Design

Contents

Hand Designs

This page maps to Objectives 1 and 2

Main Colour Scheme will be blue as it represents a calming environment which is what revision should be (reducing stress for a student just before exams is vital in achieving better[6]).

Text fonts will remain simple but inviting such as Cambria Math, so users can easily read the information required to use the application. (This textbox is using the Cambria Math font)

For better user understanding of what the application can do, there are a list of features and details. This gets users prepared to use the application.

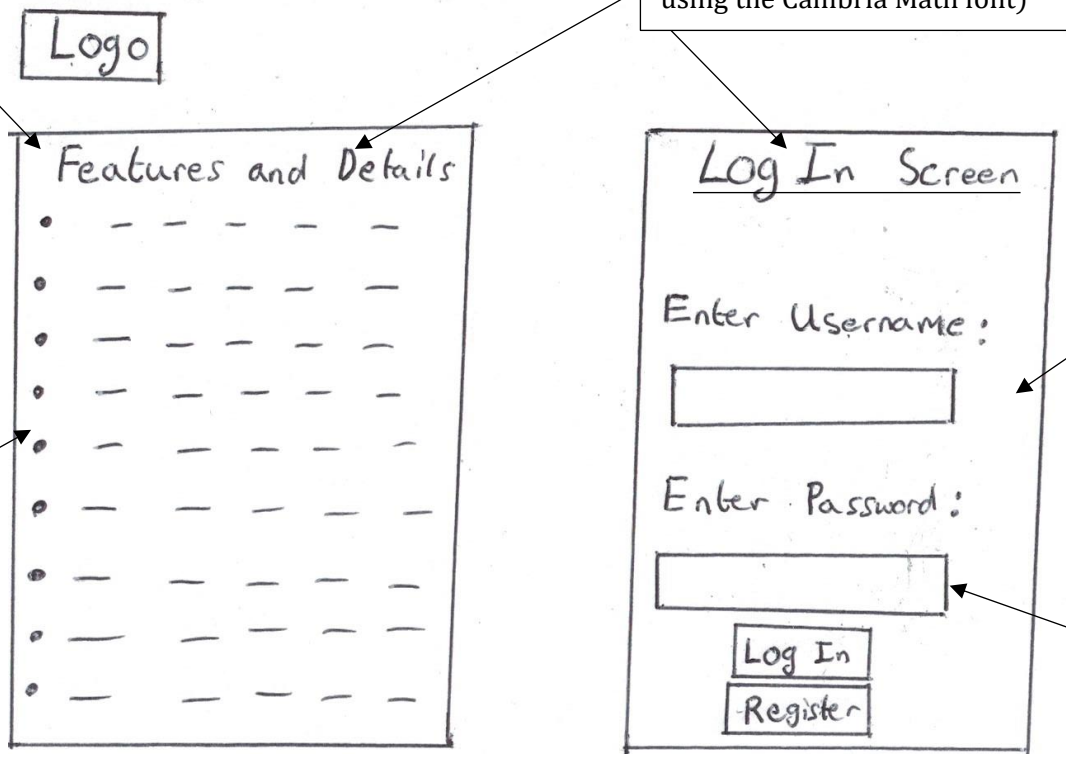


Figure 7A - Hand Design for Log in Screen

Simplified Home Screen within application allows user to easily log in or register for a new account easily. It is clear and easy-to-use which is an important aspect of Objective 1

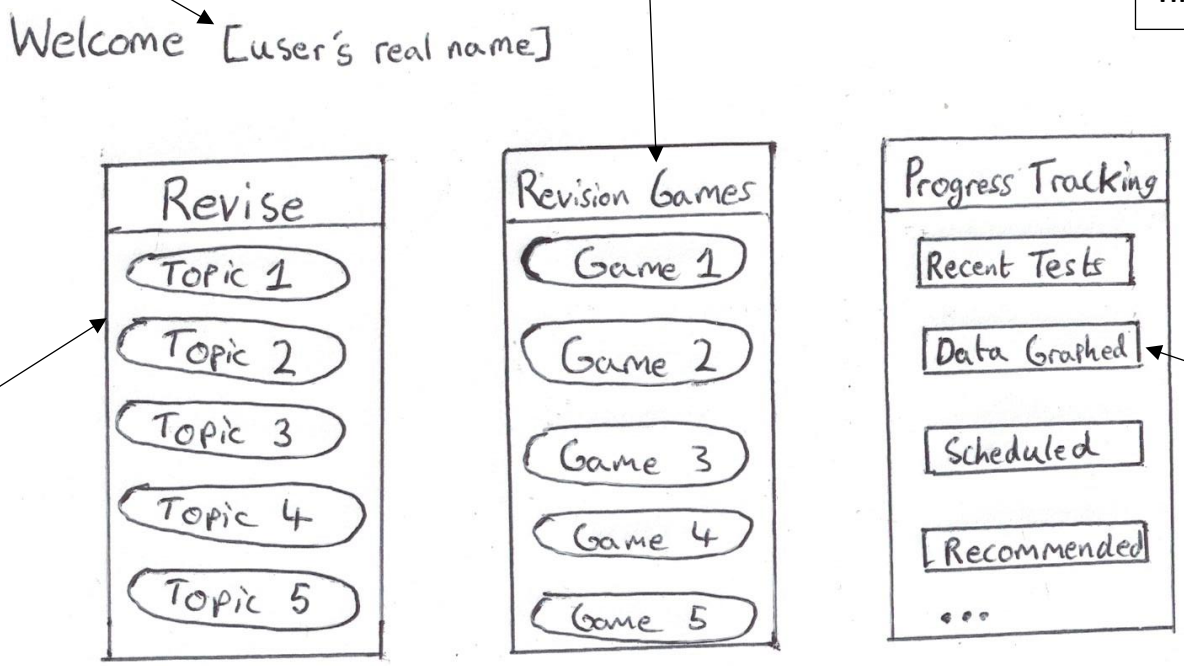
The entry field of the password will have asterisk replacement while users are typing their passwords to maintain privacy and security. This helps to meet a performance criterion of Objective 6

A personalised message with the user's name (not username).

Three distinct colours for each section will be used. These will, most likely, be blue, green and lilac respectively as they are more inviting than other colours available. This was suggested by my client and with further research[6,7], I have decided to use these colours. This allows me to meet the criteria of Objective 1.

This page maps to Objective 1

Topics clearly labelled in bubbles. The current labels are place holders and will have actual names while I am working on the technical solution of the application.



Each Option has its own process and output which will be useful to the user. For example, the 'data graphed' button will take the user's data from the database and report it in tables that the user can visually see. The scheduled option will allowing users to check if they are required to have a one-to-one meeting.

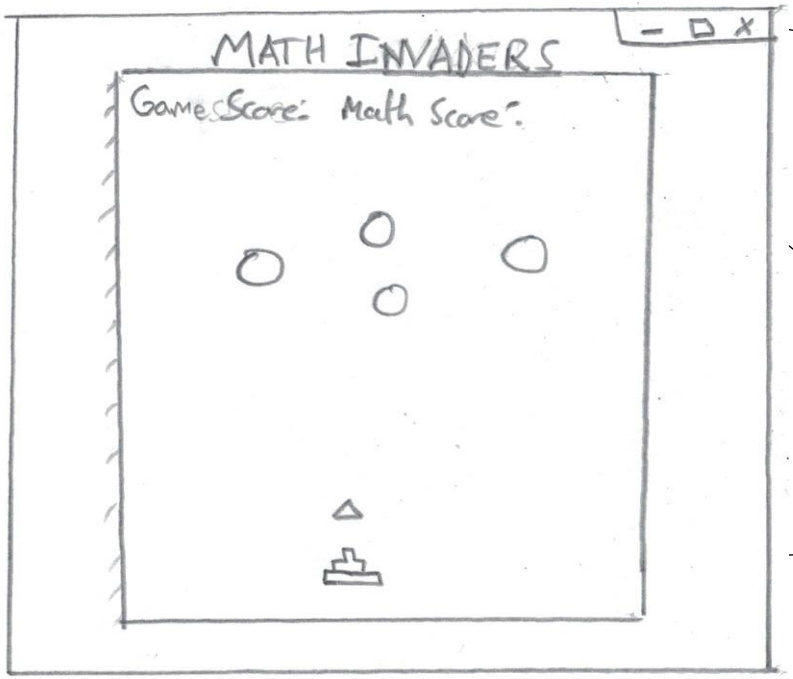
Figure 7B – Hand Design for Home Screen

Maths Score will be displayed at the top and as said before, this is the score gained by successfully answering a question.

Game Score is for the score gained whilst playing the actual Maths Invader game – by hitting aliens/

Aliens which will be constantly moving left and right and once reached a border, they will move down a level.

Triangle-Shaped Bullet



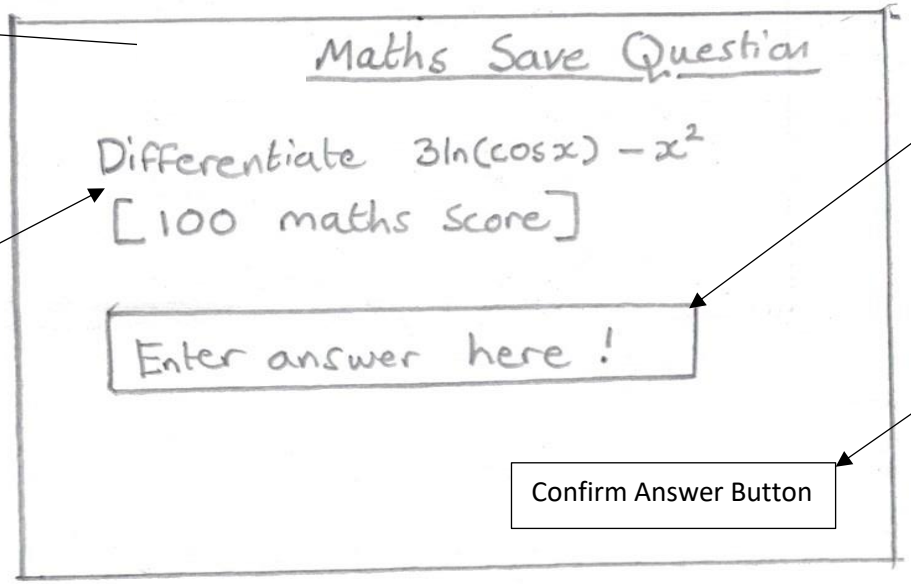
'Close' button will be used to end the game. To ensure students do not try stopping their score from being saved, e.g. if it is very low, the score will be updated in the database after every question attempted.

Title of the game is Maths Invaders

User controlled spaceship that will shoot the bullets and moves left or right depending on key press

This page maps to Objectives 1 and 4

The user will be asked 3 maths question after losing to save themselves and continue the game. The topics that a question can come up on are chosen by the user beforehand. This is a separate screen that will pop up.



Answer box for entry, there will not be multiple choice questions as that is not going to challenge students enough which is what my client wants.

Example question shown with the amount of maths score available. I have realised it will be useful to have different levels of scoring depending on difficulty to challenge users and improve their skills.

Confirm Answer Button

I added this button in afterwards as I realised that this functionality is imperative for clarity and ease-of-use in my GUI which is the criteria for Objective 1

Specific Colour Design

To match my client's requirements and meet my objectives, I will specify the colours that I will be using in different sections of the application. Although I have mentioned the names of colours already, there are vast shades of these colours and specifying with hex values will allow my client to know exactly which colours are being used. I used an online converter to acquire the values for the RGB and Hex format of each colour[8]. GUI Colours → Objective 1

Screen Name *(Objective)*	**Colour Name**	**RGB Code**	**Hex Code**	**Colour Preview**
Majority of Screens – Main colour scheme *(1)*	Light Blue	(100, 230, 255)	#64E6FF	
Log-In Entry Boxes *(2)*	White	(255, 255, 255)	#FFFFFF	
Log-In and Register Buttons *(2)*	Red	(255, 0, 0)	#FF0000	
Revise Section *(7)* *see below for amended colour from client feedback*	Lilac	(200, 160, 200)	#9B69DC	
Games Section *(4)*	Blue	(0, 150, 255)	#0096FF	
Progress Tracking *(6)*	Green	(0, 200, 100)	#00C864	
Home Section Text Colour *(1)*	White	(255,255,255)	#FFFFFF	

Screen Name	Colour Name	RGB Code	Hex Code	Colour Preview
Maths Invaders Game – Player Spaceship *(4)*	Red	(255, 0, 0)	#FF0000	
Maths Invaders Game – Aliens *(4)*	Green	(0, 200, 50)	#00C832	
Maths Invaders Game – Background *(4)*	Black	(0, 0, 0)	#000000	
Maths Invaders Game – Border *(4)*	Green	(0, 200, 50)	#00C832	
Maths Invaders Game – Game Score Text *(4)*	White	(255, 255, 255)	#FFFFFF	
Maths Invaders Game – Maths Score Text *(4)*	Orange	(255, 140, 0)	#FF8C00	

Client Feedback

Feedback on Hand Designs and Colour Choices Inbox ×

[redacted] 12:37 (13 minutes ago)

to me

Dear Luqman,

Your hand designs are what I had in mind – simplistic is key! In regard to your colour scheme, I agree with them all except the lilac tone you have chosen to use in the "Revise Section". Although I did suggest it to you, I think it isn't as full of colour as I would want it to be, if that makes sense. I would suggest choosing a slightly darker variant of that colour such as a violet tone.

After using the website that you suggested, I found that I prefer the RGB(200, 100, 200) tone of colour which uses less green than the colour you first proposed. I have also attached an image so there isn't any confusion to what I mean. Thank you!

Regards,

[redacted]

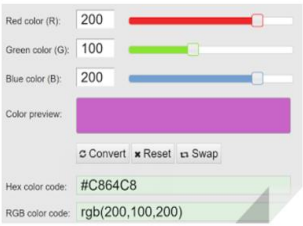
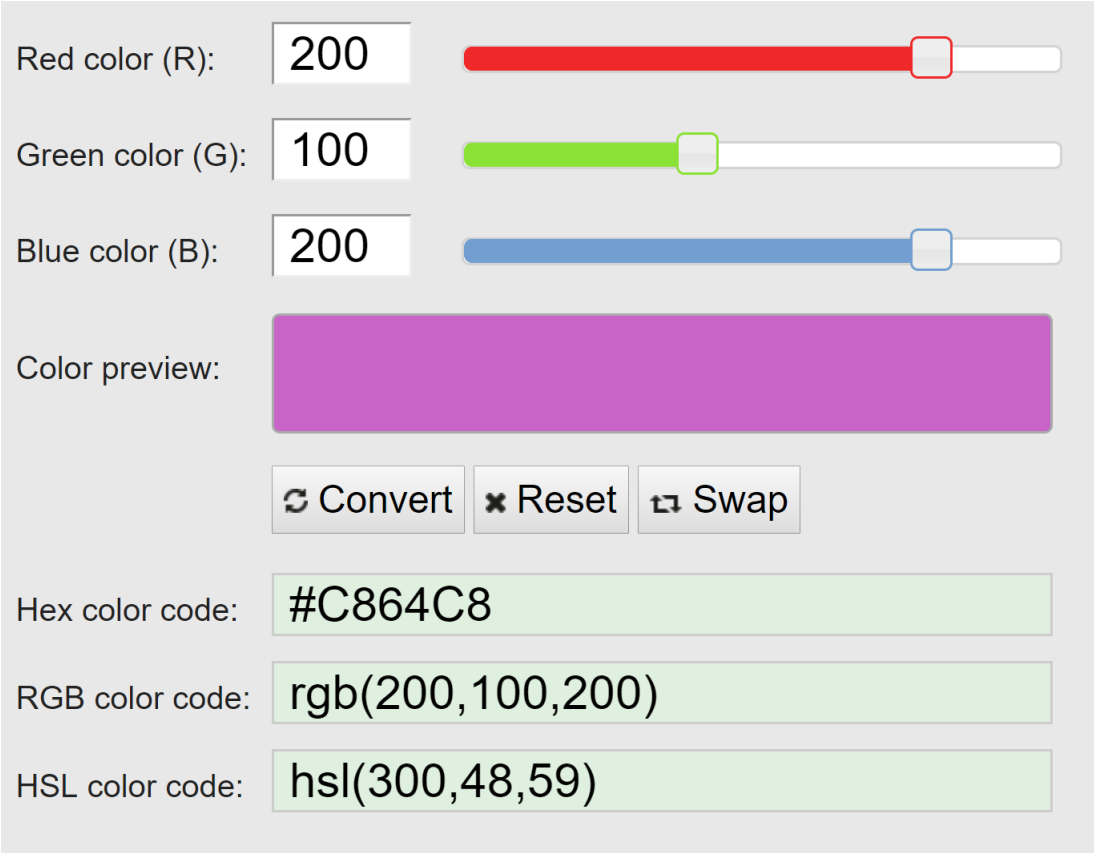


Figure 7D – Hand Design and Colour Design Feedback

From my client's feedback, I have amended the colour choice for the Revise Section. I have also included the image he sent to me of the colour design with the RGB and Hex values below.

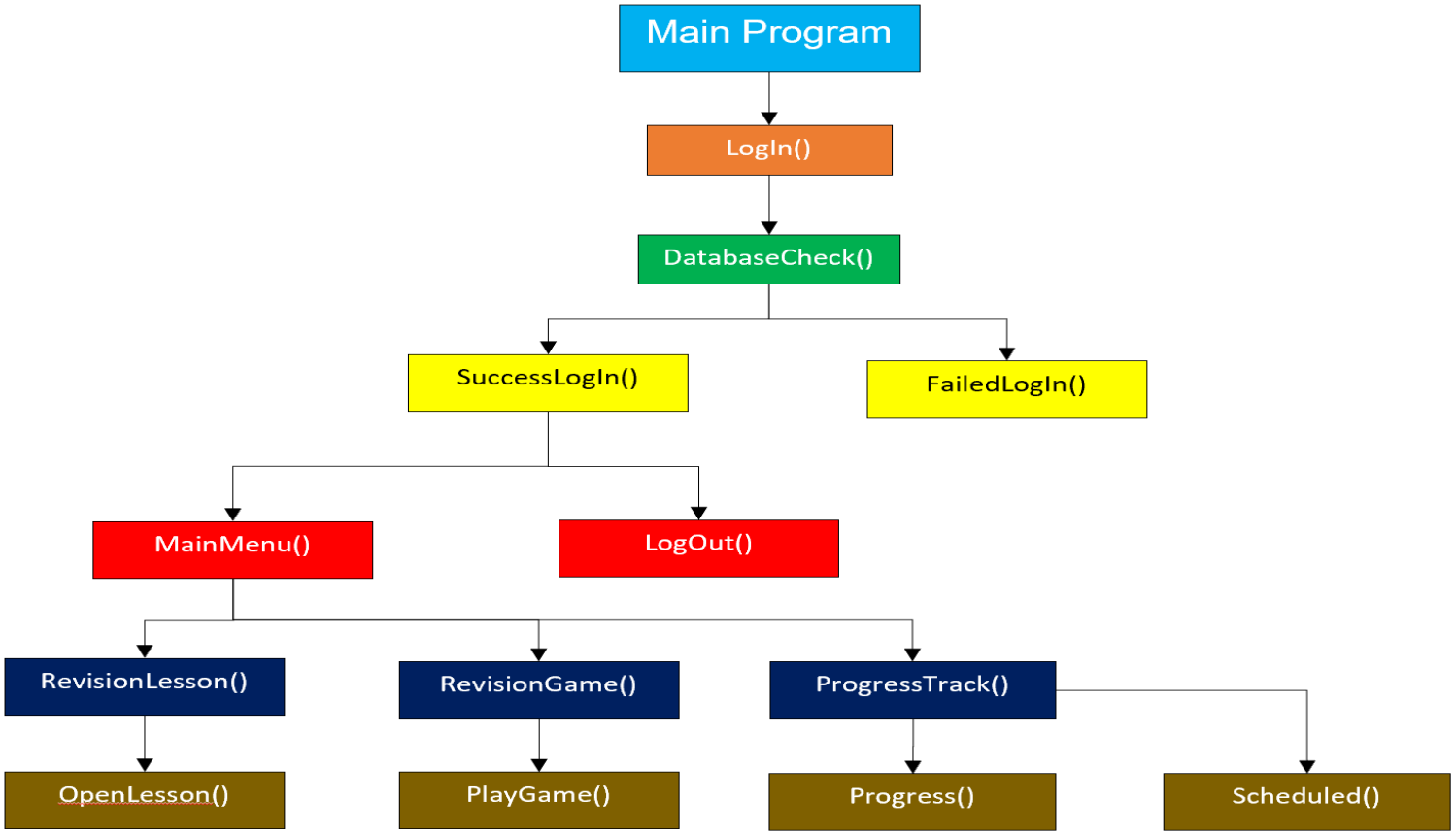


Amended:

Screen Name	Colour Name	RGB Code	Hex Code	Colour Preview
Revise Section	Violet	(200, 100, 200)	#C864C8	

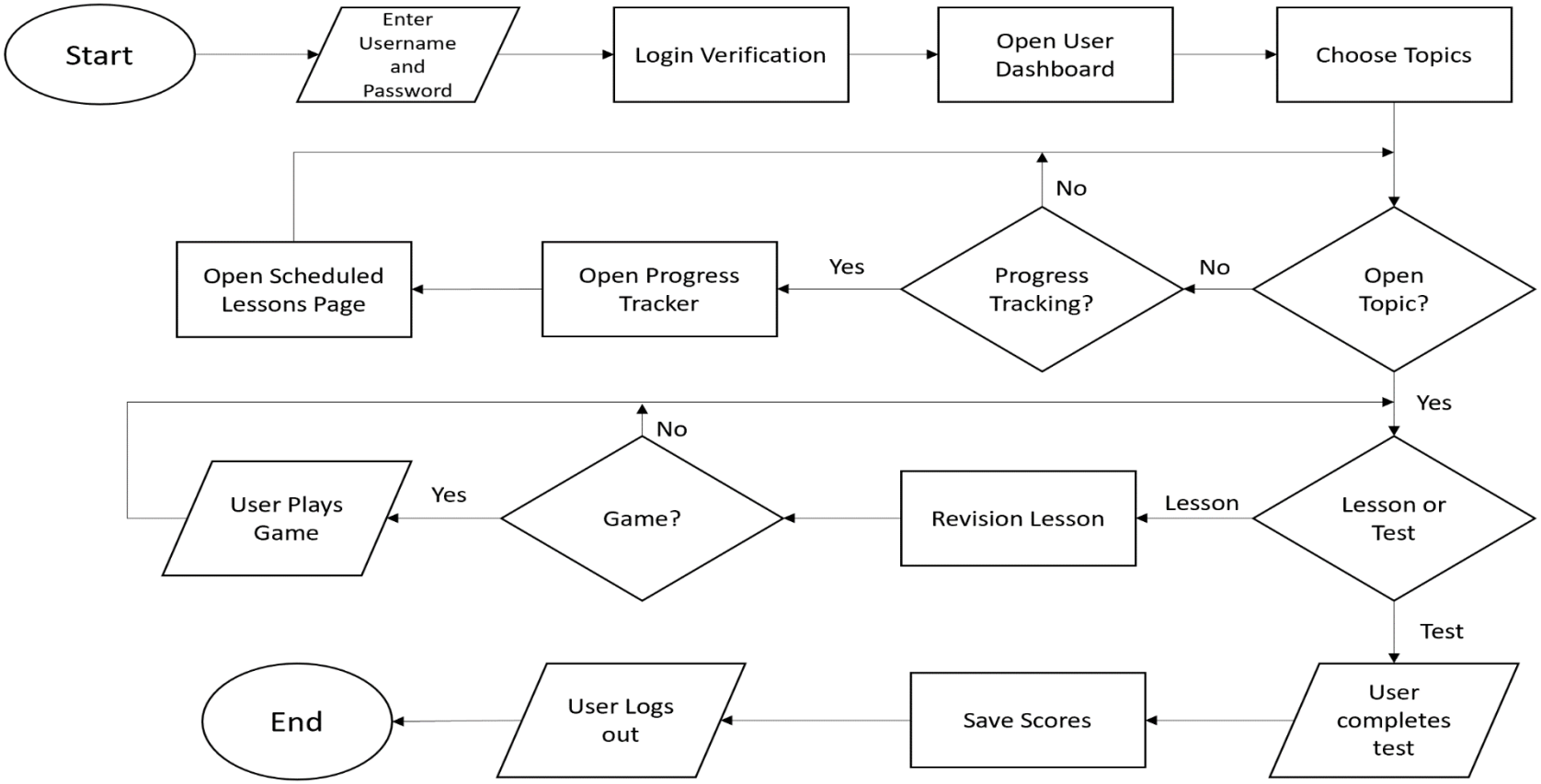
Hierarchy Chart

The Hierarchy chart below represents the likely procedures and functions used within the program and how each one is linked to the next stage of the program. Each level is clearly indicated using a different colour. The functions and procedures have also been named sensibly to allow for a better understanding of how the program will work.



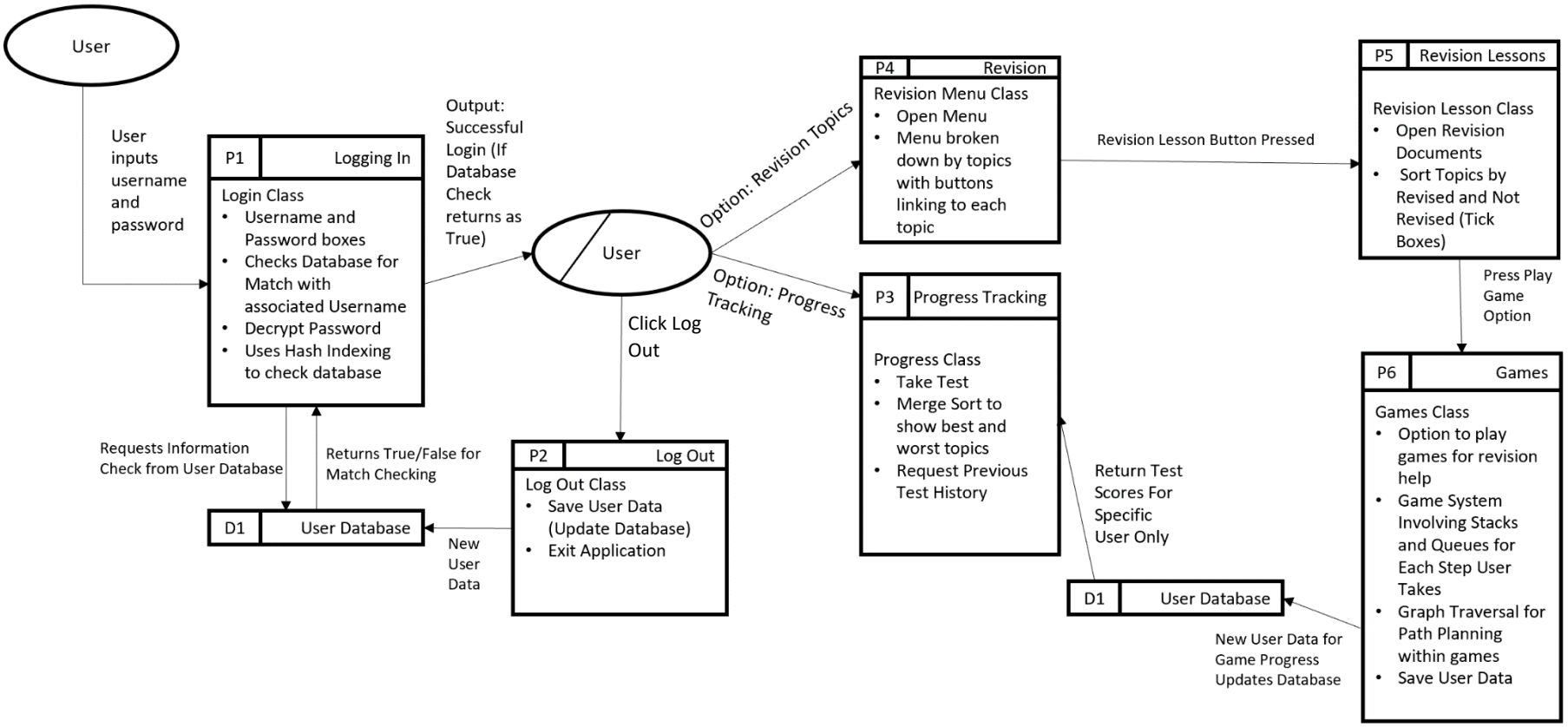
Flow Chart

The flowchart below is another crucial stage in my design process as it means I can accurately define how I want my program to work in terms of the steps the user takes, along with which processes should occur after a user input or a decision made. Please note that the logout process can occur at any point, but I have placed it at the end as that makes sense logically. This flowchart also gives a generalised take on the program and, of course, different users may have different interactions with the program, but this is to provide a strong sense of how the program will be expected to function. Each process will be fully explored and explained in the 'Pseudocode and Flowchart' sections at the end of the 'Design' Section.



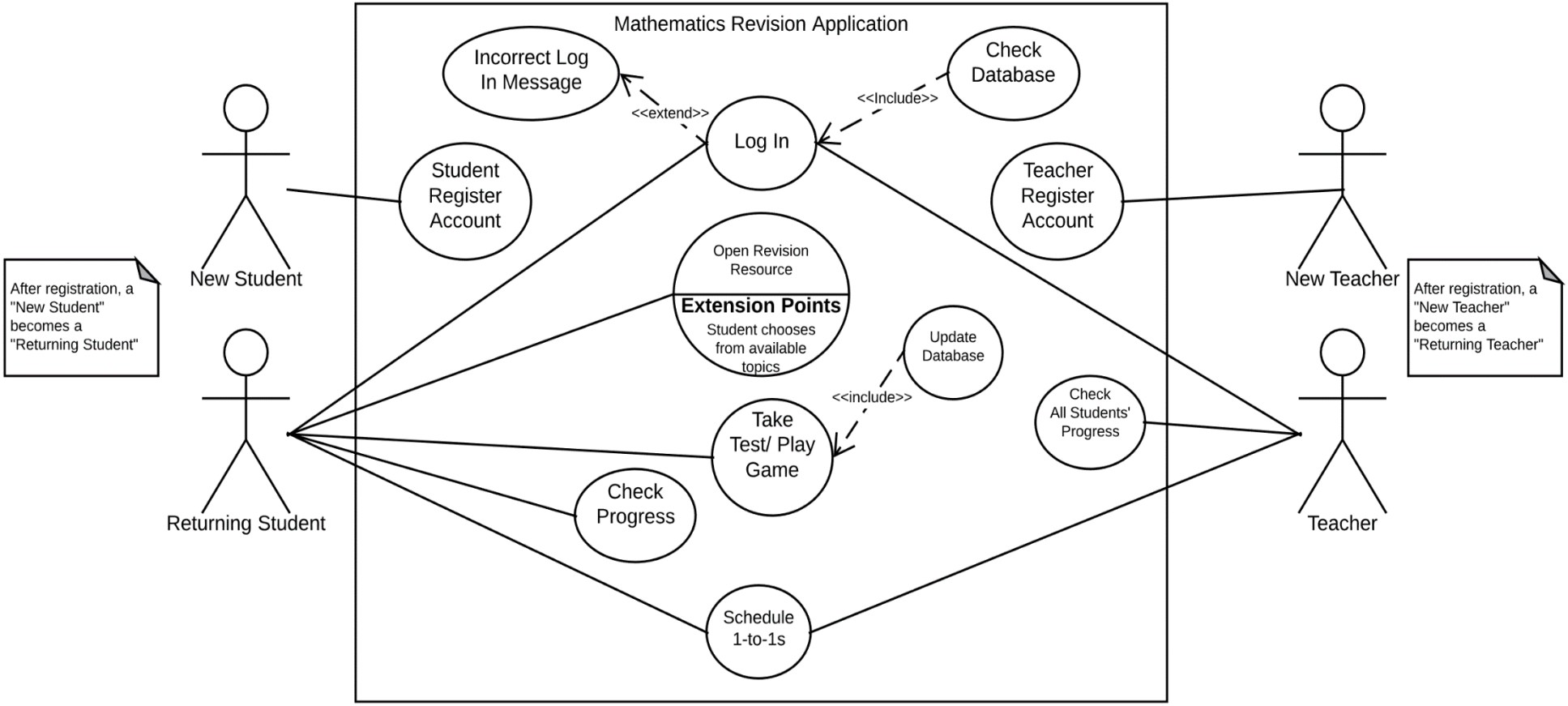
Data Flow Diagram

The following data flow diagram provides a summary of the broader processes that will take place within the proposed application and how the flow of data will initiate these. It also compounds upon the flowchart to show more information within the different sections outlined above.



Use Case Diagram

Creating a use case diagram is significant in my design process as it allows me to show a simple summary of how the different users (actors) interact with my program. It also provides a brief idea of the relationships within the system and creates a reliable guideline for the links between actors or lack thereof.



I/O Diagram

Understanding how a system will take inputs, process them, store them and output something meaningful is fundamental to all programs and is a good starting point in designing how I want the program to be. This is only a small subset of the program's features. The Inputs, Processes, Storage and Outputs will be extensively explored in the Pseudocode and Flowchart section at the end of the Design Section.

Inputs	**Processing**
1) The user attempts to log in by entering their username and password in the correct fields 2) Click a revision topic within the options available 3) Written Input to Answer Questions 4) Arrow Keys 5) Teacher Inputs Name of Student in Progress Tracker	1) Checks if the username and the hashed version of user inputted password matches values in the database 2) Check which topic button has been pressed 3) Keyword matching and comparing answers 4) Checks which key is pressed and the function assigned to that particular key 5) Queries relevant data which is related to that user in the database
Storage	**Output**
1) Stores inputted usernames and passwords in variables, and account usernames and hashed password in the database 2) Stores location of topic area/ name of topic area screen 3) Store user answer and correct answer in different variables 4) Storing the functions and key bindings for a particular arrow key 5) Stores the result of the query, so any data that is returned to the program from the database	1) Login Successful: Open Welcome Page or Unsuccessful: Retry 2) Open Specified Topic Area 3) A statement informing the user if the answer is correct or incorrect 4) Movement on the screen of the player in the direction of arrow 5) Displays the requested data about the particular student to the teacher in a table form

Data Dictionary Table

Understanding the inputs and how I will validate them is incredibly important before I program as it means that I have a clear structure and know what types of data I would be using. Doing this for the existing system was useful due to the similarities, it gives me a stronger sense of which data types I need to include. For my system, I will be looking at how I could validate this data (if possible) and sample data for each section and objective that I have. Objective 1 will use Kivy KV Buttons, Text Labels, Shapes and other design elements to create a well-designed GUI. This table will be thorough but not completely extensive as I will likely add new data items and structures while working on the technical solution.

Section ***(Objective)***	**Data Item** ***(Description)***	**Data Type/ Structure**	**Validation/ Limits**	**Sample Data**
Database *(5)*	Cursor *(To Run SQLite Queries)*	SQLite Cursor	Running Query	(ID, Name)
Database *(5)*	Connection *(To connect to SQLite Database)*	SQLite Database Connection	Running Query	Tables
Register *(2)*	Create UserEmail *(To store user's email as username for a new account)*	String	Limit Length to 60	'Luq@test.com'
Register *(2)*	Create UserPass *(To store user's password to be hashed for a new account)*	String	No Limits	'Password'
Register *(2)*	Check UserPass *(To store user's input of password for second time to confirm the password)*	String	Must match CreateUserPass	'Password'
Register *(2)*	First Name *(To store user's first name for a new account)*	String	Limit to 12 characters, no numbers and symbols	'Luqman'
Register *(2)*	Last Name *(To store user's last name for a new account)*	String	Limit to 12 characters, no numbers and symbols	'Liaquat'
Register *(2)*	Status *(To store whether the user is a student or teacher)*	String	Can only be one of: 'Student' 'Teacher'	'Student'

Register *(2)*	Hashed Password *(To Store the Hashed version of the user's password in the database)*	String	64 Characters Hash	'A3EF24…'
Register *(2)*	New IDs *(To Store the new incremented ids for the new user in the database tables)*	Integer	Must be an integer and cannot be the same as a previous ID	4
LogIn *(2)*	Taken Emails *(To check which emails are in the database already)*	List	List Operations	('Luq@test.com')
LogIn *(2)*	Username Input *(To store the user's input of the username)*	String	Limit Length to 60	'Luq@test.com'
LogIn *(2)*	Password Input *(To store the user's input of the password)*	String	No Limits	'Password'
LogIn *(2)*	AssociatedPassword *(To store the hashed password in the database associated with inputted email)*	String	64 Characters Hash	'A3EF24…'
Revision – Calculus *(4,7)*	Input Number *(To calculate the factorial of user's input)*	Integer	Below 4 Digits	20
Revision – Calculus *(4,7)*	Factorial Number *(To store the factorial of user's input)*	Integer	Any number of digits but must be a factorial number	120
Revision – Calculus *(4,7)*	User Equation *(To store the user-defined mathematical equation to plot graph)*	String	Must involve an x	'2**x'
Revision – Calculus *(4,7)*	Stored Equations *(To store all user inputted equations for one session on the stack)*	Stack	LIFO Data Structure	('2**x', '2x^2')

Revision – Calculus *(4,7)*	Start Value *(To store user-defined start value for equation to be plotted on a graph)*	Float	No limitations	0
Revision – Calculus *(4,7)*	End Value *(To store user-defined end value for equation to be plotted on a graph)*	Float	Must be greater than Start Value	100
Revision – Trigonometry *(4,7)*	Coordinates *(To store user-defined Coordinates for plotting a triangle on graph)*	Float	Coordinates cannot be the same	(3,4)
Revision – Trigonometry *(4,7)*	Degrees *(To store user inputted degrees value to convert to radians)*	Float	No Limitations	180.0
Revision – Trigonometry *(4,7)*	Radians *(To store user inputted radians value to convert to degrees)*	Float	No Limitations	3.14
Revision – Statistics *(4,7)*	Data Set *(To store list of randomly generated numbers)*	List	Must be exactly 10 integers in list each being less than 100	[10, 24, 45, 32, 25, 21, 67, 89, 34, 87]
Revision – Statistics *(4,7)*	Average *(To store the average of the list of randomly generated numbers)*	Float	No Limit	50.4
Revision – Statistics *(4,7)*	Variance *(To store variance of list of randomly generated numbers)*	Float	No Limit	10.3
Revision – Statistics *(4,7)*	Probability *(To store user inputted probability for binomial distribution)*	Float	0 <= P < 1	0.75

Revision – Statistics *(4,7)*	Number of Trials *(To store user inputted number of trials for binomial distribution)*	Integer	No Limits	23
Topic Test *(3)*	Topics *(To store which topics user wants to take the test on)*	List	Elements must be strings and match topics offered by the application	['Differentiation', 'Integration', 'Trigonometry']
Topic Test *(3)*	Questions *(To store the questions that will be asked)*	List	Elements must be strings and match topics offered by the application	["Differentiate x^8"]
Topic Test *(3)*	Answers *(To store the corresponding answers for the questions that will be asked)*	List	Elements must be strings and match topics offered by the application	["8x^7"]
Topic Test *(3)*	User Answer Input *(To store the users' inputted answers for the question that is asked)*	String	No limitations	'100'
Maths Invader Game *(4)*	Game Score *(To store the users' Game Score)*	Integer	Multiples of 100	400
Maths Invader Game *(4)*	Maths Score *(To store the users' Maths Score)*	Integer	Multiples of 100	400
Maths Invader Game *(4)*	Motion *(To store the users' key presses for movement)*	Boolean	True or False for each arrow key and spacebar	True
Maths Invader Game *(4)*	Distance *(To store the distance between the spaceship and aliens to check for collisions)*	Float	Must be greater than or equal to 0	10

Maths Invader Game *(4)*	Leaderboard *(To store and display the overall scores of students that have played the game)*	List	Elements will contain strings for names and integers for scores	[‘Luqman’, 100, 300]
Progress Tracking *(6)*	Overall *(To store the overall results for a student in all topics)*	List	List of both Floats	[1, 2, 30, 24]
Progress Tracking *(6)*	Best *(To store the best topic and results for a student)*	List	Elements contain Integers and Strings	[‘Differentiation’, 20, 25]
Progress Tracking *(6)*	Worst *(To store the worst topic and results for a student)*	List	Elements contain Integers and Strings	[‘Integration’, 7, 25]
Progress Tracking *(6)*	Percentages *(To store the percentage of correct answers given for each topic)*	List	Elements are Integers between 0 and 100	[20, 90, 64, 73]
Schedule *(8)*	Timetable *(To store the user’s timetabled lessons for each day)*	List	Must contain 5 days (Monday-Friday) Only	Monday – [‘Maths’, Computer Science’, …]
Schedule *(8)*	Scheduled *(To store the user’s scheduled meetings)*	String	Contains Meeting day and Period	‘Thursday Period 2’
Schedule *(8)*	Date *(To store the date of the user’s scheduled meetings)*	DateTime	Must be in the future from when the meeting was scheduled	01/01/2019
Schedule *(8)*	Emails *(To store the emails of the parties involved in the meeting to send them a notification)*	String	Must be stored in the database already	‘Luqman@ Email.com’

Data Volumes Table

The use of a data volumes dictionary is helpful in ensuring that I know how frequently data from each object within my system is used. Along with my previous charts and tables, this helps me to produce a better image of how I expect my final product to look and operate with the data objects, such as classes and functions, and how frequently they are used being defined clearly. It is possible that not all data objects are listed, or any that are listed may not be included, but that will be shown throughout the development of my technical solution and testing.

Data Object (Class)	Data Object (Functions)	Volume of Data + Description
Login Screen	LogIn	Every time a student or teacher wants to log into their account
Register Screen	CreateAccount	when a new user needs to register an account
Topic Test	Topics	Works out which topics the questions should be based on every time a user starts a new topic test
	Questions	Works out which question needs to be asked and finds the corresponding answer for it – used multiple times throughout one topic test
	UpdateDatabase	Inserts data about user's topic test after every question answered
	Check Answer	Checks whether the user input is correct and updates variables to reflect the outcome; it is used every time a question is answered
Stack	Push	Pushes an element onto the top of stack each time the function is called
	Pop	Removes element on top of stack each time the function is called and returns it to be used again
	View	Returns the stack to be used every time function is called
	Check Empty	Checks if the stack is empty by comparing it with an empty list every time function is called

Calculus Revision	Factorial	Calculates Factorial of a number each time user presses Factorial button
	Get Graph	Takes in user-defined equations every time Graph button is pressed
	Plot Graph	Plots graph from user-defined equations every time Graph button is pressed and when Undo is pressed
	Find Gradient	Calculates gradient of user-defined equations every time Gradient button is pressed
	Find Area	Calculates the area of user-defined equations every time Area button is pressed
	Undo	Pops off the current user-defined equation and displays the previous user-defined equation, runs the Plot Graph function
Trigonometry Revision	Plot Triangle	Plots triangle using user inputted coordinates every time Plot Triangle button is pressed
	Convert Degrees	Converts user inputted degrees value into a radian value every time Convert Degrees button is pressed
	Convert Radians	Converts user inputted radian value into a degree value every time Convert Radians button is pressed
Statistics Revision	Generate Random Data	Generates a list of 10 random integers every time the statistics revision is opened
	Statistics Calculator	Calculates average, variance and standard deviation of random data set every time Statistics Calculator button is pressed
	Binomial Distribution	Outputs a probability based on Binomial distribution every time the user enters required parameters and presses Binomial Distribution button

	Normal Distribution	Outputs a probability based on Normal Distribution every time the user enters required parameters and presses Normal Distribution button
Maths Invaders	Movement	Moves in the direction of arrow keys every time an arrow key is pressed whilst the game is running
	Fire Bullet	Fires a bullet every time a user presses the space bar key on the keyboard whilst the game is running
	Collision Detection	Checks if an alien and bullet or an alien and spaceship has collided throughout the game whilst it is being run
	Build Game	Creates the space around the game including the background and colours when the Play Maths Invaders button is pressed
	View Leaderboard	Shows the user the leaderboard of the game scores every time the View button is pressed
	Merge Sort Leaderboard	Sorts the leaderboard with highest Maths Score at the top using an efficient Merge sort algorithm every time the View Leaderboard function is run
Scheduler	Create Meeting	Creates a scheduled meeting every time the user inputs the required data and presses Create Meeting button
	Prevent Clash	Checks for any existing meetings and prevents another meeting being created on top of an existing one – every time Create Meeting function is run
	Send Email	Sends an email about a meeting to student and teacher every time a new meeting has been created after Create Meeting is run

Progress	Overall	Displays overall data on a student or class every time Overall Button is pressed
	Best	Displays student's best topic with data to explain every time Best button is pressed
	Worst	Displays student's worst topic with data to explain every time Worst button is pressed
	Topic Specific	Displays data on a specific topic, such as differentiation, when user inputs a topic name and presses Explore Topic button

Database Design

To succeed with all my objectives, it is necessary that I implement a database. To ensure that my database is as efficient as possible, I am creating the entity relationship diagrams and the tables I will be using, which will be normalised. This entire section maps to ***Objective 5.*** The database will help me achieve all my other objectives as the storage of data is necessary to complete each objective. A database brings all the data in one logical place that can be used at any time and is separate from the main program which is useful as there will be significantly less or even no data errors as the database will be able to manage these tasks using my SQLite Queries.

Table and Query Design

An integral part of my application will be the use of tables within databases. Designing both the table structure and the SQL queries that I will be using beforehand will allow me to gain a stronger understanding of how data should be analysed which will be essential in my program. The teacher will have access to data from all the tables detailed below for each student in their class (except for the hashed password field in the *User* Table).

User Table

Attribute Name	Email (PK)	FirstName	LastName	Hashed Password	StudentStatus
Data Type	Unique Varchar(60)	Varchar(12)	Varchar(12)	Char(64)	Boolean

SQL Query **=**
```
CREATE TABLE 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)
);
```

To be identified uniquely, the primary method of logging in will be through the use of an email. This reduces data redundancy as there is no need for a User ID when I can uniquely identify records using the email. This will also be useful in sending scheduled alerts as I do not need an extra column to store these details. The StudentStatus column is a Boolean data type and refers to if the user is a student or teacher: if it is set to TRUE, then the account is a Student, if it set to false then the account is a Teacher. The hashed password will use the Sha256 algorithm and will be stored in Hex meaning it will always use 64 characters.

The User table maps to multiple objectives because of its relationships, but it directly maps to ***Objective 2****.*

Topic Table

Attribute Name	QuestionID (PK)	Question	Answer	Marks	Difficulty	Email (FK)
Data Type	Unique Integer	Varchar(255)	Varchar(50)	Integer	Integer	Varchar(60)

SQL Query =
```
CREATE TABLE 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)
    FOREIGN KEY(Email) REFERENCES User(Email)
);
```

The Topic table is where each of the questions for each topic will be stored along with their respective answers and marks. It maps directly to ***Objective 3***. Each question will also be given a difficulty level which will be given by my client as he will be able to judge this well. To avoid confusion, there will be a range of unique identifiers to know which topic each question falls under within the primary key column 'QuestionID'. See the table below:

QuestionID Range	**Corresponding Topic**
100-199	Differentiation
200-299	Integration
300-399	Statistical Analysis
400-499	SUVAT
500-599	Trigonometry

Progress Table

Attribute Name	ProgressID (PK)	DiffCorrect	DiffTotal	IntCorrect	IntTotal
Data Type	Unique Integer	Integer	Integer	Integer	Integer
Attribute Name	StatCorrect	StatTotal	SuCorrect	SuTotal	TrigCorrect
Data Type	Integer	Integer	Integer	Integer	Integer
Attribute Name	TrigTotal	AvgDifficulty	Email (FK)		
Data Type	Integer	Float	Varchar(60)		

SQL Query =
```
CREATE TABLE 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)
);
```

This table is vital to store how the student is progressing across all topics with in-depth data on each topic. To allow for targeted feedback, there are individual fields for each topic referring to the number of correct answers the student has given and the total number of questions they have answered for each topic. There is no 'incorrect' field or overall 'percentage correct answers' field as these can be calculated from the data already stored. This reduces any unnecessary storage and is essential to allow for more data to be stored especially if the number of users grows. There are 3 levels of difficulty with **1** being the easiest and **3** denoting the harder questions. This is why the average defaults at a value of 1.

This table will be used for ***Objectives 3 and 6.***

Timetable Table

Attribute Name	TimeID (CPK)	DayNumber (CPK)	Period 1	Period 2	Period 3	Period 4	Period 5	Period 6	Email (FK)
Data Type	Integer	Integer	Boolean	Boolean	Boolean	Boolean	Boolean	Boolean	Varchar(60)

SQL Query =
```
CREATE TABLE 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)
);
```

Within this table, the day and the availability of the student will be stored. This means that it will not store the lessons they have, but if they have a lesson or not. This will make it easier for the scheduler to decide if the teacher and student are free to schedule a one-to-one meeting. It will use a Composite Key with TimeID and DayNumber. The DayNumber refers to what day of the week it is with ***1*** representing Monday and ***5*** representing Friday.

Scheduled Table

Attribute Name	ScheduleID (PK)	ScheduleDay	SchedulePeriod	Reason	Email (FK)
Data Type	Unique Integer	Integer	Integer	Varchar(150)	Varchar(60)

SQL Query =
```
CREATE TABLE 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)
);
```

The scheduled table will hold essential information on the automatic schedules the program has made. As with the above table, the Day will be an integer from 1 to 5. Each record in this table will be able to provide the user with when their meeting is and for what reason. The email information and relationship with the user table will allow for a higher level of personalisation in the emails. It will also be able to trace back the meeting to the specific user. Both the Timetable table and Scheduled table will be used for ***Objective 8.***

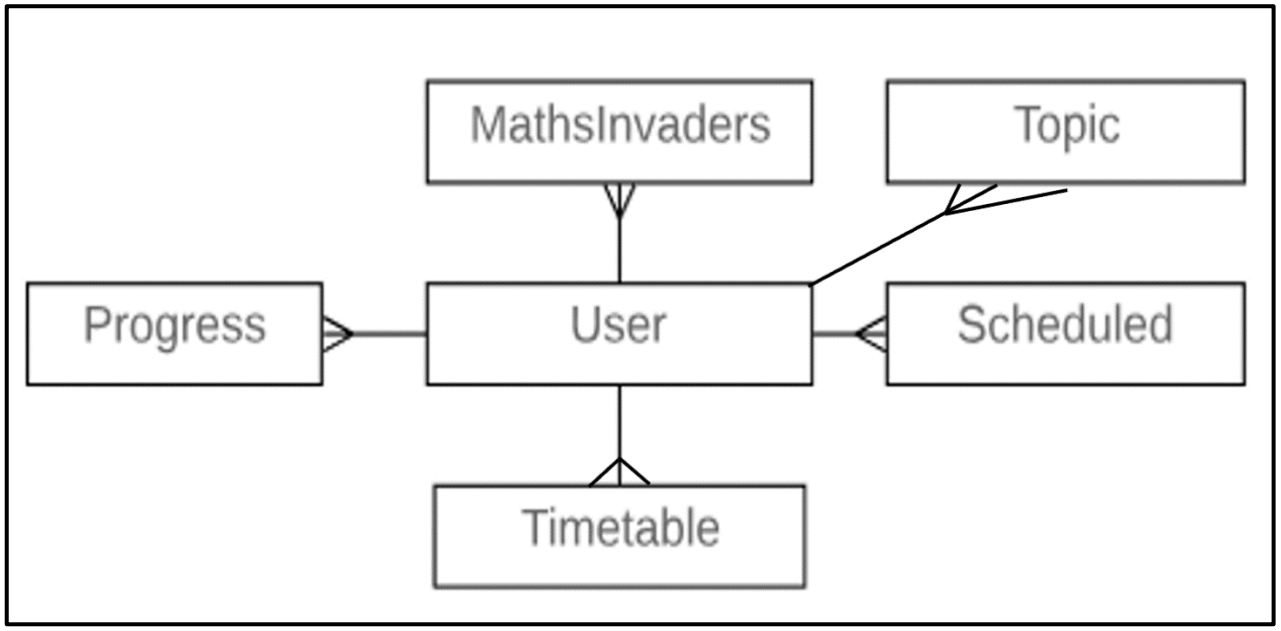
MathsInvader Table

Attribute Name	GameID (PK)	Email (FK)	ProgressID (FK)	GameScore	MathScore
Data Type	Unique Integer	Varchar(60)	Integer	Integer	Integer

SQL Query =
```
CREATE TABLE 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)
);
```

The MathsInvader Table will store critical details on the outcome of a Maths Invader Revision Game attempt if the user chooses to play it to aid revision. This will store the game score and the Maths Score. The GameID will be used to identify any single attempt at the game uniquely. This table will be used for ***objective 4***

Entity Relationship Diagram



- Each User can have more than one game of MathsInvaders and therefore it is a ONE-TO-MANY RELATIONSHIP
- Each User can take more than one maths test and therefore it is a ONE-TO-MANY RELATIONSHIP
- Each User can have more than one scheduled meeting and therefore it is a ONE-TO-MANY RELATIONSHIP
- Each User will have multiple timetables to cover all days they attend school and therefore it is a ONE-TO-MANY RELATIONSHIP
- Each Question will have one user who answered it last but more one user may have many questions that they were the last ones to answer so it is a ONE-TO-MANY RELATIONSHIP

Test Tables

To ensure that my technical solution meets my objectives as required, and the code works correctly, I must test the system with various inputs. To do this effectively, I am going to create several test tables for my objectives and algorithms that will be used. These will be used while working on the technical solution. Designing these now will allow me to rigorously analyse what my system is expected to do compared to what it actually does when I have inputted test data. After this stage, I will write the pseudocode for the algorithms I will be using to complete each set objective. I will highlight the 'Actual Outcome' and 'Comment' columns in blue to indicate these will be filled out when testing my technical solution. I will also use colours/bold text to highlight exactly what the input into the system is if there is additional text describing the input. The **Purple text** will be used to represent inputted text in any GUI fields. The **Blue text** will be used for names of Buttons on the GUI. The **Green text** will refer to physical buttons that are pressed such as those on the keyboard or mouse. The **Orange text** will represent actions which may include more than one input – such as "collision between ship and alien."

A list of the Test Tables and which aspect of the application they correspond to is shown below:

Test Table 1: *Database and Tables (Objective 5)*

Test Table 2: *Log In and Register System (Objective 2)*

Test Table 3: *GUI (Objective 1)*

Test Table 4: *Maths Topic Test (Objective 3)*

Test Table 5: *Maths Invaders Game (Objective 4)*

Test Table 6: *Progress Tracker (Objective 6)*

Test Table 7: *Scheduler (Objective 8)*

Test Table 8: *Revision Areas (Objective 7)*

Test Table 1 – Database and Tables

For success in *Objective 5*, I will need to test and verify that my database works as expected. This means that I will be testing several inputs for each table. Please note that the 'input' column within this test table will be used in conjunction with the 'INSERT INTO' SQL syntax. Also, the input column will specify which fields within the table are being referred to (which will be italicised within brackets next to the input). This will clarify my Expected Output comments and reasoning.

Test Reference	**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.		
1b	User	**luq@test.com** *(Email)*, **Luqman** *(FirstName)*, **Liaquat** *(LastName)*, **A5DE35B…** *(HashedPassword)*, **TRUE** *(Student_Status)*	This input covers all the required fields, and the table should update successfully (when the hashed password is written fully)		
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.		
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		
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.		
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.		
1g	Timetable	**2** *(TimeID)*, **Luq@test.com** *(Email)*, **2** *(DayNumber)*, **Maths** *(Period1)*, **Maths** *(Period2)*, **Physics** *(Period3)*, **Computing** *(Period4)*,	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		

		Computing *(Period5)*	therefore will cause an error.		
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 scheduled day and schedule period fields should not have numbers above 5. A new constraint will need to be added.		
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 successfully.		
1j	Progress	**1** *(ProgressID)*, **10** *(DiffCorrect),* ***11*** *(DiffTotal)*	These inputs will update the table successfully; however, it is possible that no email is associated with a record as there is no NOT NULL constraint used. This will need to be added.		
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.		
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.		
1m	MathsInvader	**2** *(GameID),* **Luq2@test.com** *(Email)* **2** *(ProgressID),* **5** *(GameScore),* **300** *(MathScore)*	This set of inputs should result in no errors.		

Test Table 2 – Log In and Register System

My *Objective 2* relies on a working account system. This features the ability for new users to register an account or for existing users to log back into their accounts. This requires me to prepare to test out the possible inputs that a student may type within this system and be prepared for any errors. A general description of where the input is happening/what the input refers to is italicised and in brackets.

Test Reference	Section	Input	Expected Output	Actual Output	Comments
2a	Register	"**Luq@test.com**" *(Enter Email)*	Moves onto creating a password (as Email is used as a username)		
2b	Register	"**Luq@test.com**" *(Enter Email)*	As this username as already been created, the program should notify the user to try a different one.		
2c	Register	"**Password**" *(Create password),* "**Password**" *(Retype password)*	The password and the retyped password match so this should allow the registration to continue.		
2d	Register	"**Password**" *(Create password),* "**NotPassword**" *(Retype password)*	The password does not match with the retyped password, so the program should ask the user to retry.		
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.		
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 the details they have typed may be incorrect.		

Test Table 3 – GUI

To succeed in *Objective 1*, I must test out the core features of the GUI that will be created. This test will check the buttons that will be used in the GUI, and the basic input/output interface the user will be presented with throughout their experience of using this application. An important note to make is that Test Table 3 will not check if other algorithms are working as expected, as separate tests will be made for those objectives. For example, one test may check if the 'Log In' button is working, i.e. it runs the LogIn() function. However, it will not test if the LogIn() function works as expected.

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		
3b	Log In Screen (Figure 4A)	Type "**Test**" into '*Password*' field	"********" will be displayed in the 'Password' field		
3c	Log In Screen (Figure 4A)	Press "**Login**" Button	The function defined for Logging in will run: LogIn()		
3d	Log In Screen (Figure 4A)	Press "**Register**" Button	The function defined for new user registration will run: Register()		
3e	Home Screen (Figure 4B)	Press "**Topic 1**" Button	Specific Topic 1 page will open.		
3f	Home Screen (Figure 4B)	Press "**Maths Invader**" Button	Maths Invader Game will be launched		
3g	Home Screen (Figure 4B)	Press "**Progress**" Button	User Progress Page will be opened		
3h	Any Screen	Press "**Minimise**" or "**Maximise**" buttons	The application will minimise or maximise accordingly		

Test Table 4 – Maths Topic Test

To meet the criteria set out in *Objective 3*, I need to carry out a test on the maths topic tests that will be available to the users. This will test out how answers are inputted and if the necessary procedures are carried out based on the result of the answer, such as a score update and progress update.

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.		
4b	Differentiate $5x^2$	**10X**	This will not be accepted without using a .lower() function. As this is still the correct answer, this .lower() function will need to be included to ensure the marks and progress tables are successfully updated.		
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.		
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 is accepted as a correct answer, giving the correct answer procedures.		
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.		

Test Table 5 – Maths Invaders Game

Objective 4 requires me to produce at least one game that will aid revision for the mathematics students. This game has been decided as a space invaders game from the Analysis section. To ensure that the game functions as required and therefore meets the objective, it will need to be thoroughly tested. This is why the need for a prepared test table is essential.

Test Reference	Input	Expected Output	Actual Outcome	Comments
5a	**LEFT Arrow**	The user ship should move left.		
5b	**RIGHT Arrow**	The user ship should move right.		
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		
5d	**SPACEBAR**	This should shoot the bullet from the ship		
5e	**Collision Between Ship and Alien**	This should end the round and open up the maths question.		
5f	**Collision Between Bullet and Alien**	This should increase the Game Score by 100		
5g	**Play Game and Answer Maths Question Correctly**	This should increase the Maths Score by 100		
5h	Press **View Leaderboard**	This should display the leaderboard for the games played		

Test Table 6 – Progress Tracker

An important aspect of my program will be the progress tracking features including the tables on the user data that will be generated. These features will need to be tested to ensure that I meet the criteria of *Objective 6*.

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		
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		
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		
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 of the topic		
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		
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)		

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		
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.		
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		
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		
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.		

Test Table 7 – Scheduler

My seventh test table will test the functionality of the scheduler aspect of the program. This will include if email alerts work successfully along with automatic scheduling when both the student and teacher are available. This is used for *Objective 8*.

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		
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		
7c	Timetable	Press **View Timetable**	This should change to the View Timetable screen and show the user's timetable in a clear table with headings		
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.		

7e	Main Meeting Scheduler	Repeat Test 7d with precisely 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		
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		
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		
7h	Main Meeting Scheduler	Enter "3**2**" 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.		
7i	Main Meeting Scheduler	Carry Out Test 7d again, but this time the code for the email function has been added.	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. This will email these details of the meeting to both the teacher and student with a personalised message using their names		

Test Table 8 – Revision Sections

For completion of objective 7, I must test out the different tools that I provide to the user to aid their revision in each of the three topics that I have decided to do. The topic that a test is referring to is clearly shown in the 'Revision Topic' column.

Test Reference	Revision Topic	Input/Action	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		
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**		
8c	Calculus Graphs	Leave all fields blank, and press **Calculate**	Error Pop Up should tell the user to fill in all fields		
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		
8e	Calculus Graphs	Enter "**2**" in 'Gradient Point' field then press **Gradient**	The gradient should be calculated from the x^2 function (from the previous test) at x=2 and therefore pop up with an answer of **4**		
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		
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		

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.		
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		
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		
8k	Trigonometry	Type "**180**" into 'Degrees' Field then press **Convert**	This should output the first few digits of pi as 180 degrees = pi		
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.		
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.		
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.		

8o	Statistics	Press the **StatCalc** button	A popup displaying the mean, variance and standard deviation of the set of data should be displayed.		
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		
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		
8r	Statistics	Type "**38**" into 'Normal Distribution' Field and then press **Normal PD**	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.		
8s	Statistics	Leave all fields empty and press **Binomial PD**	This should notify the user to enter a value in the required fields		
8t	Statistics	Leave all fields empty and press **Binomial CD**	This should notify the user to enter a value in the required fields		
8u	Statistics	Leave all fields empty and press **Normal PD**	This should notify the user to enter a value in the required field		

Hardware and Software Requirements

My proposed design for the creation of a revision application will not require specific kits or pieces of hardware for both me (the developer) or a student/teacher (the end user). This means that a computer, most likely running the Windows operating system, will be sufficient for a user to run the program with a limited strain on its hardware components (such as the CPU and GPU). Users who will be utilising the games section of the revision application may need speakers for sound output to correctly work (found in most computers/laptops already). Finally, the basics in hardware will be necessary for the user to use the program to its full potential including a working keyboard and mouse/touchpad. At this moment in time, the support for multi-touch touchscreens will not be considered to allow the successful production of the program before moving onto additional enhancements.

In terms of software, the student will require my main application. The user will not need additional software, such as Flash, to utilise the entire application.

System Integrity and Security

My proposed application will hold essential data on several different users, including personal details such as their age, name and grades/marks they have achieved within mathematics. This data is private, and that means security measures to protect this data will be vital. One method through which this data will be secured is through limited access, meaning that users will need to log into the system with their own username and password before having any access to their data. The passwords will be encrypted to add another layer of security and ensure a lower risk of 'hacking' and data theft.

System integrity will also be necessary for the scalability of my proposed application because the ability to function correctly, despite changes in the amount of data or users, is a substantial aspect of successful applications already on the market such as Integral.

Pseudocode/Flowcharts

This section will include the pseudocode/flowcharts for the core program. This will be to design the code before writing it or further breaking down the processes that should be included in the code when I begin writing it. As it is only a design, the pseudocode/flowcharts will not be exactly the same in terms of structure or techniques used but will give a good outline of the code. There will be several changes made to these initial code designs throughout the technical solution and testing phase of the project. I will split the pseudocode/flowcharts by each objective in order, to ensure that I have a complete plan and design for each of the sections that will need to be produced to meet the client's requirements.

Objective 1

Objective 1 is related to general GUI features including colours, buttons and simplistic design of the application for the user to enjoy.

GUI

As the GUI will require Kivy to complete, and the designs have already been created in this section, I will not write pseudocode for it but will instead have a flow chart for the general process of the GUI. This has already been completed and is shown in the Design Section (entitled 'Flow Chart').

Objective 2

Objective 2 is the account system objective which requires each user to have their own unique account so that they can access specific and personalised features. Therefore, the registration and login systems are integral to meet this objective.

Registration

```
FUNCTION CreateAccount():
    LIST ExistingUsernames ← SELECT Emails FROM User TABLE IN DATABASE
    FirstName← User Input from GUI Field
    LastName ← User Input from GUI Field
    EmailAddress ← User Input from GUI Field
    Password ← User Input from GUI Field
    RetypedPass ← User Input from GUI Field
    StudentStatus ← User Input from GUI Field
    IF ANY FIELD EMPTY THEN DO:
        OUTPUT "You have not filled in all the information; please try again!"
    ELSE DO:
        IF EmailAddress IN ExistingUsernames THEN DO:
            OUTPUT: "This email address already exists in this system! Try another one."
        ELSE DO:
            IF Password != RetypedPass THEN DO:
                OUTPUT "Passwords do not match, try again!"
            Else DO:
                Password ← Hash(Password)
                SQLite: INSERT INTO USER TABLE VALUES (FirstName, LastName,
                EmailAddress, Password, StudentStatus)
                OUTPUT "Account Successfully Created"
                CHANGE SCREEN TO Login Screen
```

Logging In

```
FUNCTION LogIn():
    LIST Emails ← SQLite: SELECT Emails FROM User TABLE IN DATABASE
    UserCheck ← USER TEXT FROM USERNAME GUI INPUT FIELD
    IF UserCheck IN Emails THEN DO:
        PassCheck ← USER TEXT FROM PASSWORD GUI INPUT FIELD
        PassCheck ← HASH (PashCheck)
        PassChecked ← SQLite: Select HashedPasswords FROM User TABLE IN DATABASE
        IF PassCheck == PassChecked THEN DO:
            OUTPUT "Login Successful"
            CHANGE SCREEN TO Home
        ELSE DO:
            OUTPUT "Your password does not match our records! Try Again"
```

Objective 3

Objective 3 is the practice maths topic tests section. This is where the student will choose which topics they would like to be tested on, and the questions will be displayed for them to answer.

Topic Test

```
CLASS TopicTestScreen():
  FUNCTION TestTopics():
    TOPICS ← LIST
    CheckTrig ← GUI FIELD INPUT
    CheckCalc ← GUI FIELD INPUT
    CheckStats ← GUI FIELD INPUT
    IF NOT CheckTrig AND NOT CheckCalc AND NOT CheckStats THEN DO
      RETURN False
    ELSE DO
      IF CheckTrig THEN DO
        TOPICS ← SQLite:SELECT QuestionID FROM Topic WHERE QuestionID BETWEEN 500 AND
        599
      IF CheckCalc THEN DO
        TOPICS ← SQLite: SELECT QuestionID FROM Topic WHERE QuestionID BETWEEN 100 AND
        299
      IF CheckStats THEN DO
        TOPICS ← SQLite: SELECT QuestionID FROM Topic WHERE QuestionID BETWEEN 300 AND
        399
      RETURN TOPICS
  FUNCTION UpdateDatabase(Outcome):
    CONNECT TO DATABASE
    GET QuestionID, Correct, Total
    IF Outcome == 'Success' THEN DO
      Correct ← Correct + 1
    IF QuestionID >= 100 AND QuestionID <=199 THEN DO
      SQLite: INSERT INTO DiffCorrect, DiffTotal
            VALUES(?,?) WHERE Email=? : (*Correct*, *Total*, *UserCheck*)
    CHECK OTHER TOPICS USING CODE ABOVE AND RESPECTIVE IDS
    DISCONNECT FROM DATABASE
```

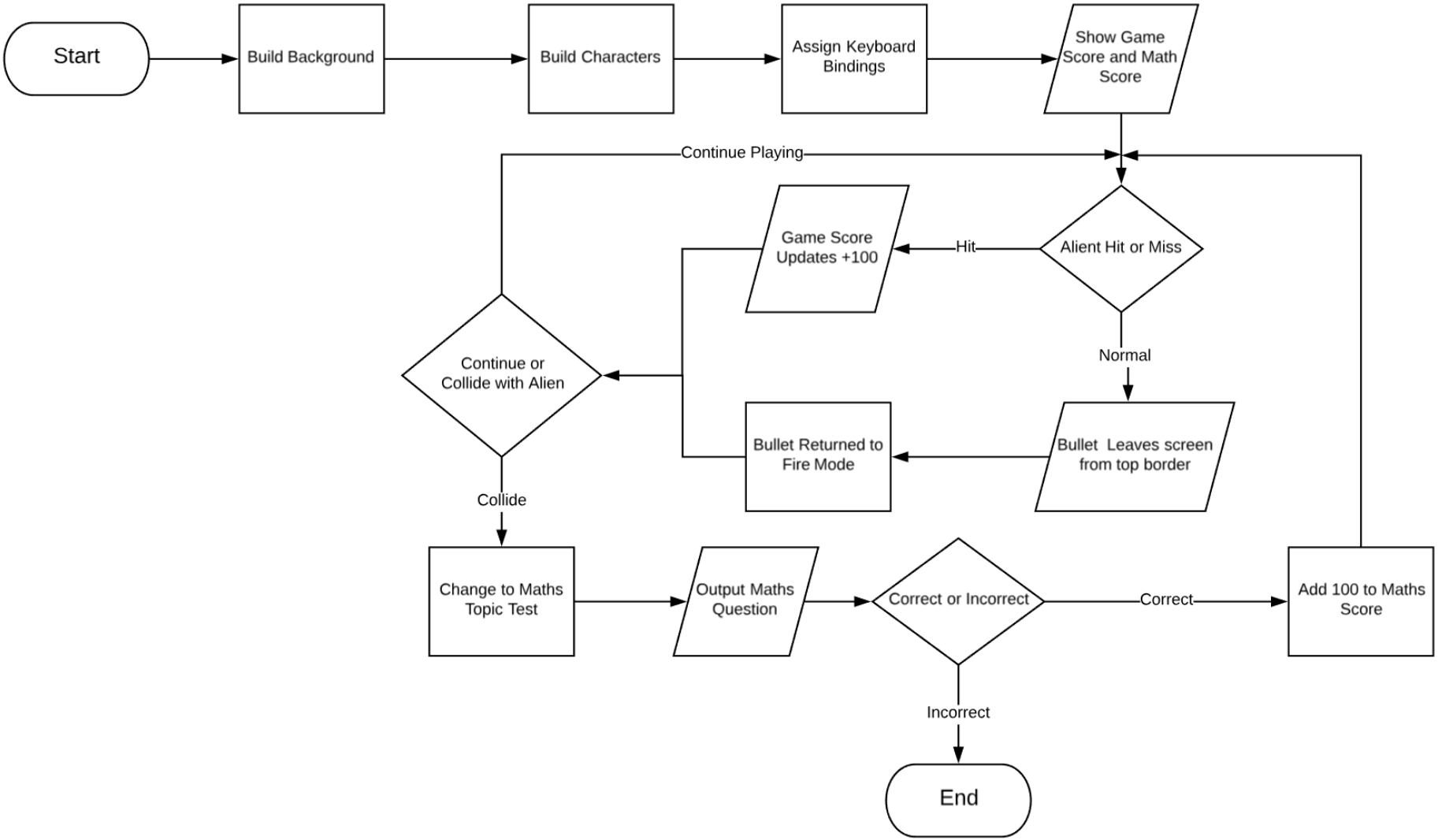
Objective 4

Objective 4 covers the Maths Invaders game and the efficient leaderboard sorting system that is required by my client.

Leader Board Sorting

```
NEW DICTIONARY points

points = {'A':10,'B':20,'C':40,'D':70,'E':30,'F':5,'G':25}

DEFINE MergeSort(LeaderSort):

  IF LENGTH LeaderSort > 1 THEN

    Midpoint = (LENGTH LeaderSort) INT DIVISION 2

    left = LeaderSort[0 to Midpoint]

    right = LeaderSort[Midpoint to Last Element]

    MergeSort(left)

    MergeSort(right)

    left_pointer = right_pointer = final_pointer = 0

    WHILE left_pointer <  LENGTH (left) and right_pointer <  LENGTH (right) DO

      #compares the term in the left to the term in the right list

      IF left[left_pointer] < right[right_pointer] THEN

        LeaderSort[final_pointer] = left[left_pointer]

        left_pointer = left_pointer + 1

      ELSE DO

        LeaderSort[final_pointer]=right[right_pointer]

        right_pointer = right_pointer + 1

      final_pointer = final_pointer + 1

    WHILE left_pointer < LENGTH (left) DO

      LeaderSort[final_pointer]=left[left_pointer]

      left_pointer = left_pointer + 1

      final_pointer = final_pointer + 1

    WHILE right_pointer < LENGTH (right):
       LeaderSort[final_pointer]=right[right_pointer]
      right_pointer = right_pointer + 1
      final_pointer = final_pointer + 1
   RETURN LeaderSort
ELSE DO
    RETURN LeaderSort
```

Main Maths Invaders Game



Objective 5

Objective 5 covers all of the database operations from creating the database to the queries involved in each of the other objectives. As I have already created all the queries for the initial database creation in the 'Database Design' section , I will not place them in the Pseudocode/Flowcharts section. Queries relating to an objective, such as the storing of user email and password in the database, will be placed in that objective's Pseudocode/Flowchart area below. Some pseudocode specific to objective 5 is included below.

Database Creation System

Connection ← CONNECT TO DATABASE 'Mathematics.db'

crsr ← CREATE CURSOR

SQLite crsr Execute: (---*This will contain the scripts already written in the Database Design Section*---)

COMMIT CONNECTION

CLOSE CONNECTION

Increment ID

Using sql aggregate function to update a new unique ID value

```
FUNCTION Increment ID(TABLE):
  MaxList ← SELECT MAX(TableID) FROM Table
  MaxID ← MaxList[0]
  IF MaxID == None or MaxID == '' THEN DO
    NewID ←1
  ELSE DO
    NewID ← INTEGER(MaxID) + 1
  RETURN NewID
```

Objective 6

Objective 6 is related to all the vital progress tracking features that my client wanted. This has strong links to Objective 5 as this requires the database throughout each process.

Data Retrieval

```
FUNCTION GetAll(Email):

    CONNECT TO DATABASE

    SQLite: SELECT DiffCorrect, DiffTotal, IntCorrect, IntTotal, StatCorrect, StatTotal, TrigCorrect,
TrigTotal, AvgDifficulty

                FROM Progress WHERE Email=?''', (*Email*))

    STORE ALL VALUES IN VARIABLES FROM QUERY

    DiffPercent ← INTEGER((diffCorrect/diffTotal) * 100)

    IntPercent ← INTEGER ((intCorrect/intTotal) * 100)

    StatPercent ← INTEGER ((statCorrect/statTotal) * 100)

    TrigPercent ← INTEGER ((trigCorrect/trigTotal) * 100)

    RETURN DiffPercent, IntPercent, StatPercent, TrigPercent, AverageDif


FUNCTION General(ProgressEmail, CorrectName, TotalName):

    RESULTS ← crsr.EXECUTE(SCRIPT 1)

    ResultHeadings ← ['FirstName', 'LastName', 'Total Correct', 'Total Answered']

    OUTPUT GRID: HEADINGS ← ResultHeadings, Data ← SQLite crsr Query Return

    DISCONNECT FROM DATABASE
```

SQL Scripts

For objective 6, each function that will be associated with a button will require a specific SQLite script that is run. The rest of the function will follow similar outputs to the General function. I have included the SQLite Queries that are needed to achieve the level of detail that my client wanted in the progress tracking. This involves the joining of tables to produce a well-informed output. This also maps to objective 5. As there will be two parts of the progress tracker, one being the specific student part and the other being the teacher part where a teacher can view data on any student, I have included possible scripts for both. The SQLite queries have been written in a different colour to make it clearer – blue for the student progress queries and green for the teacher progress queries.

All, Best and Worst Topics

Student: **SELECT DiffCorrect, DiffTotal, IntCorrect, IntTotal, StatCorrect, StatTotal, TrigCorrect, TrigTotal, AvgDifficulty FROM Progress WHERE Email = (Student's Username)**

Teacher: **SELECT FirstName, LastName, DiffCorrect, IntCorrect, StatCorrect, TrigCorrect FROM Progress INNER JOIN User ON Progress.Email = User.Email**

Differentiation

Student: **SELECT DiffCorrect, DiffTotal FROM Progress WHERE Email = (Student's Username)**

Teacher: ***SELECT FirstName, LastName, DiffCorrect, DiffTotal FROM Progress INNER JOIN User ON Progress.Email = User.Email AND Progress.Email = (Teacher-Inputted Student Username)***

Integration

Student: **SELECT IntCorrect, IntTotal FROM Progress WHERE Email = (Student's Username)**

Teacher: **SELECT FirstName, LastName, IntCorrect, IntTotal FROM Progress INNER JOIN User ON Progress.Email = User.Email AND Progress.Email = (Teacher-Inputted Student Username)**

Trigonometry

Student: **SELECT TrigCorrect, TrigTotal FROM Progress WHERE Email = (Student's Username)**

Teacher: **SELECT FirstName, LastName, TrigCorrect, TrigTotal FROM Progress INNER JOIN User ON Progress.Email = User.Email AND Progress.Email = (Teacher-Inputted Student Username)**

Statistics

Student: **SELECT StatCorrect, StatTotal FROM Progress WHERE Email = (Student's Username)**

Teacher: **SELECT FirstName, LastName, StatCorrect, StatTotal FROM Progress INNER JOIN User ON Progress.Email = User.Email AND Progress.Email = (Teacher-Inputted Student Username)**

Summations (To find totals for entire class)

Teacher: **SELECT SUM(*TOPIC NAME*) FROM Progress**

Objective 7

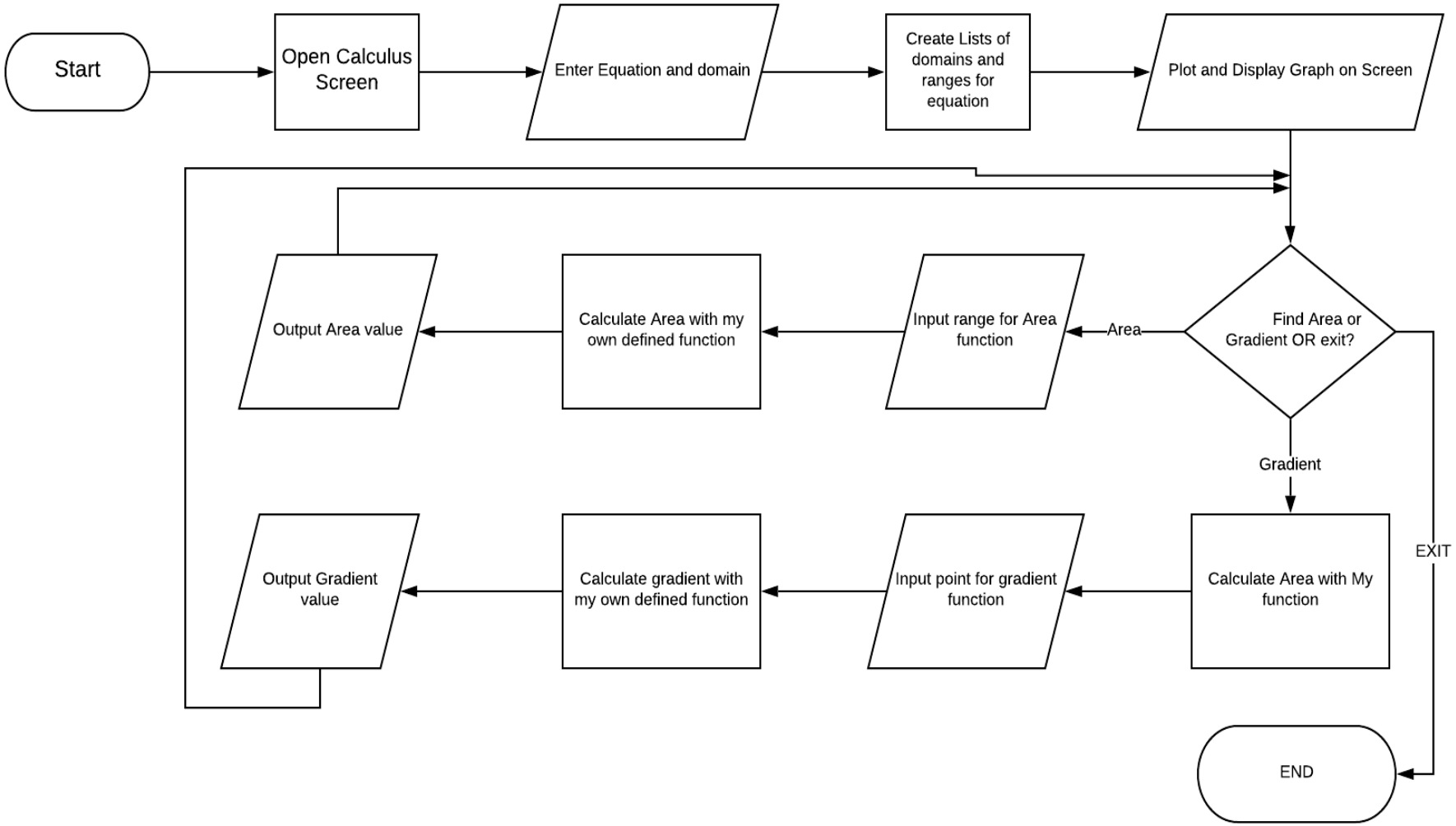
Objective 7 requires three major revision topics to be covered and have interactive features for students to learn effectively. The topics are Calculus (Which covers both Integration and Differentiation Topics), Trigonometry and Statistics.

Calculus

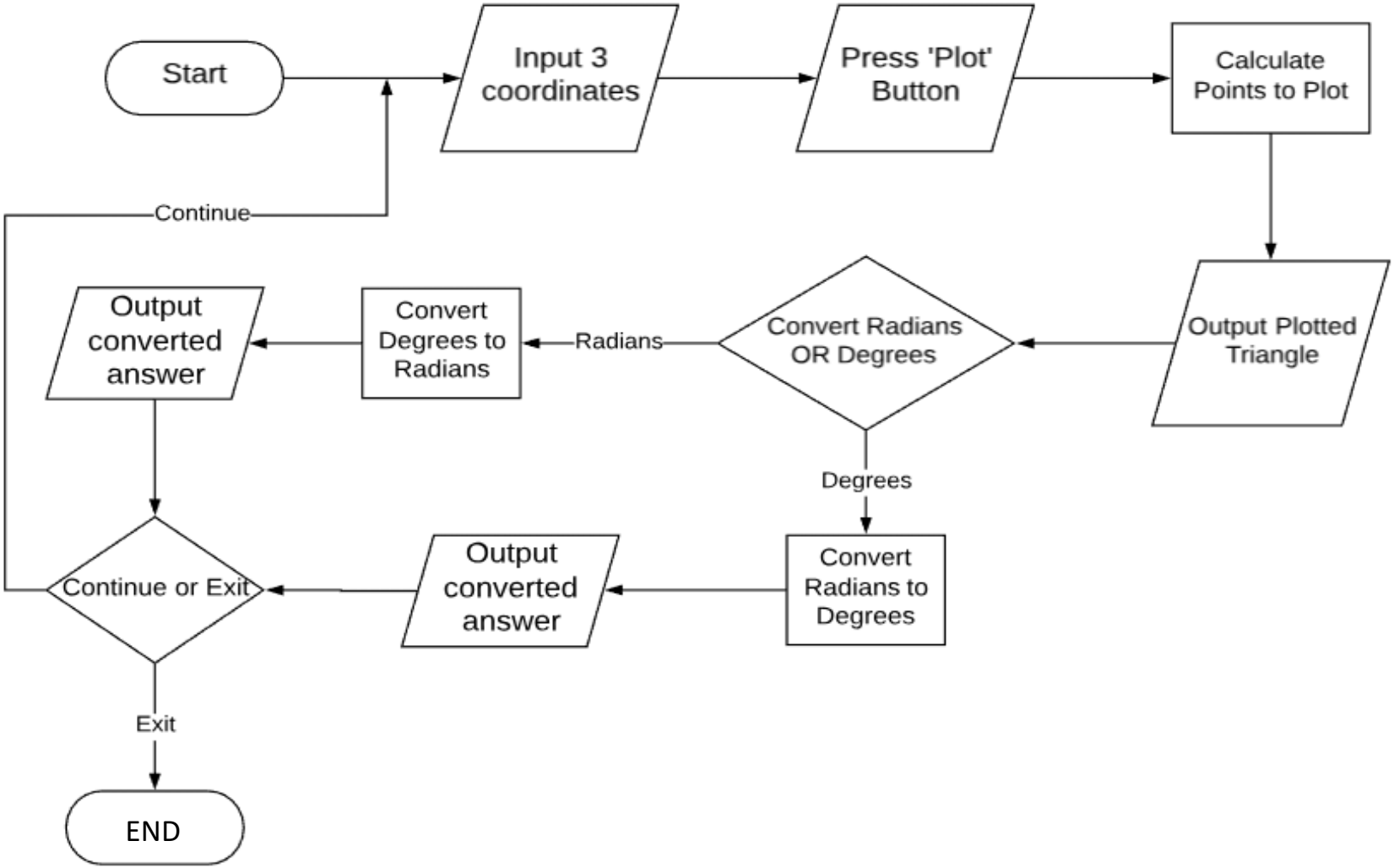
```
CLASS Stack():
  FUNCTION __init__(self):
      self.Elements ← LIST
   FUNCTION Push(self, Element):
      self.Elements APPEND (element)
  FUNCTION Pop(self):
      RETURN self.elements POP()
   FUNCTION ViewStack(self):
      RETURN self.elements
   FUNCTION isEmpty(self):
      IF self.elements == [] THEN DO
         RETURN True
      ELSE DO
         RETURN False
```

```
FUNCTION Undo():
   IF NOT StoredEquations isEmpty() THEN DO
      UndoneGraph ← StoredEquations Pop()
      UserEquation ← UndoneGraph[0]
      PLOT GRAPH (UserEquation, UndoneGraph[1],UndoneGraph[2])
   ELSE:
      OUTPUT  'There are no more equations to Undo!'

FUNCTION Factorial(n):
   IF n == 0 THEN DO
      RETURN 1
   ELSE DO
      RETURN n * Factorial(n-1)
```

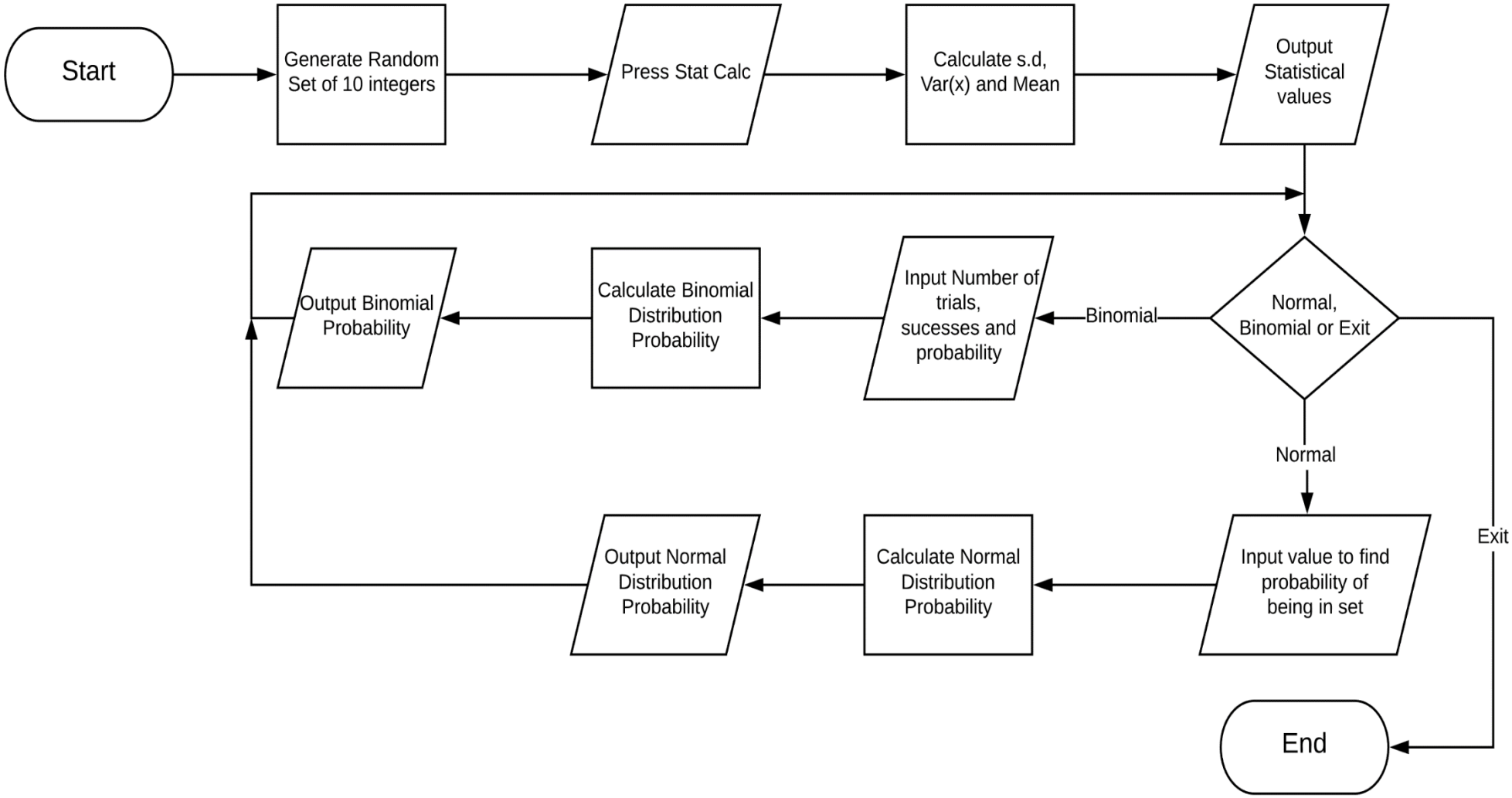
```
FUNCTION DIFFERENTIATE ():
    f(x) ← User Defined Equation
    GradientPoint ← FLOAT(User Input From GUI Field)
    dx ← 1/10000
    dy ← f(GradientPoint+dx) - f(GradientPoint)
    Gradient ← dy/dx
    OUTPUT  "Your requested gradient at the point x=(GradientPoint) is: (Gradient)"
FUNCTION INTEGRATE():
    f(x) ← User Defined Equation
    Area_Start ← FLOAT(User Input From GUI Field)
    Area_End ← FLOAT(User Input From GUI Field)
    RecWidth ← (Area_Start - Area_End)/10000
    Area ← 0
    FOR i IN RANGE(10000):
      RecHeight ← f(Area_Start + i * RecWidth)
      Area ← Area + (RecWidth * RecHeight)
    OUTPUT  "Your requested Area under the curve between x=(Area_Start) and x=(Area_End) is: (Area)"
```


Start
Open Calculus Screen
Enter Equation and domain
Create Lists of domains and ranges for equation
Plot and Display Graph on Screen
Find Area or Gradient OR exit?
Area
Input range for Area function
Calculate Area with my own defined function
Output Area value
Gradient
Calculate Area with My function
Input point for gradient function
Calculate gradient with my own defined function
Output Gradient value
EXIT
END

Trigonometry



Statistics


Start
Generate Random Set of 10 integers
Press Stat Calc
Calculate s.d, Var(x) and Mean
Output Statistical values
Normal, Binomial or Exit
Binomial
Input Number of trials, sucesses and probability
Calculate Binomial Distribution Probability
Output Binomial Probability
Normal
Input value to find probability of being in set
Calculate Normal Distribution Probability
Output Normal Distribution Probability
Exit
End

Objective 8

Objective 8 is the scheduler objective. This requires the creation of a system for users to add in the timetable for the days they attend school and stores it in the database. This also involves the ability to request meetings and view the scheduled meetings. As my client had requested, I will also plan for the ability to send email notifications when a new meeting is scheduled between the student and teacher.

Create Timetable

FUNCTION AddTimetable():

Day_Number ←INTEGER(User Input from GUI Field)

Period_One ← User Input from GUI Field

Period_Two ← User Input from GUI Field

Period_Three ← User Input from GUI Field

Period_Four ← User Input from GUI Field

Period_Five ← User Input from GUI Field

Period_Six ← User Input from GUI Field

CONNECT TO DATABASE

NewTimeID ←Increment ID(TimeTable)

SQLite Script: '''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, UserEmail))

DISCONNECT FROM DATABASE

Create Meeting

```
FUNCTION Create Meeting()
    CONNECT TO DATABASE
    TeacherTimetable ←SQLite Script: "'SELECT DayNumber, Period_1, Period_2, Period_3,
Period_4, Period_5,
    Period_6 FROM TimeTable WHERE Email =?''', (TeacherEmail))
    StudentTimetable ←SQLite Script: "'SELECT DayNumber, Period_1, Period_2, Period_3,
Period_4, Period_5,
    Period_6 FROM TimeTable WHERE Email =?''', (StudentEmail))
    Available ← LIST
    AvailableT ← LIST
    FOR i IN RANGE LENGTH(StudentTimetable) DO:
      if StudentTimetable[i] == 'FREE':
        Available APPEND(i)
   FOR i IN RANGE LENGTH(TeacherTimetable) DO:
      if TeacherTimetable[i] == 'FREE':
        AvailableT APPEND(i)
  Possible ← Available INTERSECTION AvailableT
  IF Possible == EMPTY LIST THEN DO
      OUTPUT 'No Available Times Found!'
    ELSE DO:
      ScheduledDay ← Possible[0]
      ScheduledPeriod ← Possible[1]
      ScheduleID ← IncrementID(ScheduleID)
        SQLite Script: "'INSERT INTO Scheduled(ScheduleID, ScheduleDay, SchedulePeriod, Reason,
        Email) VALUES(?,?,?,?,?)''', (ScheduleID, ScheduledDay, ScheduledPeriod, Reason, SEmail))
        OUTPUT 'Your meeting has been scheduled for:' ScheduledDay, ScheduledPeriod
        SEND EMAIL NOTIFICATION
```

Send Email

```
IMPORT SMTP LIBRARY

FUNCTION SendEmail(TeacherEmail, StudentEmail, Date, ScheduledPeriod, Reason):

    SenderAccount ←'mathematicsnea@gmail.com’

    SenderPassword = '*********'

    EmailServer ← SMTP('smtp.gmail.com', 587)

    EmailServer.ehlo()

    EmailServer.starttls()

    EmailServer.login(SenderAccount, SenderPassword)

    EmailSubject ← 'New Scheduled Meeting'

    EmailMessage ← “You have a scheduled one-to-one meeting with your maths teacher on the
        (Date)   at Period (ScheduledPeriod) The reason given for this meeting is: (Reason)”

    SENDMAIL FROM EmailServer with Message ← EmailMessage, Receivers ←  [TeacherEmail,
        StudentEmail]

    QUIT EmailServer
```

Overall Design Client Feedback

After completing the design section, I wanted to ensure all the plans that I have made to complete this project was exactly in line with my client's requests. To gain approval to continue onto the next stage of the project, I contacted my client to check if they thought everything I had done so far was what they wanted.

RE: Maths App - Design Inbox

[redacted] 12:31

to luqman.liaquat

Hi Luqman,

I've been looking through your design section of the application you are making for me and I'm definitely pleased. It's an impressive amount of work and time you have dedicated to this project!

As for the objectives, I think you've designed and linked everything to the original objectives table so it's definitely an accurate reflection of what I had intended. I also appreciate the addition of the email notifications which I asked you to do. It really is some great work and I hope you keep it up because I can't wait to see the final product!

Regards,

[redacted]

Reply Reply all Forward

Figure 8 – Overall design feedback (screenshotted from my phone Gmail app as it is much clearer than the stretched desktop app emails)

From the above email, my client has confirmed that he approves of my design section. As a result, I can now continue and work on creating the final product through my chosen solution. This solution involves using the Python programming language and KIVY kv code for the GUI and design elements of the program.