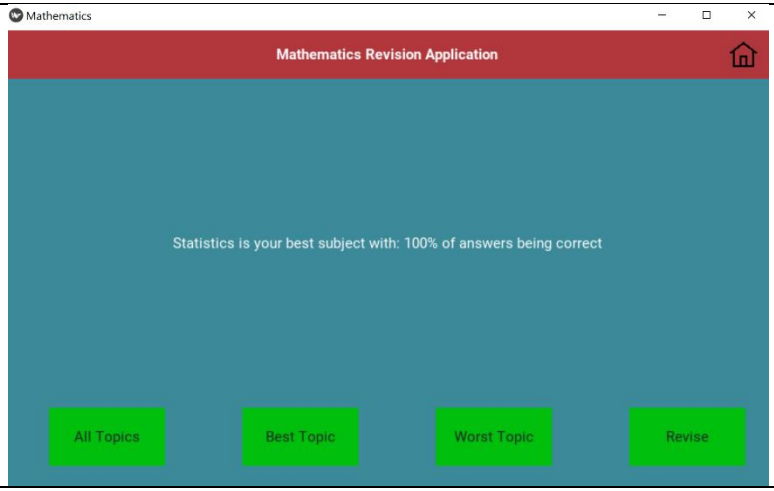
```
def Integration(self):
    #connecting to database
    DataConnect('connect')
    #Using query to get the Integration topic information
    crsr.execute('''SELECT IntCorrect, IntTotal
                FROM Progress WHERE Email=? ''', (UserCheck,))
    #Stores all data taken from the query in named variables to carry out calculations with
    for row in crsr.fetchall():
        Correct = int(row[0])
        Total = int(row[1])
    #Updates user data text property to display on the screen
    self.user_data = ProgressScreen.General(Correct, Total)


def Trigonometry(self):
    #connecting to database
    DataConnect('connect')
    #Using query to get the Trigonometry data
    crsr.execute('''SELECT TrigCorrect, TrigTotal
                FROM Progress WHERE Email=? ''', (UserCheck,))
    #Stores all data taken from the query in named variables to carry out calculations with
    for row in crsr.fetchall():
        Correct = int(row[0])
        Total = int(row[1])
    #Updates user data text property to display on the screen
    self.user_data = ProgressScreen.General(Correct, Total)


def Statistics(self):
    #connecting to database
    DataConnect('connect')
    #Using query to get the Statistics Topic data for that user
    crsr.execute('''SELECT StatCorrect, StatTotal
                FROM Progress WHERE Email=? ''', (UserCheck,))
    #Stores all data taken from the query in named variables to carry out calculations with
    for row in crsr.fetchall():
        Correct = int(row[0])
        Total = int(row[1])
    #Updates user data text property to display on the screen
    self.user_data = ProgressScreen.General(Correct, Total)
```

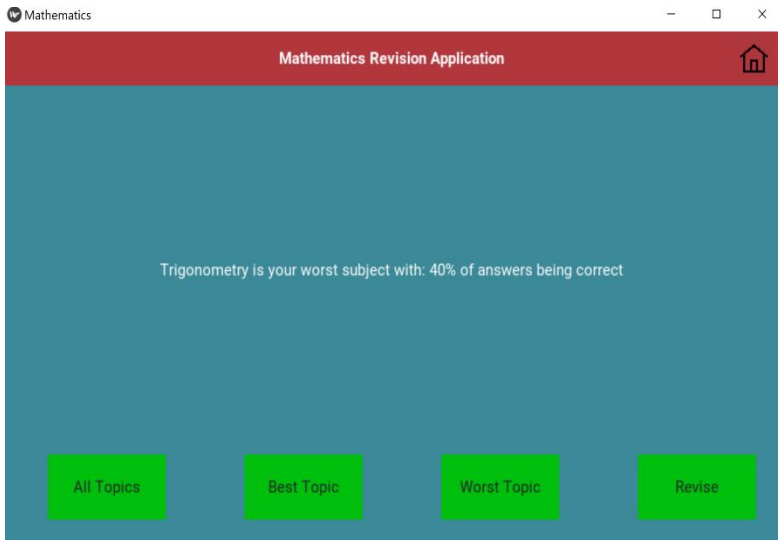
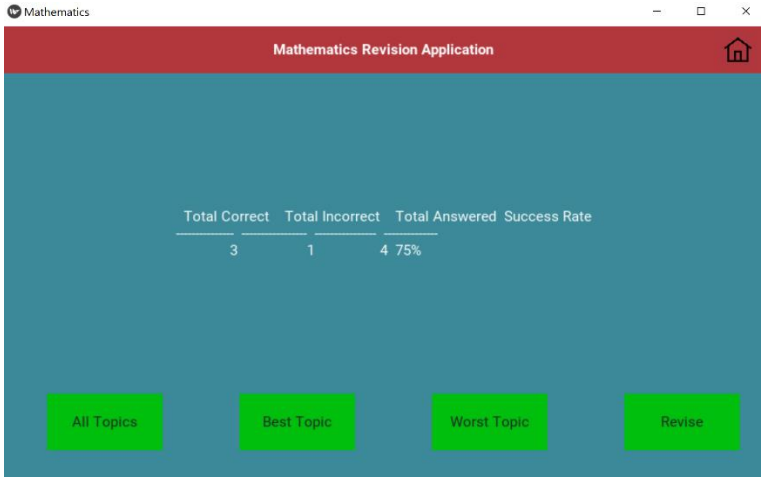
Code Screenshot 7.5 – Similar functions to output important progress information to user on other topics in program



Code Screenshot 7.6 – Progress Screen with 'Worst Topic' Button Pressed as example

Alpha Testing – Progress Tracker Student

Test Reference	User Type	Input	Expected Output	Actual Outcome	Comments
6a	Student	Press **All Topics**	This should display a table which has the percentage of correct answers given for each topic that is available and the average difficulty of the questions		Works As Expected
6b	Student	Press **Best Topic**	This should output a statement which tells the user the topic they are best at and the percentage of correct answers that they have given for that topic		Works As Expected

6c	Student	Press **Worst Topic**	This should output a statement which tells the user the topic they are worst at and the percentage of correct answers that they have given for that topic		Works As Expected
6d	Student	Press **Integration**	This should output a table for the user to see which displays the total number of correct, incorrect answered questions along with the success rate at the topic which is how many questions they have answered correctly compared to how many they have answered in total.		Works As Expected

Progress Tracking – Teachers

For this section, I will be creating similar progress tracking features for the teacher, but the teacher account will have access to a greater variety of data. This will use the combination of tables using SQLite Joins to allow the teacher to retrieve progress data on any of the students that are registered. This maps to ***objective 6***. The T in front of the function names just represents that it is used for the Teacher instead of the Student.

```
#New class for the Progress Tracking Screen
class TeacherProgressScreen(Screen):
    student_data = StringProperty()
    #using a constructor method with variable keyword arguments
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        student_data = str()


    #Defining function to retrieve and display overall information on the class or a specific student
    def TOverall(self):
        #Takes in input from the student email field
        ProgressEmail = self.ids.student_email.text
        #Connects to the database
        DataConnect('connect')
        #Creating the result headings for the tables that will be generated for the teacher
        resultheadings = ('First Name', 'Last Name', 'Diff Correct' , 'Int Correct', 'Stat Correct', 'Trig Correct')
        #Checks if it is empty and therefore means the teacher wants to see whole class results
        if ProgressEmail == '':
            #Uses an sqlite INNER JOIN to connect the Progress Table with the User Table using the student email foreign key
            #This allows the teacher to see which students have been getting which questions correct which is useful for the student
            results = crsr.execute('''SELECT FirstName, LastName, DiffCorrect, IntCorrect, StatCorrect, TrigCorrect FROM Progress INNER JOIN User
                        ON Progress.Email = User.Email;''')
        #If it is not empty then the teacher has specified a specific student to see the results for
        else:
            #Uses an sqlite INNER JOIN with another constraint using the email provided by the teacher and assigns this result to a variable
            #This means the teacher can view the results for a single specified student
            results = crsr.execute('''SELECT FirstName, LastName, DiffCorrect, IntCorrect, StatCorrect, TrigCorrect FROM Progress INNER JOIN User
                        ON Progress.Email = User.Email AND Progress.Email = ?;''', (ProgressEmail,))

        #setting the on screen text variable to be the tabulated set of results for the student(s)
        self.student_data = tabulate(results, headers = resultheadings, tablefmt = "grid")
        #Disconnecting from the database
        DataConnect('disconnect')
        print(self.student_data)
```

Code Screenshot 7.7 – New Class for Teacher Progress Screen and function for showing the overall data to the teacher about their class which uses joins between by related tables in the database

```
#Function to return totals for each topic's answered and correct questions
#This is placed in a seperate function as both the TBest and TWorst functions will
#need to use simiar code for retrieving the data and this avoids redundancy and improves efficiency
def TSummation(self):
    DataConnect('connect')
    ProgressEmail = self.ids.student_email.text
    #Setting up a list of column names to use within my SQLITE Aggregate Functions to calculate total answered questions
    #and correctly answered questions
    Columns = ['DiffTotal', 'IntTotal', 'StatTotal', 'TrigTotal','DiffCorrect', 'IntCorrect', 'StatCorrect', 'TrigCorrect']
    #Creating a list for each of the total answered questions and the total correct questions
    TotalAnswered = []
    TotalCorrect = []
    #If the input field is empty then the teacher wants to view data for entire class
    if ProgressEmail == '':
        #Loops through all 'Total' columns and uses the SQLITE SUM Aggregate function to find the total answered questions
        #for each topic
        for i in range(0,4):
            #Uses string formatting with aggregate functions to create sqlite statement
            crsr.execute('''SELECT SUM(%s) FROM Progress''' % (Columns[i]))
            #Uses list comprehension to get rid of additional tuples created by sqlite cursor object
            #then appends the answer onto the TotalAnswered list
            TotalAnswered.append([row[0] for row in crsr.fetchall()])
        print(TotalAnswered)
        #This is similar to code above but is used to find the total of the 'Correct' Columns for each
        for i in range(4,8):
            #Uses string formatting with aggregate functions to create sqlite statement
            crsr.execute('''SELECT SUM(%s) FROM Progress''' % (Columns[i]))
            #Uses list comprehension to get rid of additional tuples created by sqlite cursor object
            #then appends the answer onto the TotalAnswered list
            TotalCorrect.append([row[0] for row in crsr.fetchall()])
        #Calculates percentages for each overall topic by using the values from the
        #generated two way list from the sqlite queries above
        DiffPercent = (int(TotalCorrect[0][0])/int(TotalAnswered[0][0]))*100
        IntPercent = (int(TotalCorrect[1][0])/int(TotalAnswered[1][0])) * 100
        StatPercent = (int(TotalCorrect[2][0])/int(TotalAnswered[2][0])) * 100
        TrigPercent = (int(TotalCorrect[3][0])/int(TotalAnswered[3][0])) * 100
    else:
        #If a specific student email is given, then the function from the GetAll function from the
        #Student progress screen is used to obtain the percentages again reducing data redundancy
        DiffPercent, IntPercent, StatPercent, TrigPercent, AverageDif = ProgressScreen.GetAll(ProgressEmail)
    #Disconnects from the database
    DataConnect('disconnect')
    #Returns the percentages of success for each topic
    return ProgressEmail, int(DiffPercent), int(IntPercent), int(StatPercent), int(TrigPercent)
```

Code Screenshot 7.8 – My TSummation function which uses SQLITE Aggregate functions and list comprehension to work out the percentage success for a class or specific user on a topic

```
#Function for teacher to find the best topic for the entire class or a specified student
def TBest(self):
    #Takes in the text of the email field and the percentages for all the topics from my TSummation function
    ProgressEmail, DiffPercent, IntPercent, StatPercent, TrigPercent = self.TSummation()
    #Places the percentages in a list
    TPercentages = [DiffPercent, IntPercent, StatPercent, TrigPercent]
    #Finds max number of the list and assigns it to the 'Best' variable
    Best = max(TPercentages)
    #Checks if email input field was empty
    if ProgressEmail == '':
        #Goes through the topics with a message showing which topic was the best for the overall class
        #and gives the percentage
        if Best == DiffPercent:
            self.student_data = "Differentiation is your class' best subject with: " + str(DiffPercent) + "% of answers being correct"
        elif Best == IntPercent:
            self.student_data = "Integration is your class' best subject with: " + str(IntPercent) + "% of answers being correct"
        elif Best == StatPercent:
            self.student_data = "Statistics is your class' best subject with: " + str(StatPercent) + "% of answers being correct"
        else:
            self.student_data = "Trigonometry is your class' best subject with: " + str(TrigPercent) + "% of answers being correct"
    #If an email was given then this is run
    else:
        #Connects to database
        DataConnect('connect')
        #Adds personalisation by finding the first name of the user associated with that given email
        #to be used in the output messages
        crsr.execute('''SELECT FirstName From User WHERE Email =?''', (ProgressEmail,))
        for row in crsr.fetchall():
            #Assigns their first name to the FirstName variable
            FirstName = row[0]
        #Checks which topic was that student's best and provides a message with their name, topic and the percentage of correct questions
        #Message is then outputted on the application Progress Screen for the teacher to view
        if Best == DiffPercent:
            self.student_data = "Differentiation is " + str(FirstName)+ "'s best subject with: " + str(DiffPercent) + "% of answers being correct"
        elif Best == IntPercent:
            self.student_data = "Integration is " + str(FirstName)+ "'s best subject with: " + str(IntPercent) + "% of answers being correct"
        elif Best == StatPercent:
            self.student_data = "Statistics is " + str(FirstName)+ "'s best subject with: " + str(StatPercent) + "% of answers being correct"
        else:
            self.student_data = "Trigonometry is " + str(FirstName)+ "'s best subject with: " + str(TrigPercent) + "% of answers being correct"
        #Disconnects from the database
        DataConnect('disconnect')
```

Code Screenshot 7.9 – Function to find and display the Best Topic for the entire class or for a specified student by checking how successful students are at answering the questions of that topic

```
#This function is very similar to the one above with some changes to find the worst topic for a class or specific student
def TWorst(self):
    #Takes in the text of the email field and the percentages for all the topics from my TSummation function
    ProgressEmail, DiffPercent, IntPercent, StatPercent, TrigPercent = self.TSummation()
    #Places the percentages in a list
    TPercentages = [DiffPercent, IntPercent, StatPercent, TrigPercent]
    #Finds max number of the list and assigns it to the 'Best' variable
    Worst = min(TPercentages)
    #Checks if email input field was empty
    if ProgressEmail == '':
        #Goes through the topics with a message showing which topic was the best for the overall class
        #and gives the percentage
        if Worst == DiffPercent:
            self.student_data = "Differentiation is your class' worst subject with: " + str(DiffPercent) + "% of answers being correct"
        elif Worst == IntPercent:
            self.student_data = "Integration is your class' worst subject with: " + str(IntPercent) + "% of answers being correct"
        elif Worst == StatPercent:
            self.student_data = "Statistics is your class' worst subject with: " + str(StatPercent) + "% of answers being correct"
        else:
            self.student_data = "Trigonometry is your class' worst subject with: " + str(TrigPercent) + "% of answers being correct"
    #If an email was given then this is run
    else:
        #Connects to database
        DataConnect('connect')
        #Adds personalisation by finding the first name of the user associated with that given email
        #to be used in the output messages
        crsr.execute('''SELECT FirstName From User WHERE Email =?''', (ProgressEmail,))
        for row in crsr.fetchall():
            #Assigns their first name to the FirstName variable
            FirstName = row[0]
        #Checks which topic was that student's worst and provides a message with their name, topic and the percentage of correct questions
        #Message is then outputted on the application Progress Screen for the teacher to view
        if Worst == DiffPercent:
            self.student_data = "Differentiation is " + str(FirstName)+ "'s worst subject with: " + str(DiffPercent) + "% of answers being correct"
        elif Worst == IntPercent:
            self.student_data = "Integration is " + str(FirstName)+ "'s worst subject with: " + str(IntPercent) + "% of answers being correct"
        elif Worst == StatPercent:
            self.student_data = "Statistics is " + str(FirstName)+ "'s worst subject with: " + str(StatPercent) + "% of answers being correct"
        else:
            self.student_data = "Trigonometry is " + str(FirstName)+ "'s worst subject with: " + str(TrigPercent) + "% of answers being correct"
        #Disconnects from the database
        DataConnect('disconnect')
```

Code Screenshot 7.10 - Function to find and display the Worst Topic for the entire class or for a specified student by checking how successful students are at answering the questions of that topic

```
#Creating a general topic results function to retrieve the results from the database and display it
def TGeneral(self, ProgressEmail, CorrectName, TotalName):
    #Uses an SQLITE INNER JOIN to combine the results of the User Table and Progress Table which will allow
    #a teacher to view the entire class' results and the name of the student
    results = crsr.execute('''SELECT FirstName, LastName, %s, %s FROM Progress INNER JOIN User
                    ON Progress.Email = User.Email''' %(CorrectName, TotalName))
    #Creating the result headings
    resultheadings = ['FirstName', 'LastName', 'Total Correct', 'Total Answered']
    #Tabulating the data and updating the screen text variable
    self.student_data = tabulate(results, headers = resultheadings, tablefmt = "grid")
    #Disconnecting from the database
    DataConnect('disconnect')

#Topic Functions will display information about the class or a specified student on a single topic
#The name of the function is the topic it is referring to
def TDifferentiation(self):
    #Connecting to the database
    DataConnect('connect')
    #Taking in text field input for email and assigining it to ProgressEmail
    ProgressEmail = self.ids.student_email.text
    #If it is empty, then the entire class data is displayed
    if ProgressEmail == '':
        #This uses the TGeneral Function with the parameters being the specific column names for the topic
        self.TGeneral(ProgressEmail, 'DiffCorrect', 'DiffTotal')
    #Otherwise it will only find the results for the specific student email that was inputted
    else:
        #Uses another INNER JOIN to combine the tables and show the data to the teacher with a
        #condition to find the data that is only associated to the teacher-inputted email
        results = crsr.execute('''SELECT FirstName, LastName, DiffCorrect, DiffTotal FROM
        Progress INNER JOIN User ON Progress.Email = User.Email AND Progress.Email = ?''',(ProgressEmail,))
        #Creates the headings for the reported table
        resultheadings = ['FirstName', 'LastName', 'Differentiation Correct', 'Differentiation Total']
        #Tabulates the data and updates the student_data screen text variable
        self.student_data = tabulate(results, headers = resultheadings, tablefmt = "grid")
        #Disconnects from the database
        DataConnect('disconnect')
```

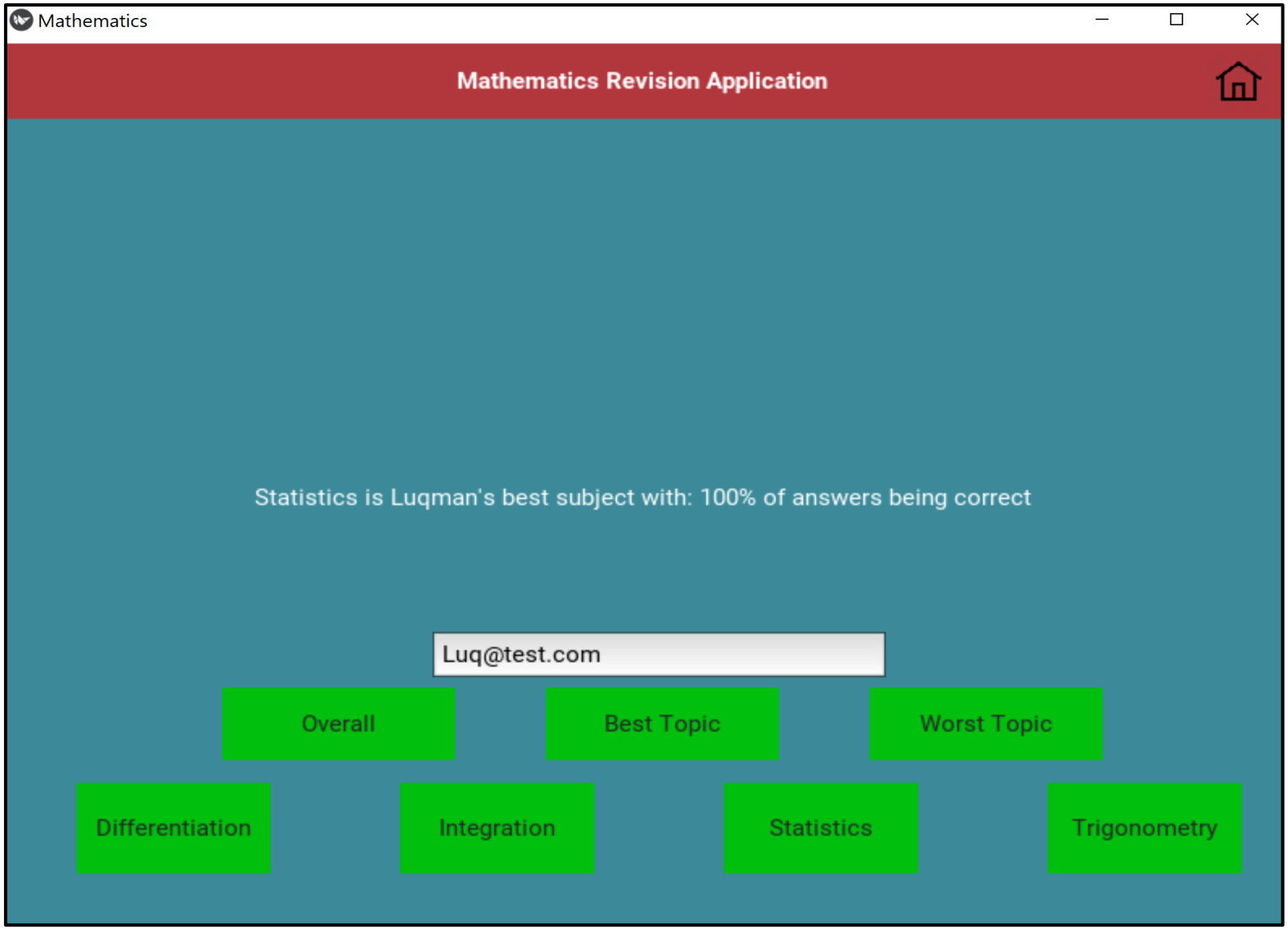
Code Screenshot 7.11 – TGeneral function to display the data to the teacher of a single topic about the entire class and TDifferentiation function to show data about differentiation topic

```
def TIntegration(self):
    #Connecting to the database
    DataConnect('connect')
    #Taking in text field input for email and assigining it to ProgressEmail
    ProgressEmail = self.ids.student_email.text
    #If it is empty, then the entire class data is displayed
    if ProgressEmail == '':
        #This uses the TGeneral Function with the parameters being the specific column names for the topic
        self.TGeneral(ProgressEmail, 'IntCorrect', 'IntTotal')
    #Otherwise it will only find the results for the specific student email that was inputted
    else:
        #Uses another INNER JOIN to combine the tables and show the data to the teacher with a condition to find the data
        #that is only associated to the teacher-inputted email
        results = crsr.execute('''SELECT FirstName, LastName, IntCorrect, IntTotal FROM Progress INNER JOIN User
                    ON Progress.Email = User.Email AND Progress.Email = ?''',(ProgressEmail,))
        #Creates the headings for the reported table
        resultheadings = ['FirstName', 'LastName', 'Integration Correct', 'Integration Total']
        #Tabulates the data and updates the student_data screen text variable
        self.student_data = tabulate(results, headers = resultheadings, tablefmt = "grid")
        #Disconnects from the database
        DataConnect('disconnect')

def TStatistics(self):
    #Connecting to the database
    DataConnect('connect')
    #Taking in text field input for email and assigining it to ProgressEmail
    ProgressEmail = self.ids.student_email.text
    #If it is empty, then the entire class data is displayed
    if ProgressEmail == '':
        #This uses the TGeneral Function with the parameters being the specific column names for the topic
        self.TGeneral(ProgressEmail, 'StatCorrect', 'StatTotal')
    #Otherwise it will only find the results for the specific student email that was inputted
    else:
        #Uses another INNER JOIN to combine the tables and show the data to the teacher with a condition to find the data
        #that is only associated to the teacher-inputted email
        results = crsr.execute('''SELECT FirstName, LastName, StatCorrect, StatTotal FROM Progress INNER JOIN User
                    ON Progress.Email = User.Email AND Progress.Email = ?''',(ProgressEmail,))
        #Creates the headings for the reported table
        resultheadings = ['FirstName', 'LastName', 'Statistics Correct', 'Statistics Total']
        #Tabulates the data and updates the student_data screen text variable
        self.student_data = tabulate(results, headers = resultheadings, tablefmt = "grid")
        #Disconnects from the database
        DataConnect('disconnect')
```

Code Screenshot 7.12 – Data finding and displaying functions for the Statistics and Integration topics

```
def TTrigonometry(self):
    #Connecting to the database
    DataConnect('connect')
    #Taking in text field input for email and assigining it to ProgressEmail
    ProgressEmail = self.ids.student_email.text
    #If it is empty, then the entire class data is displayed
    if ProgressEmail == '':
        #This uses the TGeneral Function with the parameters being the specific column names for the topic
        self.TGeneral(ProgressEmail, 'TrigCorrect', 'TrigTotal')
    #Otherwise it will only find the results for the specific student email that was inputted
    else:
        #Uses another INNER JOIN to combine the tables and show the data to the teacher with a condition to find the data
        #that is only associated to the teacher-inputted email
        results = crsr.execute('''SELECT FirstName, LastName, TrigCorrect, TrigTotal FROM Progress INNER JOIN User
                    ON Progress.Email = User.Email AND Progress.Email = ?''',(ProgressEmail,))
        #Creates the headings for the reported table
        resultheadings = ['FirstName', 'LastName', 'Trigonometry Correct', 'Trigonometry Total']
        #Tabulates the data and updates the student_data screen text variable
        self.student_data = tabulate(results, headers = resultheadings, tablefmt = "grid")
        #Disconnects from the database
        DataConnect('disconnect')
```

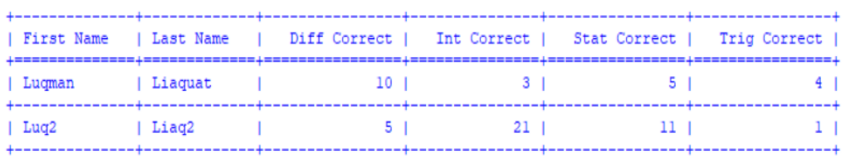
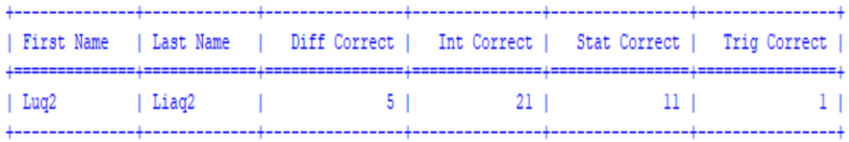
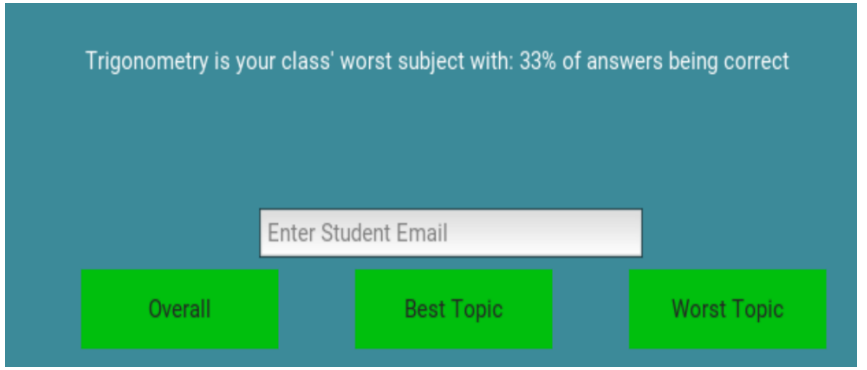
Code Screenshot 7.13 - Data finding and displaying functions for the Trigonometry Topic



Code Screenshot 7.14 – Teacher Progress Screen with example email used and result after pressing 'Best Topic' Button

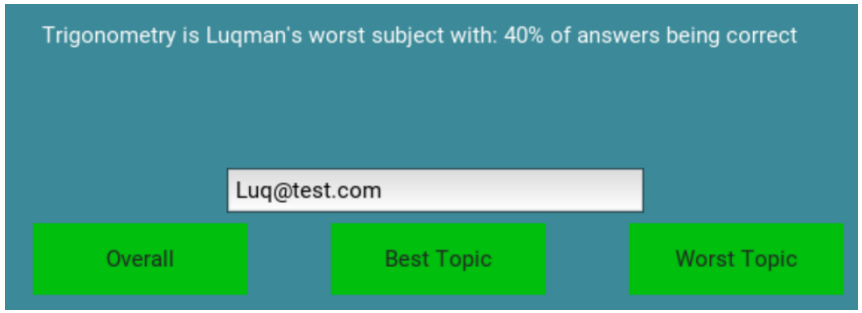
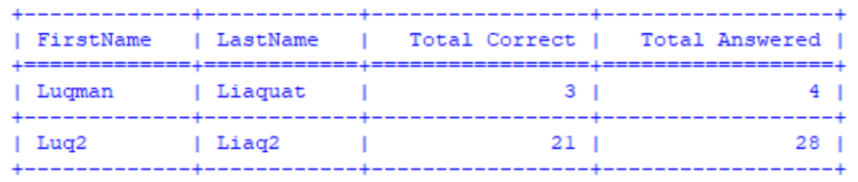
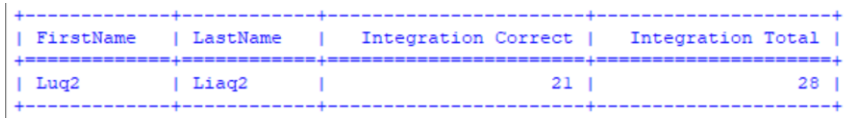
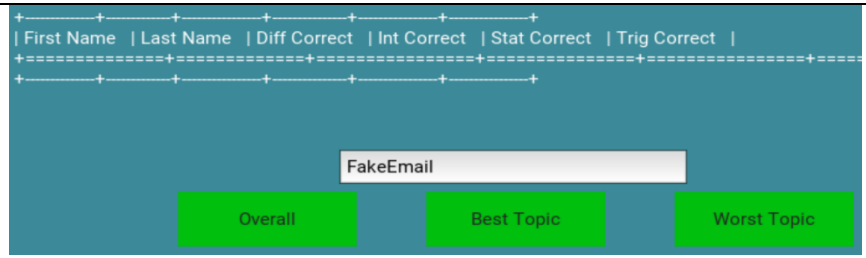
Alpha Testing – Progress Tracker Teacher

For these tests, I will show the table outputs on the terminal as it is easier to see on a word document. When screenshotting the tables, it is difficult to show the full table on the application and thus requires the application to be stretched for the table to be shown correctly. I have shown, in Code Screenshot 7.14, the screen works as expected with the text updating correctly when pressed, but that example test needs to take up one whole page in my document. It will be easier to show testing of all the necessary features using the tables in the terminal. Both the tables generated on the terminal and on the progress screen are entirely identical.

Test Reference	User Type	Input	Expected Output	Actual Outcome	Comments
6e	Teacher	Press **Overall**	This should display a table which has the number of correct answers given for each topic for the entire class with the first and last names of the user's also shown		Works As Expected
6f	Teacher	Type **Luq2@test.com** into Student Email field and then press **Overall**	This should display a table which has the number of correct answers given for each topic for only my test account (Luq@test.com)		Works As Expected
6g	Teacher	Press **Worst Topic**	This should output a statement which tells the teacher the topic that the overall class is worst at and the percentage of correct answers that they have given for that topic		Works As Expected

First Name	Last Name	Diff Correct	Int Correct	Stat Correct	Trig Correct
Luqman	Liaquat	10	3	5	4
Luq2	Liaq2	5	21	11	1

First Name	Last Name	Diff Correct	Int Correct	Stat Correct	Trig Correct
Luq2	Liaq2	5	21	11	1

6h	Teacher	Type '**Luq@test.com**' into Student Email field and then press **Worst Topic**	This should output a statement which tells the teacher the topic that only the test account is worst at and the percentage of correct answers that they have given for that topic.		Works As Expected
6i	Teacher	Press **Integration**	This should output a table which shows the teacher how all the students in the class are doing in Integration with the names of the students		Works As Expected
6j	Teacher	Type **Luq2@test.com** into Student Email field and then press **Integration**	This should output a table which shows the teacher how my test account is doing in Integration with the number of correct and total answered questions		Works As Expected
6k	Teacher	Type **FakeEmail** into Student Email field and then press **Overall**	This should not cause a break or error and should display the headings of the Overall table but have no data inside as the email doesn't exist in the application.		Works As Expected

All tests were successful, and I can continue on to the final coding part of the application from my client's objectives.

Scheduler

This scheduler section consists of 3 parts to fully meet the requirements of ***Objective 8*** given by my client. It will consist of the Timetable section which will be the code and GUI aspects of the creating and viewing a user's timetable within the application. There will also be code for setting a scheduled one-to-one meeting with a teacher depending on if both parties are free. Finally, there will be the implementation of the email section to notify both the student and teacher when the program has placed a meeting in their schedule.

Timetable

```
#New class for the Timetable functions and screen
class TimetableScreen(Screen):
    timetable = StringProperty()
    #using a constructor method with variable keyword arguments
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        timetable = str()

#New class for the Timetable functions and screen
class CreateTimetableScreen(Screen):
    timetable = StringProperty()
    #using a constructor method with variable keyword arguments
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        timetable = str()

    #Function to clear the contents of the Text Input fields when user presses the button
    #to add a timetable entry so they can enter another one quickly
    def ClearTable(self):
        #This just goes through and sets the text for each field as an empty string
        self.ids.day_number.text = ''
        self.ids.period_one.text = ''
        self.ids.period_two.text = ''
        self.ids.period_three.text = ''
        self.ids.period_four.text = ''
        self.ids.period_five.text = ''
        self.ids.period_six.text = ''
```

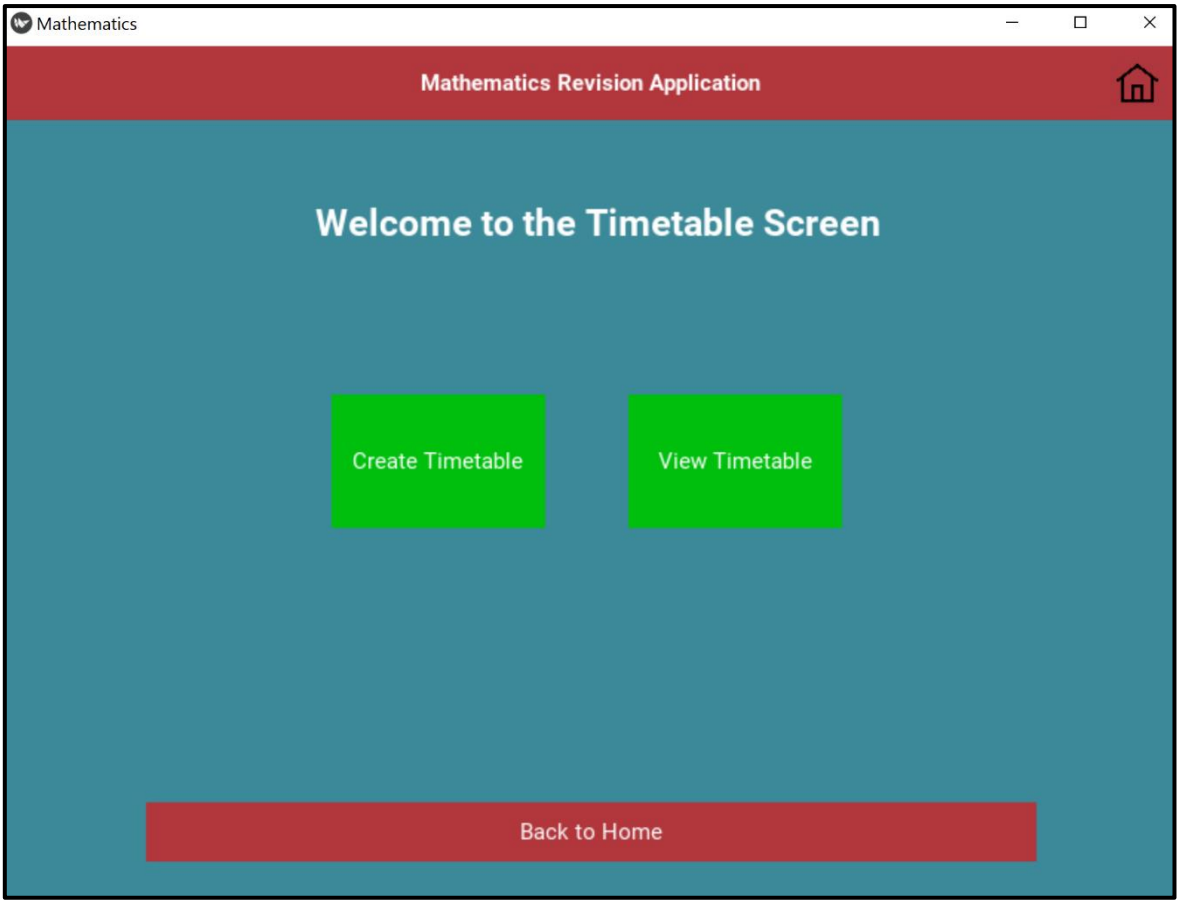
Code Screenshot 7.1 – New classes for Timetable Screen and Create Timetable screen along with a new function for the create timetable screen to clear the table when the user has submitted their response (so they can easily submit another one)

```
#Function to take in user's input of timetable and store it in the database to
#be used with the schedule system
def AddTimetable(self):
    #Uses exception handling to prevent the program from being stopped if user
    #enters incorrect values
    try:
        #Takes in all inputs from the time table creation screen and stores it in
        #respective variables
        day_number = int(self.ids.day_number.text)
        period_one = self.ids.period_one.text
        period_two = self.ids.period_two.text
        period_three = self.ids.period_three.text
        period_four = self.ids.period_four.text
        period_five = self.ids.period_five.text
        period_six = self.ids.period_six.text
        #Connecting to the database
        DataConnect('connect')
        #Using SQL Aggregate function to find a new ID and increment it when
        #new timetable entry is added
        NewTimeID = IncrementID('''Select MAX(TimeID) FROM TimeTable''')
        #Inserting all the new timetable data into the TimeTable table in the database
        crsr.execute('''INSERT INTO TimeTable(TimeID, DayNumber, Period_1, Period_2, Period_3, Period_4, Period_5, Period_6, Email)
        VALUES(?,?,?,?,?,?,?,?,?)''', (NewTimeID, day_number, period_one, period_two, period_three, period_four, period_five, period_six, UserCheck))
        #Disconnecting from database
        DataConnect('disconnect')
    #If missing or incorrect values are found then a PopUp is used to inform the user
    except ValueError:
        PoppingUp("Missing Values", "Please fill in all of the fields before pressing 'Add Timetable'!")
    #Clears table after rest of function has run
    self.ClearTable()
```

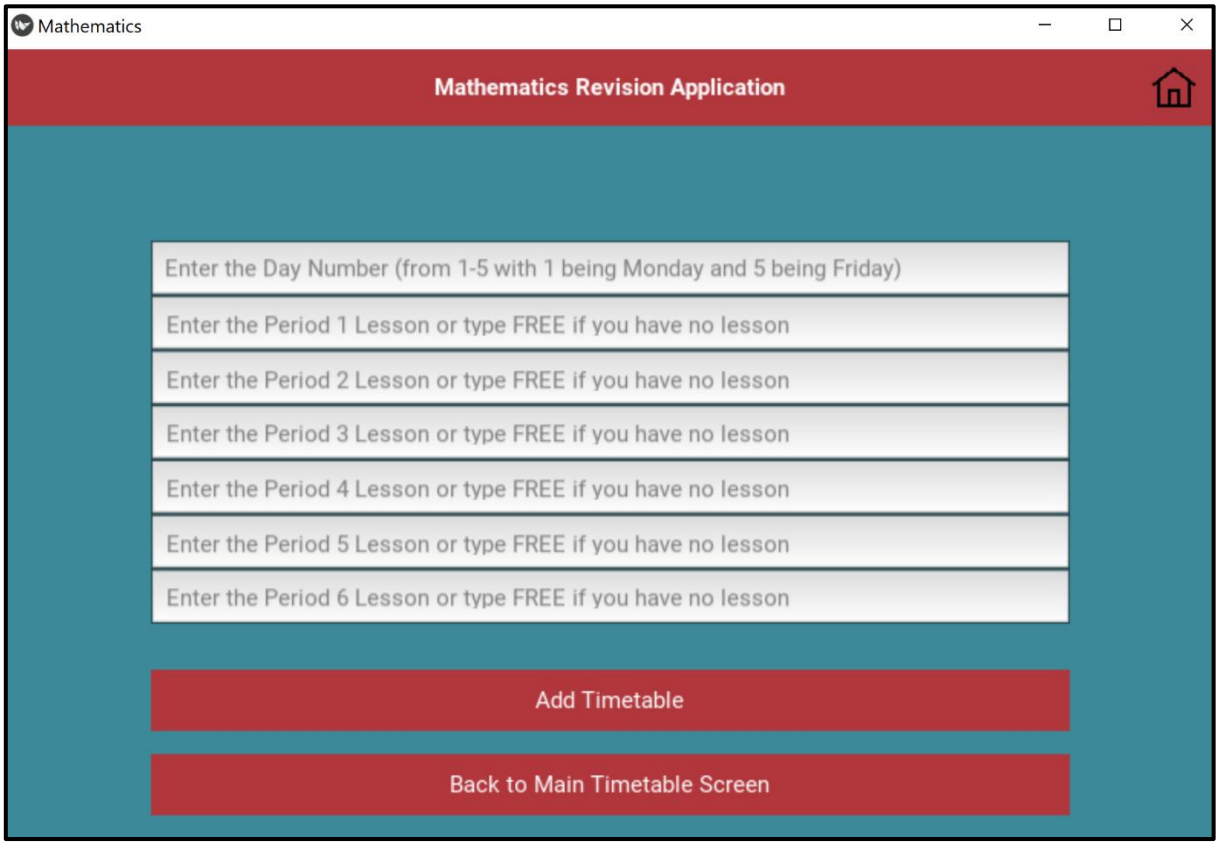
Code Screenshot 8.2 – Function to add timetable by taking in user input from GUI screen fields and storing it logically within the database

```
#New class for screen to view user's timetable
class ViewTimetableScreen(Screen):
    #Creating a new timetable variable with a string property
    timetable = StringProperty()
    #using a constructor method with variable keyword arguments
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        #using the timetable variable as an attribute of the class' objects
        timetable = str()

    #Defining view timetable function to allow the user to see their stored timetable
    #This will run when the View Timetable button is pressed from the Timetable Screen
    def ViewTimetable(self):
        #Connecting to database
        DataConnect('connect')
        #Querying data from database and storing it in a variable to be ready for output
        UserTimetable = crsr.execute('''SELECT DayNumber, Period_1, Period_2, Period_3, Period_4, Period_5,
                                        Period_6 FROM TimeTable WHERE Email = ?''', (UserCheck,))
        #Creating the headings for the displayed table on screen
        TimeHeadings = ['Day Number', 'Period 1', 'Period 2', 'Period 3', 'Period 4', 'Period 5', 'Period 6']
        #Tabulates the data and updates the timetable variable on screen
        self.timetable = tabulate(UserTimetable, headers = TimeHeadings, tablefmt = "grid")
        #Disconnecting from database
        DataConnect('disconnect')
```

Code Screenshot 8.3 – New class for viewing the user's timetable on different screen and function to show timetable by tabulating sql queried data

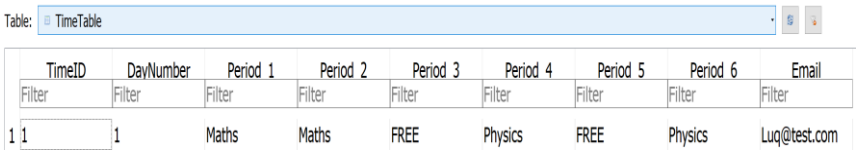
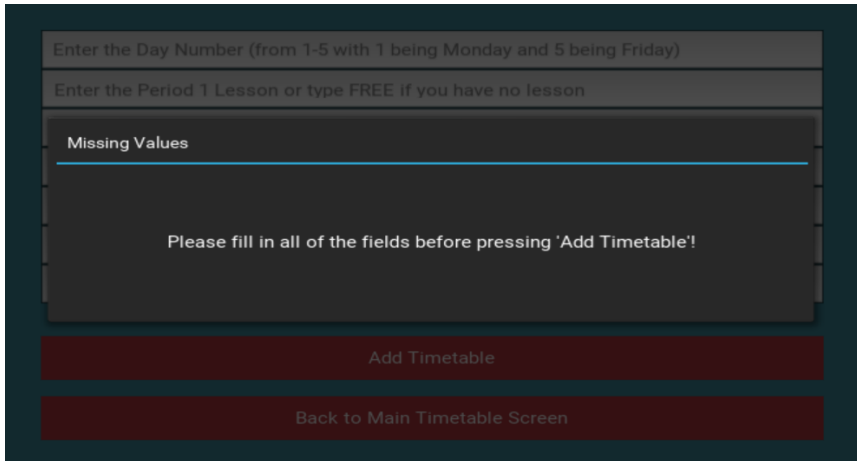


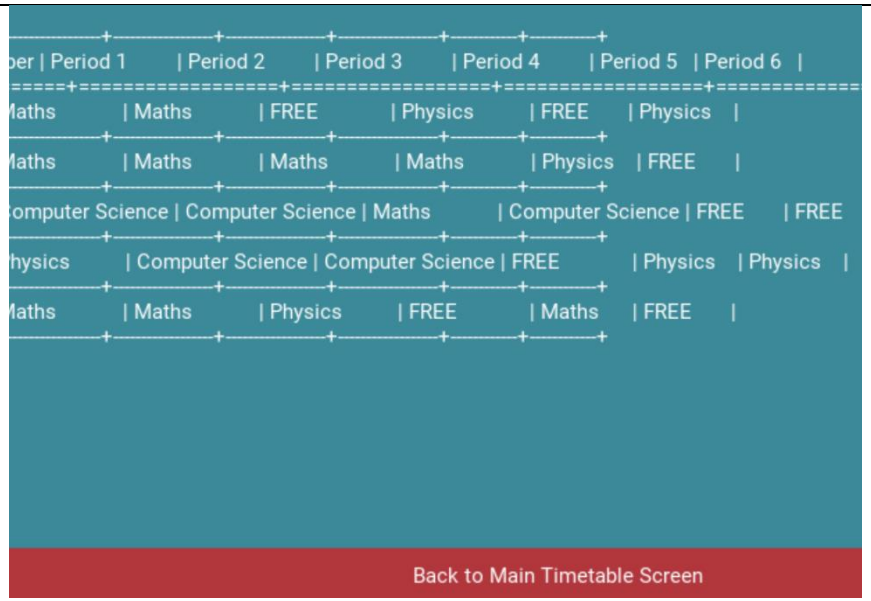
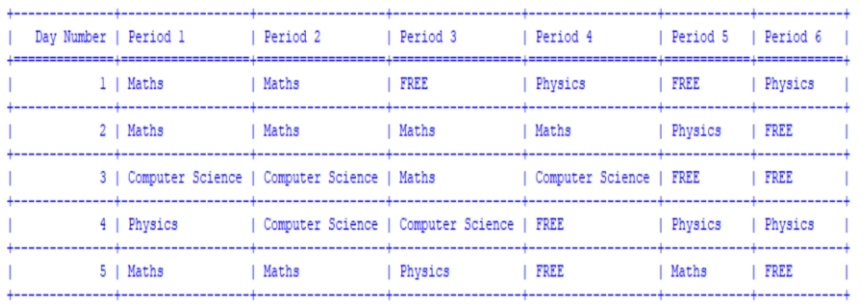
Code Screenshot 8.4 – Main Timetable Screen



Code Screenshot 8.5 – Timetable new entry screen

Alpha Testing – Timetable

Test Reference	Section	Input	Expected Output	Actual Outcome	Comments
7a	Timetable	In this order of the timetable GUI fields, type: "**1**" "**Maths**" "**Maths**" "**FREE**" "**Physics**" "**Free**" "**Physics**" then press: **Add Timetable**	This should successfully update the TimeTable table within the Mathematics database with these values and have a new ID and be linked with the same email as the account currently logged in		Works As Expected
7b	Timetable	Leave all fields empty then press: **Add Timetable**	This should display a pop-up message informing the user to input a valid value for each of the fields given		Works As Expected

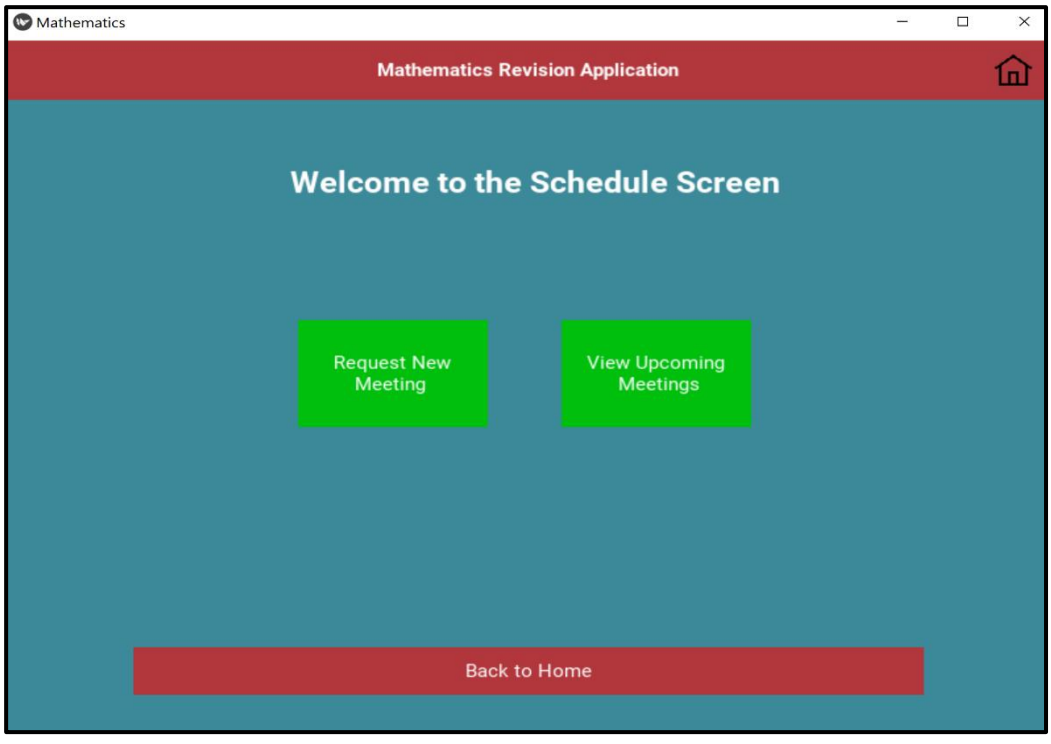
7c	Timetable	Press **View Timetable** (Ensure there are a few timetable day entries for a single user after completing test 7a)	This should change to the View Timetable screen and show the user's timetable in a clear table with headings	 *I included the above screenshot to show that it works on the GUI, but because of the stretching required to take a screenshot, it isn't very clear when I screenshot it and place it in a word document. Therefore, I have also included a terminal version which is identical to this and prints at the same time as this is outputted from the function* 	Works As Expected

Main Meeting Scheduler

The main meeting scheduler code consists of creating a scheduled meeting by taking in timetable data for both the student and teacher. One will request a meeting where they enter a preferred day, and the program attempts to find the days and periods in which both the student and teacher are available to carry out this meeting, with priority given to the preferred day. This meeting is then added into the database, and the user is informed of the exact date and period when the meeting will take place. This section also contains the code and screens for viewing the upcoming meetings for any user.

```
#Creating Schedule Screen to be midpoint to get to both
#Creating the meeting and viewing the meeting screens
#code for this is in the kv file as it is only GUI elements
class ScheduleScreen(Screen):
    pass

#Screen Class for all methods associated with creating a one-to-one meeting
class CreateMeetingScreen(Screen):
    #defining my constructor method
    def __init__(self, **kwargs):
        super().__init__(**kwargs)

    #Function to get values from text fields
    def GetValues(self):
        #Assigning text field inputs to variables to be processed
        PreferredDay = int(self.ids.preferred_day_number.text)
        Reason = self.ids.reason.text
        teacher_email= self.ids.teacher_email.text
        student_email = UserCheck
        #returns all the values
        return PreferredDay, Reason, teacher_email, student_email
```

Code Screenshot 8.6 – New class for schedule screen and create meeting screen, along with the function to obtain the user inputs from the GUI text fields, store them in variables and return them to be used in any other function.



Code Screenshot 8.7 – Main Schedule Screen

```
#Creating a function which will check if there is already a scheduled
#day in the Possible Options then removes it to prevent clashes
def RectifyClash(self, Possible):
    #Connecting to database
    DataConnect('connect')
    #retrieving the day and period of already scheduled meetings
    #from database
    crsr.execute('''SELECT ScheduleDay, SchedulePeriod
    FROM Scheduled''')
    #storing results of query in variable
    Schedules = crsr.fetchall()
    #creating a new list to store any clashes
    deleted = []
    #Compares possible free periods with already
    #scheduled times to ensure there are no clashes
    for i in range(len(Possible)):
        if Possible[i] in Schedules:
            #if a clash is found, the tuple with the schedule
            #information is added to a delete list
            deleted.append(Possible[i])

    #Two seperate loops are used as if one loop was used then the 'for loop'
    #condition would change whilst the loop is running (as the list size decreases)
    #and therefore len of the list will be out of the index range;
    #using two loops prevents this error from happening

    #This then iterates through the delete list
    for i in range(len(deleted)):
        #it then removes all clashes from the Possible list
        Possible.remove(deleted[i])
    #Disconnects from the database
    DataConnect('disconnect')
    #returns the correct Possible list of meetings
    return Possible
```

Code Screenshot 8.8 – Function to rectify any clashes and remove them from the possible options of a meeting between a student and teacher in school by checking what meetings have already been confirmed in the database

```
#Creating a new function which finds the date of the scheduled meeting
#making it easier for the user to know when they need to be at the meeting
def ScheduleDate(Day):
    #Works out the date at the current time
    Today = date.today()
    #'Day' variable is the targeted day for when the meeting will occur
    #DayDifference calculates the difference in days between the current day
    #and the targeted day modulo 7 to ensure the number is positive
    DayDifference = (Day - Today.weekday()) %7
    #The function returns the new date by using a timedelta method, adding on
    #the correct number of days to today's date and then formatting it to the
    #well known day, month, year format.
    return (Today + datetime.timedelta(DayDifference)).strftime("%d/%m/%Y")
```

Code Screenshot 8.9 – Function to calculate the date of the next scheduled meeting to provide to the user

```
#Schedule Meeting method, all functionality in this is seperate from the CreateMeeting Screen
#objects or fields (shown by the general parameters) so that it may be used again in other
#screens or functions outside this class
def ScheduleMeeting(self, PreferredDay, Reason, T_Email, S_Email):
    #Connecting to the database
    DataConnect('connect')
    #storing timetable of the teacher
    crsr.execute('''SELECT DayNumber, Period_1, Period_2, Period_3, Period_4, Period_5,
    Period_6 FROM TimeTable WHERE Email =?''', (T_Email,))
    teacher_timetable = crsr.fetchall()
    #storing timetable of the student
    crsr.execute('''SELECT DayNumber, Period_1, Period_2, Period_3, Period_4, Period_5,
    Period_6 FROM TimeTable WHERE Email =?''', (S_Email,))
    student_timetable = crsr.fetchall()

    #Creates a new list to store the available combinations of days and periods
    #when the student is free to have a meeting
    Available = []

    #Preferred day code is run first to place it at the front of the list
    #Iterates for the length of the timetable of the student on one day
    for i in range(len(student_timetable[PreferredDay-1])):
        #As it is a tuple stored within a list, the referencing of a particular field is done
        #using a similar method to matricies with the 'row' and 'column'
        #This checks if any of the lessons are FREE
        if student_timetable[PreferredDay-1][i] == 'FREE':
            #If it is free then it is added on to the list of available meeting days
            Available.append((PreferredDay, i))

    #Iterates for the length of the timetable of the student on each day
    #to prevent it from unnecessarily iterating over the 'PreferredDay' Day,
    #I have used list comprehension with a condition to remove it from the range
    for j in [x for x in range(len(student_timetable)) if x != PreferredDay]:
        #Once in a specific day, it loops over the periods in that day by checking length of the list
        for k in range(len(student_timetable[j])):
            #This checks if any of the lessons are FREE using the method from above
            if student_timetable[j][k] == 'FREE':
                #If it is free then it is added on to the list of available meeting days as a tuple
                Available.append((j, k))
```

```
    #The Possible list stores the possible meeting days and periods where both the student and the teacher
    #are free
    Possible = []
    #This iterates for how many available day-period combinations the student is available for
    for i in range(len(Available)):
        #Checks if the teacher is free on the day and period that the student is free
        if teacher_timetable[Available[i][0]][Available[i][1]] == 'FREE':
            #If the student and teacher are both free then a tuple is appended to the Possible List
            Possible.append((Available[i][0]+1,Available[i][1]))

    #After the possible meeting times have been generated and appended to the list,
    #The rectify clash functions checks and fixes any possible clashes
    Possible = CreateMeetingScreen.RectifyClash(self, Possible)
```

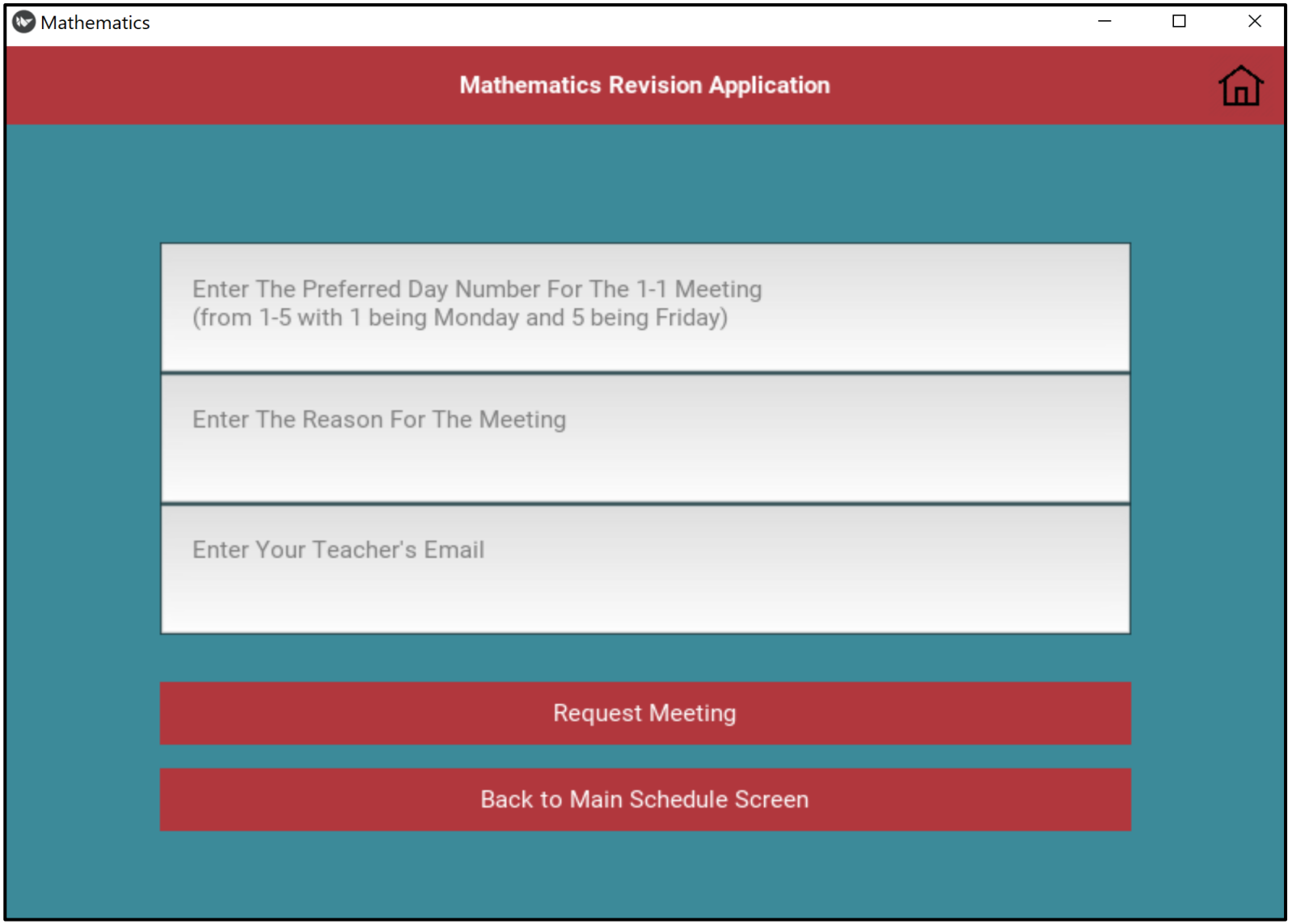
Code Screenshot 8.10 – First Part of ScheduleMeeting function which contains all the main code for program to process and calculate a suitable meeting for both student and teacher, and store this in the database

```
#Checks if no times were found to have the one-to-one meeting
if Possible == []:
    #A pop up is used to inform the user that a meeting could not be placed
    PoppingUp("None Found", "An attempt was made to find a meeting time, however there were none available!")
else:
    #Setting the schedule days and periods from the first available time in the Possible list (which has a preference for preferred day)
    ScheduledDay = Possible[0][0]
    ScheduledPeriod = Possible[0][1]+1
    #Creating a new ID for the Scheduled table for the new meeting using an SQL aggregate function
    ScheduleID = IncrementID('''Select MAX(ScheduleID) FROM Scheduled''')
    #Inserting the new information for the schedule into the database
    crsr.execute('''INSERT INTO Scheduled(ScheduleID, ScheduleDay, SchedulePeriod, Reason, Email)
    VALUES(?,?,?,?,?)''', (ScheduleID, ScheduledDay, ScheduledPeriod, Reason, S_Email))
    #Finds the date of the meeting using the ScheduleDate function
    NewDate = CreateMeetingScreen.ScheduleDate(ScheduledDay-1)
    #After completion, user is informed when their meeting is set with the full date and the Period
    PoppingUp('Success', ("Your meeting has been set up for: "+ str(NewDate) + "at Period " + str(ScheduledPeriod)))
#Disconnecting from the database
DataConnect('disconnect')
```

Code Screenshot 8.11 – Second Part of the ScheduleMeeting function.

```
#This function is for the GUI button to take in the user inputted values
#and call the ScheduleMeeting function
def RequestMeeting(self):
    #Variables assigned to the return values from the GetValues() method
    PreferredDay, Reason, teacher_email, student_email = self.GetValues()
    #ScheduleMeeting function is called
    self.ScheduleMeeting(PreferredDay, Reason, teacher_email, student_email)
```

Code Screenshot 8.12 – RequestMeeting function is bound to the GUI button and uses the GetValues ScheduleMeeting functions to schedule a meeting



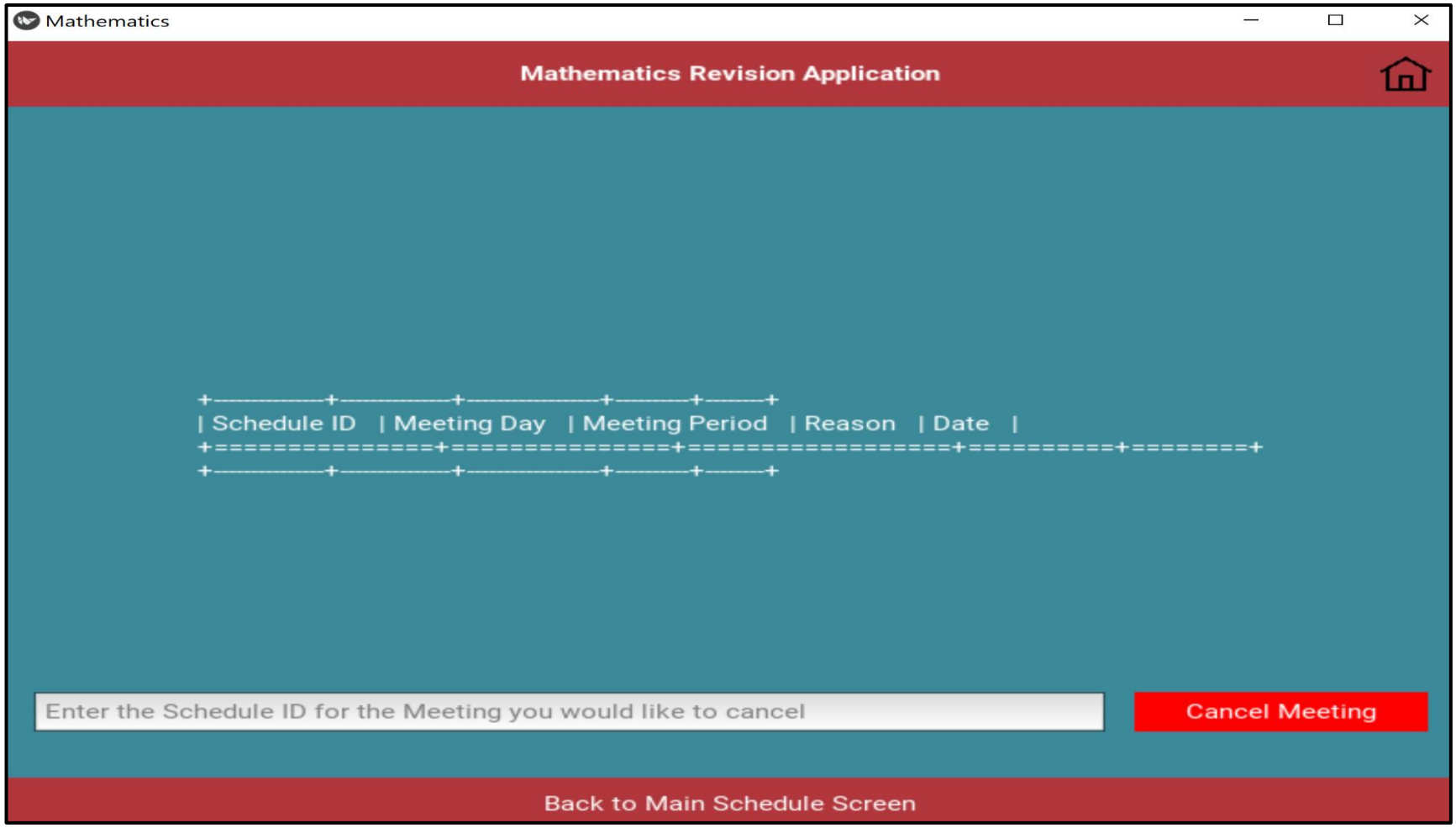
Code Screenshot 8.13 – Requesting a one to one meeting screen

```
#New screen and Class for viewing the upcoming meetings for that particular user
#and option to cancel an upcoming meeting (only meetings for that user are shown)
class ViewMeetingScreen(Screen):
    #Creating a new scheduler variable with a string property
    user_schedule = StringProperty()
    #using a constructor method with variable keyword arguments
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        #using the user_schedule variable as an attribute of the class' objects
        user_schedule = str()

    #Function to view upcoming meetings
    def ViewMeeting(self):
        #Connecting to database
        DataConnect('connect')
        #Uses as an sqlite query to obtain the required information
        #from the Scheduled table in the database
        crsr.execute('''SELECT ScheduleID, ScheduleDay, SchedulePeriod, Reason
                    FROM Scheduled WHERE Email =?''', (UserCheck,))
        #Assigns the result from the sql query to a variable
        Meetings = crsr.fetchall()
        #Goes through all meetings and attaches a date using the
        #ScheduleDate function I created
        #Iterates through the Meetings list of queried data
        for i in range(len(Meetings)):
            #Calculates the date by working out the day the meeting is set out for
            #and then finding the next available date using the ScheduleDate function
            AddDate = CreateMeetingScreen.ScheduleDate(Meetings[i][1]-1)
            #Concatenates the original tuple of data for one meeting with
            #a tuple that stores the date for the meeting and replaces
            #the original tuple(as the returned query data is in tuples which are immutable)
            Meetings[i] = Meetings[i] + (AddDate,)

        #Creating headers for the tabulated meeting table to display to user
        MeetingHeaders = ["Schedule ID", "Meeting Day", "Meeting Period", "Reason", "Date"]
        #tabulating data and updating on screen text variable to show to user
        self.user_schedule = tabulate(Meetings, headers = MeetingHeaders, tablefmt = "grid")
        #Disconnecting from database
        DataConnect('disconnect')
```

Code Screenshot 8.14 – New class for the screen to view all scheduled meetings for the logged in user

```
#Function for user to cancel a meeting specified by the ID
def CancelMeeting(self):
    DataConnect('connect')
    #Takes in user input from field
    ScheduleID = self.ids.meeting_id.text
    #Uses error handling to check if user input is valid
    try:
        #Runs query to delete the entry in the table matching the id given by user
        #and ensures that they can only delete their own meetings by checking if the
        #Email as well
        crsr.execute('''DELETE FROM Scheduled WHERE ScheduleID =? AND Email=?''', ((ScheduleID,), (UserCheck,)))
        #Pop up message telling user their meeting has been deleted
        PoppingUp('Success', "Your meeting has been deleted")
        #Commiting query and disconnecting from database
        DataConnect('disconnect')
        #Runs the View Meetings again to refresh the page and show the remaining meetings
        self.ViewMeeting()
    #If the ID given by the user does not exist or the entry they try to delete is not
    #associated with their email then an sqlite3 operational error occurs
    #this code catches that error
    except sqlite3.OperationalError:
        #Informs user of the error when trying to delete the meeting
        PoppingUp('Error', "That ID does not exist or is not associated with your Account!")
```

Code Screenshot 8.15 – Cancel Meeting function to allow a user to cancel a meeting and delete it from the database: function also ensures the user can only delete their own meetings



Code Screenshot 8.16 – Viewing scheduled meetings table, no data shown in this screenshot as entries haven't been made but this will be shown in the alpha testing on the next few pages.

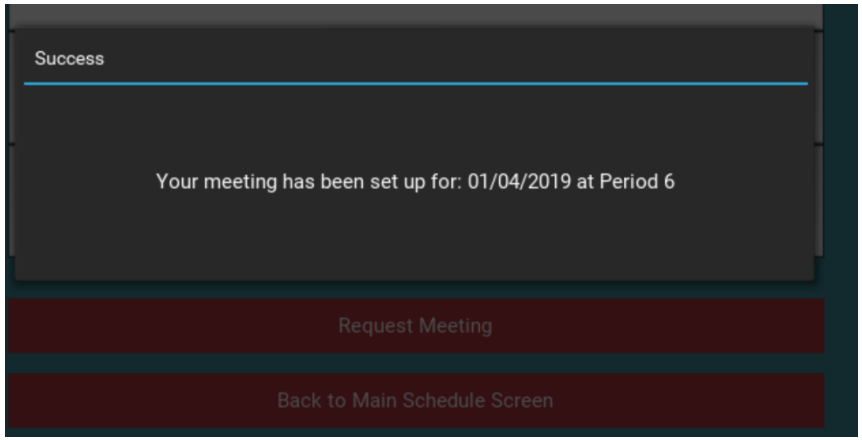
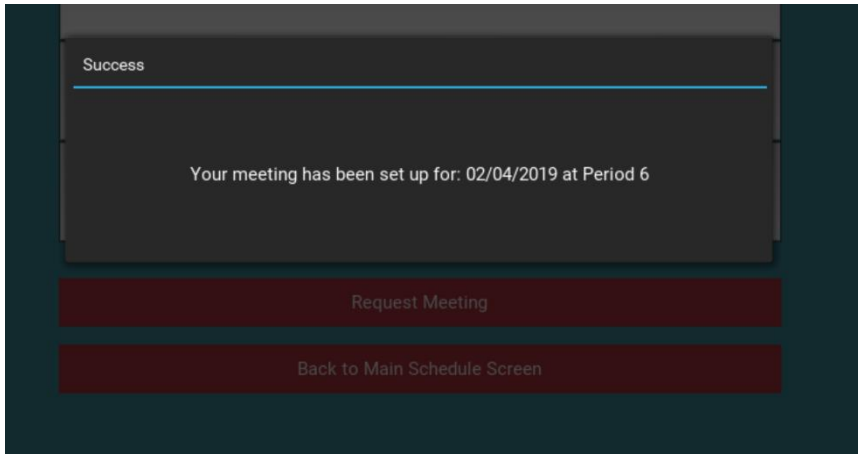
Alpha Testing – Main Meeting Scheduler

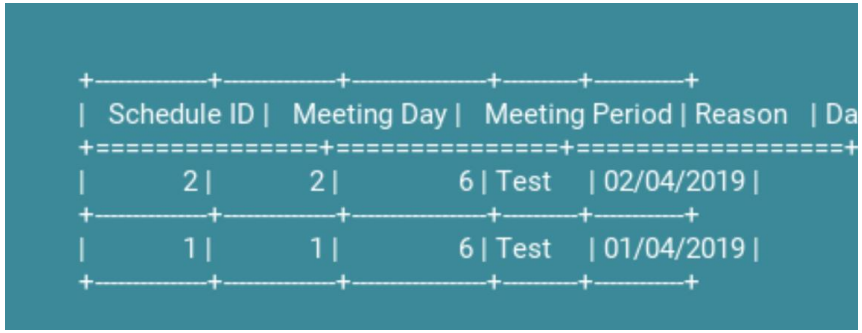
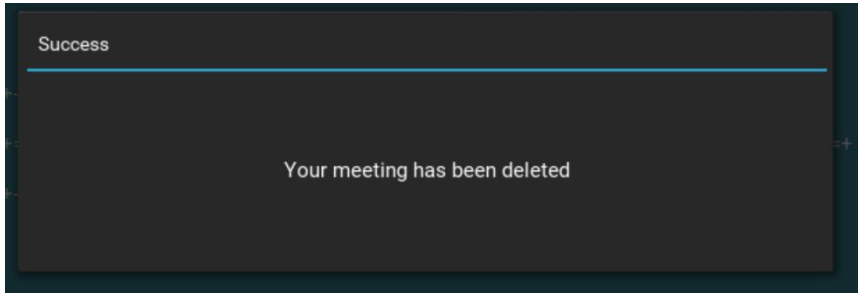
For these tests, I have inserted in some test data, within one test student account and one test teacher account, for the timetables (using my program from above). The timetables that are in the database and that are being analysed by the program during the running of the functions for this section are shown below. TimeIDs 1 – 5 are for the test student account, and TimeIDs 6-10 are for the test teacher account. This section maps to Objective 8A.

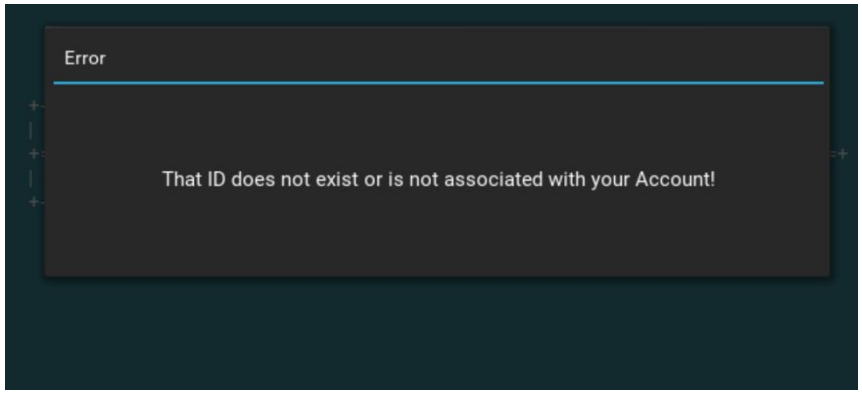
TimeTable

TimeID	DayNumber	Period 1	Period 2	Period 3	Period 4	Period 5	Period 6	Email
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	Maths	Maths	FREE	Physics	FREE	Physics	Luq@test.com
2	2	Maths	Maths	Maths	Maths	Physics	FREE	Luq@test.com
3	3	Computer Scie...	Computer Scie...	Maths	Computer Scie...	FREE	FREE	Luq@test.com
4	4	Physics	Computer Scie...	Computer Scie...	FREE	Physics	Physics	Luq@test.com
5	5	Maths	Maths	Physics	FREE	Maths	FREE	Luq@test.com
6	1	Maths	Maths	Maths	FREE	FREE	Maths	Teacher@test....
7	2	Maths	Maths	Maths	Maths	FREE	FREE	Teacher@test....
8	3	Maths	Maths	FREE	Maths	FREE	Maths	Teacher@test....
9	4	Maths	Maths	Maths	Maths	FREE	FREE	Teacher@test....
10	5	Maths	Maths	FREE	FREE	FREE	FREE	Teacher@test....

Code Screenshot 8.17 – Timetables for test student and teacher accounts to be used in this section of alpha testing.

Test Reference	Section	Input	Expected Output	Actual Outcome	Comments
7d	Main Meeting Scheduler	Enter "**1**" in Preferred Day Number field, Enter "**Test**" in Reason field and Enter "**Teacher@test.com**" in teacher email field then press: **Request Meeting**	This should search through the teacher's and student's timetable and find a time to create a scheduled meeting, with day 1 (Monday) being prioritised. A pop up detailing the data and period for this meeting will be shown as well.		Works As Expected
7e	Main Meeting Scheduler	Repeat Test 7d with exactly the same inputs (to test the rectify clash functions)	This should search through the teacher's and student's timetable and find a time to create a scheduled meeting. As one day is already taken from the previous test, this will not be displayed, and the rectify function will get rid of it from the possible dates for the meeting, showing a new and different meeting time to test 7d	 New meeting scheduled for the next day as the first day is already taken.	Works As Expected

7f	Main Meeting Scheduler	Press **View Upcoming Meetings** Button (On Main Schedule Screen)	This should show the user only their upcoming meetings in a table in the GUI with the date, period and the ID for the scheduled meeting	 Clearer version from terminal: Schedule ID | Meeting Day | Meeting Period | Reason | Date 2 | 2 | 6 | Test | 02/04/2019 1 | 1 | 6 | Test | 01/04/2019	Works As Expected
7g	Main Meeting Scheduler	Enter “**1**” in ID to Delete field then press **Cancel Meeting**	This will allow the user to cancel this meeting as it is linked to their name. This should show as successfully cancelling the meeting and will update the screen accordingly	 Schedule ID | Meeting Day | Meeting Period | Reason | Date 2 | 2 | 6 | Test | 02/04/2019	Works As Expected

7h	Main Meeting Scheduler	Enter “32” in ID to Delete field then press **Cancel Meeting**	This ScheduleID is not a valid ID or not linked to the user’s account, and therefore the user will not be able to cancel this meeting. Hence, a pop up should appear informing the user they cannot cancel this meeting.		Works As Expected

Meeting Email Sender

For ***objective 8 (Specifically 8B)***, my client asked to add on the criteria of sending an email to both the student and teacher to notify them of a scheduled meeting. To fully map to this objective, I researched further into the processes required to add in this functionality in my program. This led me to explore the SMTP library for Python, which allows me to use the Application-Layer SMTP protocol to send emails from within Python[16]. The functions that I have created below are used in the *CreateMeetingScreen* class and are called when a meeting has been created at the end of the *ScheduleMeeting* function (Code Screenshot 8.11).

```
#Creating a function to get the user's first name to personalise the emails
def GetName(self, Email):
    #Creating a new variable to store the name in
    Name = ''
    #connecting to database
    DataConnect('connect')
    #Selecting name from database for the email given
    crsr.execute('SELECT FirstName FROM User WHERE Email=?''', (Email,))
    #converts list of tuples into a single variable that can be used easily
    for row in crsr.fetchall():
        #assigns the first name to the Name variable
        Name = row[0]
    #Returns the first name of user
    return Name
```

Code Screenshot 8.18 – GetName function to return the first name of a user to be used in the email for personalisation

```
    PoppingUp('Success', ("Your meeting has been set up for: "+ str(NewDate) + " at Perio
    #Sending email to teacher and student when meeting has been set up
    CreateMeetingScreen.MeetingEmails(T_Email, S_Email, NewDate, ScheduledPeriod, Reason)
#Disconnecting from the database
DataConnect('disconnect')
```

Code Screenshot 8.19 – Adding in email function to the ScheduleMeeting function within CreateMeetingScreen class

```
#Creating function to produce and send emails to the student and teacher
#after a meeting has been scheduled
def MeetingEmails(self, T_Email, S_Email, Date, ScheduledPeriod, Reason):
    #Assigning Username and Password for the maths gmail account that was
    #created for this project
    SenderAccount = 'mathematicsnea@gmail.com'
    #Password for this account (redacted from this screenshot for security reasons)
    SenderPassword = '********'

    #Making a connection to the SMTP gmail server using port 587
    EmailServer = smtplib.SMTP('smtp.gmail.com', 587)
    #Sending an 'Extended Hello' command to the server to initiate SMTP Conversation
    EmailServer.ehlo()
    #Using the Start TLS protocol command to upgrade security with a secure connection
    #using either TLS or SSL
    EmailServer.starttls()
    #Sends the login command with the username and password to the server to attempt logging in
    EmailServer.login(SenderAccount, SenderPassword)
    #assigning a general variable with the email subject for both teacher and student being
    #a new scheduled meeting
    EmailSubject = 'New Scheduled Meeting'

    #using the GetName function to assign the first name of the student to a variable
    StudentName = CreateMeetingScreen.GetName(S_Email)
    #Creating a personalised email message for the student
    StudentMessage = ('''Dear %s,
                \nYou have a scheduled one-to-one meeting with your maths teacher on the %s at Period %s.
                \nThe reason given for this meeting is: %s
                \nPlease ensure you attend this meeting.
                \nKind Regards,
                \nLuqman's Mathematics Application''' % (StudentName, Date, ScheduledPeriod, Reason))
    #Creating a body to fill in and join the fields for To, From and the Subject of an email
    #along with the message itself and assigning to a variable to be used in the sendmail function
    StudentBody = '\r\n'.join(['To: %s' % S_Email,'From: %s' % SenderAccount, 'Subject: %s' % EmailSubject,
                           '', StudentMessage])
```

Code Screenshot 8.20 – First Part of the Email generating and sending function using SMTP protocol.

```
#Similar code for the maths teacher which personalises it to them
#using the GetName function to assign the first name of the teacher to a variable
TeacherName = CreateMeetingScreen.GetName(T_Email)
#Creating a personalised email message for the teacher
TeacherMessage = ('''Dear %s,
            \nYou have a scheduled one-to-one meeting with your student (%s) on the %s at Period %s.
            \nThe reason given for this meeting is: %s
            \nKind Regards,
            \nLuqman's Mathematics Application''' % (TeacherName, StudentName, Date, ScheduledPeriod, Reason))
#Creating the body to fill in all fields and the message
TeacherBody = '\r\n'.join(['To: %s' % T_Email,'From: %s' % SenderAccount, 'Subject: %s' % EmailSubject,
                    '', TeacherMessage])

#Uses exception handling to send emails in case there is an error to prevent the program
#from stopping
try:
    #Sends an email to both the teacher and the student
    EmailServer.sendmail(SenderAccount, [S_Email], StudentBody)
    EmailServer.sendmail(SenderAccount, [T_Email], TeacherBody)
#If email fails to send then a message pops up to inform the user
except:
    PoppingUp('Email Error', '''An attempt was made to send an email but it has failed.
              Please keep note of your meeting date and time''')
#exits and closes the connection to the server after emails are sent
EmailServer.quit()
```

Code Screenshot 8.21 - Second Part of the Email generating and sending function using SMTP protocol.

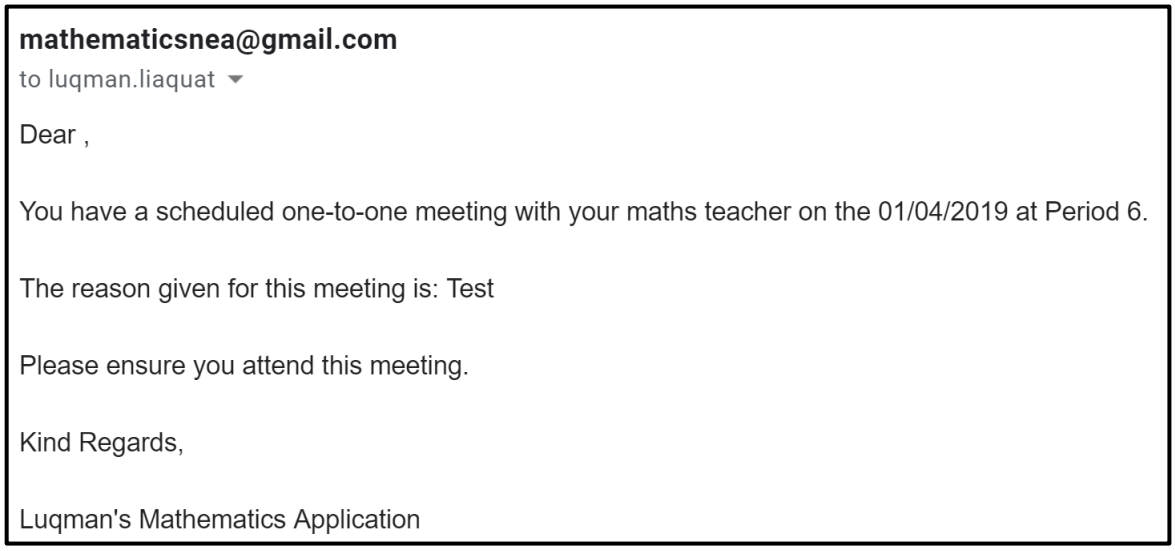
Alpha Testing – Meeting Email Sender

Ensuring that I have met all the requirements including the additional requirement set is vital, and the test table shows the outcomes of the alpha tests for the email sending functions. This maps to objective 8B.

Test Reference	Section	Input	Expected Output	Actual Outcome	Comments
7i	Main Meeting Scheduler	Carry Out Test 7d again, but this time the code for the email function has been added. The emails will be replaced with my own email addresses (as the test email addresses are just test accounts and are not registered with any email service)	This should search through the teacher's and student's timetable and find a time to create a scheduled meeting, with day 1 (Monday) being prioritised. A pop up detailing the data and period for this meeting will be shown as well. This will also email these details of the meeting to both the teacher and student with a personalised message using their names	Actual outcome and emails that were sent are shown on the next page. The emails were sent successfully and as expected.	Works As Expected

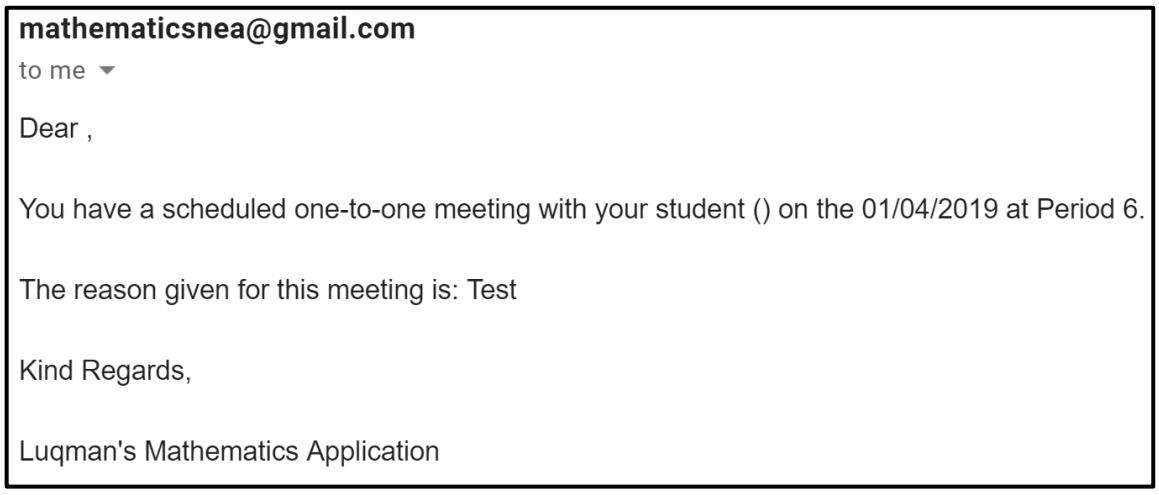
As the emails that were used in my test accounts did not exist, I used two of my own email addresses and ran that in the program. You will notice that the name is missing as the program attempted to search for the name associated with the email that I had given it but it wasn't a registered account, and therefore no name was returned. However, the two screenshots below show that the email functionality works exactly as intended and specified by the client and therefore fully meets the requirements for ***Objective 8***.

Test 7i - Student Email:



Code Screenshot 8.22 – Student received email from test 7i

Test 7i - Teacher Email:



Code Screenshot 8.23 – Teacher received email from test 7i