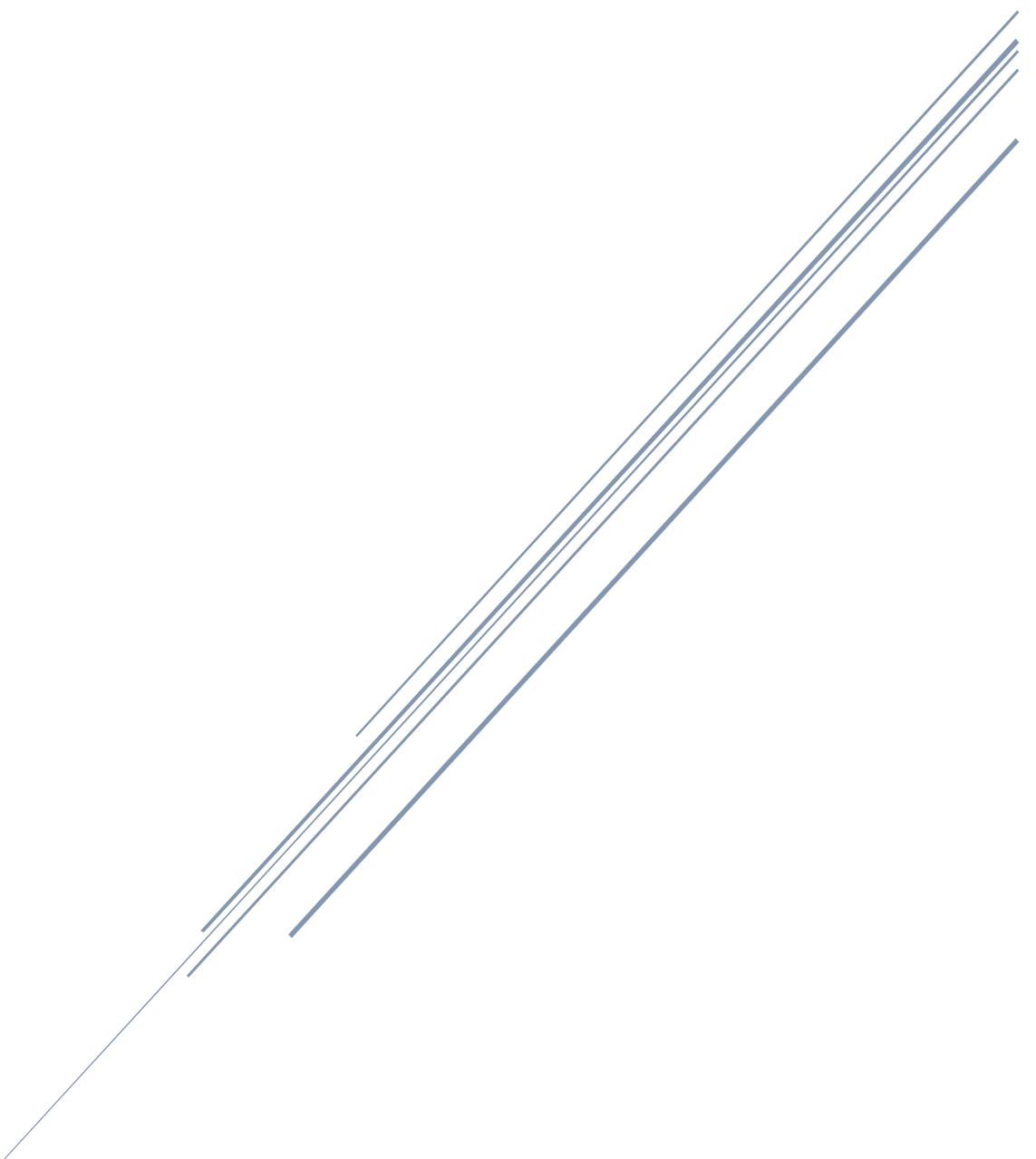


MATHS QUIZ

Joe Down



Royal Grammar School High Wycombe
H446

1 CONTENTS

2	Analysis of Problem.....	6
2.1	Problem Identification	6
2.1.1	Problem.....	6
2.1.2	Why it is solvable by computational methods.....	6
2.2	Stakeholders	7
2.2.1	Identification	7
2.2.2	Teachers	7
2.2.3	Students	8
2.3	Research.....	9
2.3.1	Mathmo	9
2.3.2	MathsBot.....	10
2.3.3	Wolfram Problem Generator	11
2.3.4	Summary	13
2.4	Proposed Solution.....	13
2.4.1	General proposal.....	13
2.4.2	Software Limitations	13
2.4.3	Hardware limitations.....	14
2.4.4	Software requirements	14
2.5	Stakeholder response to analysis.....	16
2.5.1	Troy Garvin, teacher of Mathematics.....	16
2.5.2	Tommy Clair, Year 12 student currently studying A Level Maths and Further Maths .	16
2.5.3	Analysis	16
2.6	Success Criteria	16
2.6.1	Summary	16
2.6.2	Specific details.....	16
3	3 Design of Solution	21
3.1	Decomposition of problem	21
3.2	Solution Structure	24
3.2.1	User Interface.....	24
3.2.2	Data Structure	32
3.2.3	Class structure.....	36
3.2.4	Algorithms to develop.....	37
3.2.5	Algorithms to be used	59
3.3	Usability Features.....	65

3.4	Approach to Testing	66
3.4.1	During Development	66
3.4.2	Post Development.....	66
3.5	Stakeholder response to design	70
3.5.1	Troy Garvin, teacher of Mathematics.....	70
3.5.2	Tommy Clair, Year 12 student currently studying A Level Maths and Further Maths .	70
3.5.3	Analysis of responses	70
4	Development of Solution	71
4.1	Database Setup	71
4.1.1	Create classes table using SQLite DB Browser GUI (renamed from class in design as plural is more descriptive of contents)	71
4.1.2	Create users table using SQLite DB Browser GUI (Code generated can later be used to automate creation of database in python if not found).....	72
4.1.3	Create class_user table using SQLite DB Browser GUI	72
4.1.4	Create homework table using SQLite DB Browser GUI.....	73
4.1.5	Create class_homework table using SQLite DB Browser GUI	73
4.1.6	Create question_types table using SQLite DB Browser GUI	74
4.1.7	Create questions table using SQLite DB Browser GUI	74
4.1.8	Create homework_questions table using SQLite DB Browser GUI	75
4.1.9	Create question_results table using SQLite DB Browser GUI	75
4.2	Populating Database (testing.db_populate).....	76
4.2.1	Populate users table.....	76
4.2.2	Populate classes table	77
4.2.3	Populate class_user table.....	78
4.2.4	Populate homework table	79
4.2.5	Populate class_homework table	80
4.2.6	Populate question_types table	80
4.2.7	Populate questions table.....	81
4.2.8	Populate homework_questions table	82
4.2.9	Populate question_results table	83
4.3	UI Implementation (window)	84
4.3.1	Base UI	85
4.3.2	Login screen	86
4.3.3	Create account screen.....	87
4.3.4	Student main menu screen	88
4.3.5	Teacher main menu screen	89

4.3.6	Homework select screen	90
4.3.7	Previous scores screen	90
4.3.8	Set homework screen.....	91
4.3.9	View classes screen	93
4.3.10	Admin screen	94
4.3.11	Account management screen.....	99
4.3.12	Template quiz screen	100
4.4	Login Scripts (scripts.db_scripts)	100
4.4.1	Import required Python libraries to perform hashing algorithms and communicate with an SQLite database	100
4.4.2	Connect to database	101
4.4.3	Function to identify if the value passed matches a username stored in the database (True or False should be returned)	101
4.4.4	Function to find the salt for a given username.....	103
4.4.5	Function to generate a salt	104
4.4.6	Function to hash a password.....	104
4.4.7	Function to check if a password hash matches the stored hash	106
4.4.8	Function to check if a password is correct for the given username	107
4.4.9	Function to create an account (insert valid data into the users table)	108
4.4.10	Function to get the user id which matches a given username	110
4.4.11	Function to get the account type for a given user id.....	111
4.4.12	Function to get the username for a given user id.....	112
4.4.13	Function to get the first name for a given user id	113
4.4.14	Function to get the last name for a given user id	114
4.4.15	Function to format an inputted first/last name according to database rules (all lowercase).....	115
4.4.16	Function to update first name in database for a given user id	115
4.4.17	Function to update last name in database for a given user id.....	117
4.4.18	Function to update password in database for a given user id	118
4.5	UI Scripts (scripts.ui_scripts)	120
4.5.1	Import required Python libraries to communicate with SQLite database and use scripts from login scripts	120
4.5.2	Connect to database	120
4.5.3	Function to get a list of all classes for a given user id.....	120
4.5.4	Function to get a list of all homework for a given class id	122
4.5.5	Function to get a list of all classes for a given teacher id	124
4.5.6	Function to get a list of all students for a given class id	125

4.5.7	Function to get a list of all questions for a given homework id.....	126
4.5.8	Function to get the score for a given homework for a given user	128
4.5.9	Function to get the homework name and due date for a given homework id (used in get_homework_score).....	133
4.5.10	Function to insert a student into a class.....	135
4.5.11	Function to check if a given student is in a class	139
4.5.12	Function to remove a student from a class	142
4.5.13	Function to create a new class	144
4.5.14	Function to remove a class.....	147
4.5.15	Function to get results for a homework for all students in a class	154
4.5.16	Function to get the name of a homework.....	162
4.5.17	Function to get the all homework scores for a given user in a given class	163
4.5.18	Function to get the question text for a given question	167
4.5.19	Function to get the correct answer for a given question	169
4.5.20	Function to remove a question from a homework.....	170
4.5.21	Function to insert a question into a homework	172
4.5.22	Function to get the incorrect answers for a given question	177
4.5.23	Function to get the correct status for a given question and user	178
4.5.24	Function to increment user attempts at question.....	181
4.5.25	Function to mark question as correct.....	183
4.5.26	Function to insert a new homework into database.....	186
4.5.27	Function to add a homework to a class.....	188
4.5.28	Function to remove a homework from database.....	191
4.5.29	Function to get type of a question	199
4.5.30	Function to get graph equation for a question.....	202
4.5.31	Function to set graph equation for a question	203
4.6	UI Programming (main).....	206
4.6.1	Function to initialise program UI and halt program	206
4.6.2	Function to setup database or, if already present, make sure database structure is correct	208
4.6.3	Window class	215
4.7	Series Scripts (maths_scripts.series)	338
4.7.1	Taylor Series Class	338
4.7.2	Maclaurin Series Class	344
4.7.3	Sine function	344
4.7.4	Cosine function	350

4.8	Calculus Scripts (maths_scripts.calculus)	352
4.8.1	Integral approximation base class	352
4.8.2	Trapezium rule class	355
4.8.3	Simpson's rule class.....	357
4.8.4	Comparison of Simpson's rule to Trapezium rule.....	359
5	Evaluation	363
5.1	Final Testing	363
5.1.1	Testing.....	363
5.1.2	Stakeholder response to finished program	399
5.2	Level of Success.....	402
5.2.1	Success Criteria Summary	402
5.2.2	Evaluation of success criteria	407
5.3	Maintenance and Further Development.....	407
5.3.1	Current program limitations.....	407
5.3.2	Summary of further development and maintenance	408

2 ANALYSIS OF PROBLEM

2.1 PROBLEM IDENTIFICATION

2.1.1 Problem

Particularly with the introduction of the new A-Level specifications, it can often prove very difficult for both teachers and students to find new and appropriate sample material to assist in practising for the exams. In addition to this, it can be difficult for teachers to keep track of how all students are progressing and in what areas they need to improve due to the large volume of data such as test and homework marks which need to be considered for each student, making it possible for struggling students to be unknowingly left behind.

2.1.2 Why it is solvable by computational methods

While, in some subjects, questions are very subjective and therefore make more sense to be written by a human, the logical nature of mathematics means that it should be possible to computationally generate unique questions and accurate solutions based around frameworks mimicking the style of actual A-Level questions. Storing and searching large amounts of data is also a far more trivial task for a computer than a human through the implementation of databases, further making this problem amenable to a computational approach as it allows for a vast number of test scores, both on an individual student basis and across a class, to be processed in a variety of different ways in a minimal amount of time and with little to no user input in order to quickly identify areas of weakness, outputting results as needed. A database can also be implemented through a database server, allowing user details and scores to be stored on a single centralised server which could allow client programs to be implemented and submit question responses and pull results from this server over the internet. This would allow a teacher to instantly be able to see how a student is doing in a quiz in real time, something that would not be possible using non-computational methods unless the teacher were with the student while they attempting a test, something not feasible in classes larger than only a handful of students, or in the case of homework where a teacher physically cannot be present in person.

In addition to this, the problem is further amenable to a computational approach as computer systems allow for the inputting, processing and dynamic outputting of data to a user whilst abstracting away the more complex details required to do this in order to simplify all tasks for the end user. For example, scores can be shown without having to show the user how they were calculated, nor require the user to calculate them themselves, a computer is simply reliable enough at tasks such as these that if programmed correctly these can be assumed to be correct.

Additionally, for tasks such as a quiz with multiple choice answers, a choice can simply be selected and submitted, with the process of the computer identifying what was selected and validating and storing data related to whether this was correct or incorrect abstracted behind a simple ‘Correct’ or ‘incorrect’ response, giving the student immediate useful feedback without unnecessarily complicating things like by requiring a student to check a mark scheme to confirm all their results manually and tediously themselves.

2.2 STAKEHOLDERS

2.2.1 Identification

The target audience for this program will be both teachers teaching and students studying A-Level Mathematics from 2017 (specifically the program will be based around the Edexcel 9MA0 specification, though the same skills and style of questions should apply to any other exam board). It is possible that this software may also be appropriate for other users in addition to those stated, such as teachers and students in other year groups or on different exam boards, however doing this would exceed the feasibility of this project unless further time was available or additional developers could work on the project. It may also be useful at higher levels than this such as university, or even for self-teaching topics, though again this would exceed the scope of this project.

2.2.2 Teachers

2.2.2.1 *Initial interview with Troy Garvin, teacher of Mathematics*

2.2.2.1.1 Questions

- 1) Do you feel that there are enough resources available to students with regards to the new style A Level mathematics syllabus?
- 2) Do you feel that it is useful for students to be given unique questions as homework for which answers are not available to find online? If so, do you find it possible to provide such resources using currently available services?
- 3) Are there any specific ways you believe a computer program could be used to improve upon existing infrastructure (useful features etc.)?

2.2.2.1.2 Responses

1. No, definitely not. While many resources are available from the previous specification, a lot of these are only partially applicable to the new exams with significant changes to the way questions are marked and the methods used to find solutions. In addition to this, many topics have been removed so applicable questions need to be removed from all past papers, as well as many topics added not showing up at all. With the amount of work required to adjust available resources I may as well be making them again from scratch. For the new exams themselves we have the 2018 papers and almost nothing else to work with, with very little of the rest of this coming from the exam board, leaving many of myself and my students in the dark about how the exams will really be.
2. It's very useful for students to have access to a wide range of unique, unseen. With a subject such as mathematics, there isn't a better means of improving than continuing to practise, and this can't be done if there aren't enough questions to do as, even if they were done a long time ago, doing the same questions a second time never has the same beneficial effects. Currently we're finding the best resources for providing these questions are the Pearson Active Learn online textbooks as these have detailed explanations of how to solve various types of problem under the new specification and many sample questions. Over the course of a year, however, there simply still aren't enough here and questions also need to be collated from less well-made sources and even then, many questions aren't representative of the format of questions in the final exam.
3. The main useful feature a computer program could provide is a means of getting immediate feedback to questions as this allows a student to know exactly where they are struggling without my intervention, allowing them to know exactly what it is they need to go back and revise further. To do this normally, a mark scheme would need to be provided and in my

experience students are often tempted to simply copy the mark scheme and avoid doing the work, costing them in the long term as they cannot do this in an exam and I am unable to identify where their weaknesses are if they hide them.

2.2.2.2 Summary

For teachers, difficulty comes from the fact that the primary available source of sample questions is from older specification past papers and the few sample papers produced based on the new specification. This is an issue as in the case of the older specification papers, marks are awarded differently to how they will be awarded in upcoming examinations with a newer increased focus on the method used rather than purely the answer provided, making it harder for a teacher to identify how representative a student's scores in classwork are of the marks they will receive at the end of the course. It is also very easy for students to find mark schemes for these papers as they are released by each exam board ~1 year after the exam takes place, making it difficult for a teacher to identify how genuine a student's answers are and again making it more difficult for a teacher to identify how well a student is progressing. Because of this, a program to generate exam-style sample questions would be very useful as it means a teacher has access to an unlimited supply of sample questions for which answers cannot be found online, allowing enough questions to be set without the possibility of students finding mark schemes. Using relational databases answers and questions can also be stored and re-accessed later, meaning a teacher can also refer to the problems set later without having to print out mark schemes as the questions are generated, nor will they have to create mark schemes themselves, reducing work load and the potential stress of doing everything manually.

2.2.3 Students

2.2.3.1 *Initial interview with Tommy Clair, Year 12 student currently studying A Level Maths and Further Maths*

2.2.3.1.1 Questions

- 1) Do you feel that there are enough resources available to students with regards to the new style A Level mathematics syllabus?
- 2) Is helpful to you if feedback is provided immediately as to whether a question has been answered correctly or not something which particularly matters to you?
- 3) Do you find being able to practise new, previously unseen, questions a useful means of developing your mathematical skills or do you prefer simply learning the theory behind each topic?
- 4) Are there any specific ways you believe a computer program could be used to improve upon existing infrastructure (useful features etc.)?

2.2.3.1.2 Responses

- 1) No way. Our teachers haven't given us much besides the textbook and I get the impression that's because that really is all there is. It's quite annoying because the school really pushes for us to be doing independent learning outside of lessons and we can't really do that if there isn't anything for us to learn from or practise with.
- 2) Yeah. Some of my teachers can be quite slow when it comes to returning work because they teach so many of us, so often by the time I get something back I've forgotten about it. That doesn't really help me learn because I've usually either already improved or reinforced my mistakes by repeating them. So yes, it would be helpful to be able to get immediate feedback so I can immediately get to work fixing my mistakes.

- 3) There's really no point doing the same questions again and again because I can usually at least roughly remember the answer and it doesn't introduce you to different ways of solving a problem. It's definitely very helpful reading the textbook and learning the theory behind doing different types of question, but there's nothing quite like just doing some.
- 4) I'm not sure. Well, you were talking about getting immediate responses to questions answered so if that's a feature you're planning to implement, that's something that would be very useful. I think it's probably just a case of a computer being able to set lots of questions that would help the most because a textbook can only fit so many inside it.

2.2.3.2 *Summary*

For students, revision can also be difficult, particularly since many resources related to the new maths specifications are not yet accessible unless a subject teacher. For this reason, a means of accessing a large supply of questions is invaluable. Another issue that can be encountered when revising is the fact that a student must see what the correct answer is to know if their answer is correct, preventing them from reliably retrying the question as they already know what they should get as an answer. This could be solved however by using a computer to verify if an answer is correct and allow the student to reattempt the question without being shown the real answer.

2.3 RESEARCH

2.3.1 Mathmo

2.3.1.1 *Description*

While numerous question banks exist for A-Level mathematics exist, the only A-Level specific question generator appears to be Mathmo (nrich.maths.org/mathmoApp), developed by the University of Cambridge.

Getting Started
Use the input box in the Mathmo header to add a new named exercise. Each exercise appears in a new tab. Click on the tab to add questions on any of a large choice of topics. These same questions stay with you until you choose to delete them.

Mathmo remembers
When you close down mathmo, your exercises will be remembered in local storage. When you return to mathmo, the same exercises and questions will be restored. Even if you start mathmo on a different computer, you can generate the same questions by creating an exercise with the same name.

This works because the random number generator used to generate questions is seeded from:

- The exercise name
- The question topic
- The question number within a topic

When all of these are the same mathmo generates the same question.

Sharing
You can share questions with your friends. Press the share button to see available methods.

Copyright © University of Cambridge [About](#) [Clear all](#)

This tool allows a user to create an exercise and add to it a uniquely generated question from a list of possible topics.

This program does however have several limitations. First it is web based, without a user login system, meaning all exercises are stored locally as cookies, meaning if a different browser is used or cookies are wiped all previously generated questions are lost.

The tool also predates the new style A-Level so is missing some topics and includes some topics which have either been removed from the specification or moved to GCSE.

qualifications.pearson.com/content/dam/pdf/A%20Level/Mathematics/2017/specification-and-sample-assessment/a-level-l3-mathematics-specification.pdf

2.3.1.2 Advantages

- Web based so accessible from many different devices running many OSes
- Generates unique questions and answers so impossible to cheat and find solutions online
- Large variety of topics
- Allows mixing of topics

2.3.1.3 Disadvantages

- Web based so inaccessible without an active internet connection
- No login system so data can only be saved locally as cookies (so will be lost if cookies are cleared or using a different device)
- No way of tracking results

2.3.2 MathsBot

2.3.2.1 Description

An alternative maths question generator is the MathsBot question generator (mathsbot.com/questionGenerator). This tool, while aimed at a much lower level than A-Level, is more interactive than Mathmo, illustrating how topics can be selected and question output modified using a difficulty selector to tune the tool to different users.

This tool also implements features such as a print function to allow questions to be completed more easily in a classroom if a teacher prefers work to be done by hand, as well as a timer if completing questions online to increase difficulty and ensure students are working at a fast enough pace that they are able to learn from what they are doing and improve later, as well as ensure they put all focus into completing the questions rather than multitasking and lacking concentration.

2.3.2.2 Advantages

- Difficulty selector allows questions to be tailored to different users
- Print function allows questions to be generated for use in a classroom without individual computer access
- Timer to encourage tasks be completed in a reasonable amount of time
- Questions uniquely generated
- Allows mixing of topics

2.3.2.3 Disadvantages

- Limited range of topics
- Results are not stored

2.3.3 Wolfram Problem Generator

A final maths question generator tool is the Wolfram Problem Generator (wolframalpha.com/problem-generator/). This tool is again not aimed at A-Level specifically; however, it includes the ability to select questions of almost any sort that would be required and more.

Again, this tool includes a difficulty selector which affects question difficulty by increasing the complexity of the data used (e.g. number of decimal places, number of values to be used etc.), a feature which it is increasingly clear is required to create a competing product.

This tool is based around the Wolfram Alpha APIs meaning it can compute a large range of mathematical problems including complex algebra. This is all far beyond the scope of something that can be done for this project, however it is possible that I could also use the same APIs to assist in the development of my solution. This tool is also not very focused on the task of improving exam results specifically, so it also still is not a solution for the problem described, particularly since there is no way to collate questions and solutions from a variety of different topics into quiz form, nor is there any means, in this tool or any of the others, of going back and checking the progress of question scores, nor a sufficient means of saving questions and solutions for later (there is a print function though no digital means of saving and a premium subscription is also required to do this, something many teachers may be unwilling to pay). When solutions are provided by this program it is also only

the answer without workings, a key feature for improving exam results to know where mistakes have been made.

2.3.3.1 *Advantages*

- Uniquely generates questions and marks answers
- Large range of topics of high complexity
- Difficulty level can be set

2.3.3.2 *Disadvantages*

- Results are not stored

2.3.4 Summary

2.3.4.1 *Key features to include*

- Uniquely generated questions
- Auto-mark answers
- Multiple choice
- Difficulty adjustment
- Large range of questions
- Allow mixing of topics
- Login system to save questions and scores

2.4 PROPOSED SOLUTION

2.4.1 General proposal

Due to the lack of other tools up to date with the new specification, I propose the development of a program to allow an A-Level maths topic to be selected and a unique practise question, similar in style to a real exam question, to be generated in a style similar to that of the example programs given (particularly similar to Mathmo in terms of complexity of question), though with much more of an emphasis on matching the style of a real exam question. I intend to then perform a different task depending on the user (Student or teacher, a login system should be implemented to allow for this) where a teacher is then given a worked answer and the opportunity to save the question and its answer for future reference, while a student is given an input box into which they must enter the correct answer before they are given solutions in a similar way to the question answering method of Wolfram Problem generator. I intend for the program to be able to save scores depending on how many questions a student can correctly answer and allow teachers and students to browse all relevant scores to identify strengths and weaknesses as this is a feature missing from all the identified similar existing solutions.

2.4.2 Software Limitations

There are some obvious software limitations to this proposal. The main potential issue is with the volume of questions types which would need to be coded for the program to produce questions covering the entire syllabus. This is infeasible with the time limitations of this project and therefore only as many question generators will be created as can be done within this time as a proof of concept of what could be done if the program were to be developed full time. In addition to this, the nature of a multiple-choice quiz means that a student is marked by the program as either right or wrong. In what way they were wrong or their efficiency in reaching the correct answer cannot be assessed and so the program should only be used as a source of practise and a means of identifying questions which a student frequently gets wrong, rather than actively teaching how to solve each

type of problem. Additionally, due to time and cost constraints, the proposed solution is a standalone application, with no web-based component to allow questions and scores to be retrieved from/written to a server which controls a large shared database to which an instance of the program connects as a client. This means homework must be set by teachers and homework attempted by students on the same computer unless they can share the database with each other remotely using their own methods. This also means that, as the database is stored locally, it is possible that a user may be able to tamper with it, for example to change homework scores, from outside the program to cheat which would not be possible if the database was stored remotely as direct access to data would be impossible and validation could be put in place to ensure only valid data edits are committed.

2.4.3 Hardware limitations

All calculations required for the A-Level maths exam are completable using no more than a regular scientific calculator. All UI elements and database management required for this project should also be relatively lightweight. It would also be inappropriate for hardware requirements to be beyond those of a low-end laptop as students and teachers may be on a limited budget, meaning I will ensure that superfluous features are kept to a minimum. It is also therefore most appropriate to not utilise any proprietary tools within the program which require license fees as it is unlikely that a teacher, or particularly a student, would be willing to pay more than a minimal amount of money, if anything at all.

Specifications	Features
 14.1" Full HD IPS Screen	<ul style="list-style-type: none"> • Built-in Webcam with Microphone – Ideal for Skype video calls • 1x High-Speed USB 3.0 Port and 1x USB 2.0 Port • Tf-card Slot (Supports Micro sd cards upto 256GB) • Mini HDMI Port • 15mm Thickness • UK Keyboard Layout • Full HD IPS Display • Upto 8 Hours Battery Backup

Above are the advertised specifications of the current best-selling laptop (laptops are far more common in a school environment than a desktop) on Amazon UK ([amazon.co.uk/Computer-Lapbook-x5-Z8350-Windows-Bluetooth/dp/B0764G3FLG](https://www.amazon.co.uk/Computer-Lapbook-x5-Z8350-Windows-Bluetooth/dp/B0764G3FLG)). From this, test specifications (these can be emulated using a virtual machine) should be at a maximum:

CPU clock: 1.92GHz (ark.intel.com/products/93361/Intel-Atom-x5-Z8350-Processor-2M-Cache-up-to-1.92-GHz)

Core count: 4

RAM: 4GB

Storage capacity: 64GB

Graphics: Intel HD 200MHz

OS: Windows 10

2.4.4 Software requirements

I plan to utilise Python 3.6 as a programming language through which to develop this solution with a UI developed using the PyQt5 module. This is because python is a highly adaptable programming language with an emphasis on object orientation, which I plan to use to create dynamic objects for each style of question which can calculate a range of relevant values depending on randomly generated set of relevant variables. PyQt will be used for user interface as it streamlines the process

of creating and positioning UI elements compared to other libraries such as Tkinter by including the Qt Designer which provides a visual interface whereby elements can be arranged by the user visually and the relevant code to create the window's composition is generated automatically. It also provides a variety of widgets such as input boxes and buttons to simplify the system of inputting and outputting data to a user. Python and PyQt are also easily lightweight enough to run on the target specifications defined above. Python does have one potential drawback in that it is interpreted rather than compiled. This means that as high-level code is translated to machine code as it runs rather than before like a compiled program would, it will require additional processing to run and therefore have higher system requirements than a compiled equivalent code. This is mostly counteracted however by the fact that any advanced processing done by the program is contained within only a few features which will only be run as requested, such as when calculating scores or generating a question, and not constantly. This means for most of the program's usage (navigation, simpler database checks etc.) it is undemanding enough that the performance impact of an interpreted language is negligible and so convenience of language can be prioritised. In addition to this, interpreted code is not platform specific, unlike compiled code which can only be run on a particular type of system, and so the same program can be written and potentially run on various different operating systems with only minimal tweaks to the code being required (i.e. only system calls need to be changed for different platforms), something far more feasible within the scope of this project and it's time constraints in the event it is decided to create versions of the program for OSes other than Windows.

For handling the database, SQLite will be used. Ideally a database server such as PostgreSQL or MySQL would be used for this project, allowing many clients to remotely connect to the same database and therefore allow teachers and students to set and take homework from any location on any system. Additionally, as a client can only request to make changes or read from the database server and cannot directly access the database itself, the integrity of the database is increased as all validation can be controlled by the server, which can be trusted unless the server itself is compromised as all validation is guaranteed to be implemented as intended by the developer, whereas a local server could potentially be edited by a user if they wish to edit data stored as the database itself is directly accessible. Maintaining a server does however incur costs and opening a server to outside connections introduces potential security issue and so the maintenance which would be required to implement this exceeds what is possible for a project of this scope. SQLite has therefore been selected as it is one of the few SQL dialects which operates serverless and is the most popular dialect of this type for Python, with the most robustly implemented Python libraries and therefore the best documentation and support available in the event of error fixing being required.

Windows will be the target operating as it currently accounts for ~80% (en.wikipedia.org/wiki/Usage_share_of_operating_systems#Desktop_and_laptop_computers) of the desktop/laptop operating system market and is by far the most accessible to teachers. This does not affect the development of the program in any way as all tools being used are designed for Windows, though changes may have to be made for compatibility if a decision were to be made to get the program running on other operating systems as it is not designed for use with other platforms as, while all major x86 based operating systems support Python and PyQt, testing will need to be done for each platform and code adapted accordingly. For example, most UNIX based OSes use a different standard file system to Windows, and it would be expected of users for the program to conform to these standards.

2.5 STAKEHOLDER RESPONSE TO ANALYSIS

Stakeholders were allowed to view the analysis documentation and asked to give any relevant feedback

2.5.1 Troy Garvin, teacher of Mathematics

You've hit the nail on the head with this. The exam board is clearly not able to supply resources quickly enough and in large enough quantities to satisfy our needs as teachers and so an unlimited supply of questions is hard to argue with. I like the idea of being able to create and monitor classes as it means I can keep track of where people are struggling on an individual and class basis as it would help with lesson planning, deciding which topics to focus on and which to spend less time on. Having to do less marking is also always a positive.

2.5.2 Tommy Clair, Year 12 student currently studying A Level Maths and Further Maths

This sounds very useful as a revision tool. I like that teachers aren't needed to mark work so I can immediately work out where I'm struggling both while I'm doing a homework and in the grand scheme of things across all topics. Being able to be set such large quantities of work might be a bit daunting but I'm sure my teachers will be able to moderate themselves and set a reasonable amount. I also like the slightly competitive aspect of all our scores being shown to the teacher as a class because it will motivate me to ensure I'm getting the highest scores in the class!

2.5.3 Analysis

The analysis appears to satisfy the needs of the stakeholders on a basic conceptual level, however further feedback will need to be requested after a more detailed design has been completed

2.6 SUCCESS CRITERIA

2.6.1 Summary

Ultimately, the success of this program will be measured by how it is received by teachers and students and whether they consider it better than other solutions currently on offer or simply using the limited material supplied by exam boards. To be successful the project must also always produce correct answers to questions and when quizzed never reject an answer that is valid, in whatever form it is entered and should be identified as something enjoyable to use by most students that try it. The program should be navigable by users of any computer literacy level and be intuitive to use. For teachers it must be capable of giving insight into how well both individual students and classes are progressing across different topics and allow them insight into how they can adapt and improve their lessons to accommodate for this, as well as give insight into how hard students are working through the precise knowledge of when students work and for how long.

2.6.2 Specific details

2.6.2.1 General

Criteria	Evidence of success
User can create account	User details (username, password, name etc.) are stored in database: <ul style="list-style-type: none">• Correctly• Permanently
User can login with account details	User login details can be: <ul style="list-style-type: none">• Correctly compared to those in database

	<ul style="list-style-type: none"> • Matching user access given if login details are matching
User can logout	<p>Logout button:</p> <ul style="list-style-type: none"> • Can be pressed • Returns to login screen • Revokes access to the rest of the program
User can update account details	<p>Non-identifying user details (i.e. All details except for username and id):</p> <ul style="list-style-type: none"> • Can be changed and amended • With new values permanently stored • Without introducing data confliction
User details are stored securely	<p>Password is:</p> <ul style="list-style-type: none"> • stored • salted hashed • form
Enough usability features are in place	<p>Features are:</p> <ul style="list-style-type: none"> • Sufficiently labelled • Features such as buttons or dropdown menus etc. are used as appropriate • Destructive features such as deleting information are made clear to not take lightly (e.g. buttons made red to alert user)
Navigation features	Ability to navigate to key pages such as the main menu
All features contained within a graphical user interface	<p>All features of the program can:</p> <ul style="list-style-type: none"> • Be accessed through a user interface • With user typing kept to a minimum • At no point requiring interaction with a terminal
Database integration to permanently store data	<ul style="list-style-type: none"> • All data is retained when the program is restarted • Structured in such a way that data can be accessed, and new data added as needed
Features should be all accessible as required from a main menu	<p>Main menu through which:</p> <ul style="list-style-type: none"> • All pages are reachable by navigation • Through a minimal number of other pages
Permission system	<p>Access to data should be:</p> <ul style="list-style-type: none"> • Restricted to only those it relates to (i.e. teachers can only see their details and student details in their classes and students should be unable to access teacher or other student data). <p>Program features should be limited to:</p> <ul style="list-style-type: none"> • Only the type of user they apply to (only teachers can make and manage classes and only students can be set homework)

2.6.2.2 Teacher

Criteria	Evidence of success
Teacher can create class	<p>Class details (Name, teacher, students, homework etc.) are:</p> <ul style="list-style-type: none"> • Correctly • And permanently

	<ul style="list-style-type: none"> • Stored in a database
Teacher can add student to class	<p>Class student relationship is:</p> <ul style="list-style-type: none"> • Correctly • And permanently • Stored in database
Teacher can remove student from class	<p>Class student relationship is:</p> <ul style="list-style-type: none"> • Correctly • And permanently • Removed from database
Teacher can add homework to class	<p>Class homework relationship is:</p> <ul style="list-style-type: none"> • Correctly • And permanently stored in database • With relevant homework details (name, questions etc.)
Teacher can remove homework from class	<p>Class homework relationship is:</p> <ul style="list-style-type: none"> • Correctly • And permanently • Removed from database
Teacher can delete class	<p>Class details are:</p> <ul style="list-style-type: none"> • Correctly • And permanently • Removed from database
Teacher can add custom question to homework	<p>Question homework relationship is:</p> <ul style="list-style-type: none"> • Correctly • And permanently stored in database • With relevant question details (name, question text, correct answer, incorrect answers etc.)
Teacher can add automatically generated question to homework	<p>Question homework relationship is:</p> <ul style="list-style-type: none"> • Correctly • And permanently stored in database • With relevant question details (name, question text, correct answer, incorrect answers etc.)
Teacher can select question difficulty and question adjusts as needed	<p>Question difficult is:</p> <ul style="list-style-type: none"> • Correctly stored • With question in database • Question generation criteria are satisfied (see below)
Teacher can remove question from homework	<p>Question homework relationship is:</p> <ul style="list-style-type: none"> • Correctly • And permanently removed from database with relevant question details
Teacher can view results for each student for a given homework	<p>For each student, question results for a given homework and student are:</p> <ul style="list-style-type: none"> • Retrieved from the database • A score calculated from the number of correct responses
Teacher can view results for each homework for a given student	<p>Question results for each homework in the class and a given student are:</p> <ul style="list-style-type: none"> • Retrieved from the database

	<ul style="list-style-type: none"> A score is calculated from the number of correct responses
Scores correctly calculated	<p>Score calculated as:</p> <ul style="list-style-type: none"> The sum of each correct response Divided by the number of times that question was attempted All divided by the number of questions Multiplied by 100 to give a percentage
Database usability	<p>Database should be structured in such a way that it is:</p> <ul style="list-style-type: none"> Fast to insert and find relevant data Without the need for superfluous extra processing <p>Database relations should allow for:</p> <ul style="list-style-type: none"> The introduction of new tables Introduction of new types of relationship between tables To allow for potential future maintainability and updates without extensive restructuring
Validation	<p>All inputs should be validated to be</p> <ul style="list-style-type: none"> Of a valid form Usable for their intended purposes

2.6.2.3 Student

Criteria	Evidence of success
Student can access homework set for a given class	<p>Program can:</p> <ul style="list-style-type: none"> Output a list of homework from the database for the given class Launches the quiz if a homework is selected
Student can only do homework before due date	<p>Program excludes student from accessing quiz past stored due date</p>
Quiz gives question and allows choice and submission of multiple-choice response	<p>Question correct status:</p> <ul style="list-style-type: none"> Is correctly changed to True in database for given user and question Only if the correct response was selected and submitted
Student can see results for each homework in a class by class basis	<p>Question results for each homework in the class and the current student are:</p> <ul style="list-style-type: none"> Retrieved from the database A score is calculated from the number of correct responses
Score correctly calculated	<p>Score calculated as:</p> <ul style="list-style-type: none"> The sum of each correct response Divided by the number of times that question was attempted All divided by the number of questions Multiplied by 100 to give a percentage

2.6.2.4 Questions

Criteria	Evidence of success
Enough topics covered	Topics that can be generated: <ul style="list-style-type: none"> • Cover multiple different topics
Generated correct answers are correct	When manually calculating answers to generated questions: <ul style="list-style-type: none"> • Answer found matches the one generated
Generated incorrect answers are unique and incorrect	When manually calculating answers to generated questions: <ul style="list-style-type: none"> • Answer found does not match any of the ones generated • Each incorrect answer is different
Questions are relevant to the mathematics specification	Topics covered are: <ul style="list-style-type: none"> • Within the Edexcel mathematics specification

3 DESIGN OF SOLUTION

3.1 DECOMPOSITION OF PROBLEM

- 1) Login
 - a) Username and password input
 - b) Username and password verification and program access/denial of access by comparing to data in database
 - c) Go to relevant main menu for user type
 - d) Option to create an account
- 2) Logout
 - a) Return to login screen
 - b) Unselect current user (remove all references to logged in user)
- 3) User access levels (student/teacher)
- 4) Data privacy between students (i.e. one student cannot see another's scores)
- 5) Navigation
 - a) Logout button
 - b) Home button
- 6) Account creation
 - a) Request input of
 - i) Username
 - ii) Password
 - iii) first name
 - iv) last name
 - v) account type (teacher or student)
 - b) Require input of password a second time to verify user has entered their new password correctly
 - c) Password security
 - i) Generate unique salt for each new user (to prevent the usage of lookup tables to find passwords from unsalted password hashes)
 - ii) Hash password with salt before storing
 - d) Store username, first name, last name, account type, password salt and password hash to database
 - e) Button to return to login screen
- 7) Main menu options dependent on user access levels
 - a) Both pupil and teacher
 - i) Account management
 - (1) Set new first name
 - (2) Set new last name
 - (3) Set new password
 - (4) Require new password to be entered twice to verify entered correctly
 - (5) Require old password to make any changes to user data
 - b) Pupil
 - i) View previous results
 - (1) View data about previous quizzes
 - (a) Homework name
 - (b) Homework score

- (c) Homework date
- (d) Keep track of number of attempts and factor this in to calculation of final score
(i.e. reduce score given for a question as number of attempts increases)
- ii) Homework
 - (1) Select class to do homework from
 - (2) Select homework task from list of homework
 - (3) Show due date of each homework
 - (4) Go to homework task when selected (see quiz requirements)
 - (5) Do not show homework with due date before current date in list of homework to do
- c) Teacher
 - i) View scores
 - (1) Select class
 - (2) Select homework
 - (3) Show homework results list
 - (4) For each student in class show:
 - (a) First name
 - (b) Last name
 - (c) Raw score
 - (d) Percentage score
 - (e) Number of attempts at homework
 - ii) Manage class
 - (1) Create class
 - (a) Enter class name
 - (b) Insert into database
 - (2) Manage class
 - (a) Add user
 - (i) Enter username
 - (ii) Verify user not already in class
 - (iii) Verify user exists
 - (iv) Create class student relationship in database
 - (b) Remove user
 - (i) Select student from list of students in class
 - (ii) Remove class student relationship from database
 - (c) Add homework
 - (i) Enter homework name
 - (ii) Enter homework description
 - (iii) Enter homework due date
 - 1. Verify due date is in future
 - (iv) Insert homework into database
 - (v) Establish class to homework relationship in database
 - (d) Remove homework
 - (i) Select from list of homework in class
 - (ii) Remove class homework relationship from database
 - (iii) Remove class data from database
 - (3) Delete class
 - (a) Select from list of classes
 - (b) Remove from database
 - iii) Manage homework

- (1) Select class
 - (2) Select homework to add questions to
 - (3) Add automatic question
 - (a) Select topic of generated question from list of available topics
 - (b) Select question difficulty
 - (c) Generate question for matching type
 - (d) Create question to homework relationship in database
 - (4) Add custom question
 - (a) Enter question name
 - (b) Enter question text
 - (c) Enter correct answer
 - (d) Enter incorrect answers
 - (e) Create question to homework relationship in database
 - (5) Remove question
 - (a) Select question to remove from list of questions in homework
 - (b) Show question text and correct answer to ensure user knows which homework is being removed
 - (c) Remove question to homework relationship in database
- 8) Quiz
- a) Show first question
 - b) Provide navigation buttons
 - i) Next question
 - ii) Previous question
 - c) Show matching question text
 - d) List possible answer
 - i) Randomise order of answers to obscure correct answer
 - ii) Allow only 1 answer to be selected at a time
 - e) Allow response to be submitted
 - i) Increment number of attempts
 - ii) Check if correct answer is selected
 - (1) If selected, update database to show question answered by user correctly
 - (2) If not selected output that answer is incorrect
- 9) GUI
- a) GUI design
 - i) Design and create login screen, accounting for all features described above (submit buttons, inputs etc.)
 - (1) Must link to:
 - (a) Account creation screen
 - (b) Main menu pages
 - (i) Goes to matching main menu depending on user type
 - ii) Design and create account creation screen, accounting for all features described above (submit buttons, inputs etc.)
 - (1) Must link back to login screen
 - iii) Design and create student main menu screen, accounting for all features described above
 - (1) Must allow:
 - (a) Logout
 - (b) Returning to main menu

- (c) Link to homework screen
 - (d) Link to view results screen
 - iv) Design and create teacher main menu screen, accounting for all features described above
 - (1) Must allow
 - (a) Logout
 - (b) Returning to main menu
 - (c) Link to view scores screen
 - (d) Link to manage class screen
 - (e) Link to manage homework screen
 - v) Design and create view results screen, accounting for all features described above
 - vi) Design and create view homework menu, accounting for all features described above
 - vii) Design and create view scores screen, accounting for all features described above
 - viii) Design and create quiz screen, accounting for all features described above
 - ix) Design and create manage class screen, accounting for all features described above
 - x) Design and create manage homework screen, accounting for all features described above
 - b) GUI scripts
 - i) Create script to initialise all buttons
 - ii) Create scripts to navigate between pages
 - iii) Create scripts to modify page elements
 - iv) Create scripts to reset page elements
- 10) Question generation
- a) Generate questions and answers based on a range of mathematical topics
 - b) Adjust question generated based on difficulty entered
 - c) Generate incorrect answers based off the generated correct answer
 - i) Must all be unique
 - ii) Must not match correct answer
 - d) Round all values to a reasonable number of significant figures to be usable
 - e) Save question to database
 - f) Return information about database entry to allow creation of homework question relationship
 - i) Question id

3.2 SOLUTION STRUCTURE

3.2.1 User Interface

3.2.1.1 Login Screen

Must take a username and password and have a button to check against the database and have the program respond accordingly

Username <input type="text" value="Username"/>	Password <input type="password" value="Password"/>
<input type="button" value="Submit"/>	

Create Account

- Input box for username
- Input box for password
- Submit button
- Go to create account page button

3.2.1.2 Create Account

Must take all inputs required for, or used to generate, a record in the user table and have a button to validate this data and have the program respond accordingly

First Name <input type="text" value="First Name"/>	Last Name <input type="text" value="Last Name"/>
Username <input type="text" value="Username"/>	
Password <input type="password" value="Password"/>	
Verify Password <input type="password" value="Verify Password"/>	
• Student <input type="radio"/> Teacher	
<input type="button" value="Submit"/>	

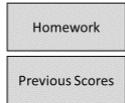
Login

- Input box for first name
- Input box for surname
- Input box for password
- Input box for password verification
- Radio buttons to select account type (student or teacher)
- Submit button
- Go to login page button

3.2.1.3 Student main menu

Must have buttons linking to all student program feature pages

Logout



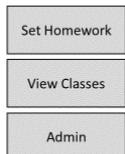
Account Management

- Logout button
- Go to homework page button
- Go to view previous scores page button
- Go to account management page button

3.2.1.4 Teacher Main Menu

Must have buttons linking to all teacher program feature pages

Logout



Account Management

- Logout button
- Go to set homework page button
- Go to view classes page button
- Go to admin page button
- Go to account management page button

3.2.1.5 Homework select page

Must allow a class to be selected by a student and show and link to all homework due

[Logout](#)

Homework:

[Class ▼](#)

Homework	Due Date
Homework Name	dd/mm/yyyy

[Return to menu](#)

- Logout button
- Select class dropdown
- List of homework, each linking to relevant quiz
- Return to menu button

3.2.1.6 Previous scores page

Must allow a user to view all homework results for each class

[Logout](#)

Previous scores:

[Class A ▼](#)

Homework A	XXX%	DD/MM/YYYY
Homework B	XXX%	DD/MM/YYYY
Homework C	XXX%	DD/MM/YYYY
Class B ▶		
Class C ▶		
Class D ▶		

[Return to menu](#)

- Logout button
- Dropdown menus for each class student is in
 - List of homework for each class with score and due date
- Return to menu button

3.2.1.7 Set homework page

Must allow teacher to select a class and homework and add or remove questions of various types

[Logout](#)

Set Homework:

Select Class:	Dropdown ▼		
Select Homework:	Dropdown ▼		
Add automatic question:	Topic ▼	Difficulty ▼	Add
Add custom question:	Name Input	Question Input	Add
	Correct Input	Incorrect Input	
	Incorrect Input	Incorrect Input	
Remove question:	Question ▼	REMOVE	
	Question Text output	Question answer output	

[Return to menu](#)

- Logout button
- Dropdown box to select class
- Dropdown box to select homework
- Automatic question
 - Dropdown to select topic
 - Dropdown to select difficulty 1-5
 - Add question button
- Custom question
 - Question name input
 - Question text input
 - Correct answer input
 - Incorrect answer 1 input
 - Incorrect answer 2 input
 - Incorrect answer 3 input
 - Add question button
- Remove question
 - Select question dropdown
 - Selected question text output
 - Selected question answer output
 - Remove question button
- Return to menu button

3.2.1.8 View classes page

Must allow teacher to see student results for each homework in class

Logout

View classes:

Class ▼		
Homework A ▼		
Student A	XXX%	Raw mark
Student B	XXX%	Raw mark
Student C	XXX%	Raw mark
Homework B ▶		
Homework C ▶		
Homework D ▶		

[Return to menu](#)

- Logout button
- Class select dropdown menu
- Dropdown menus for each homework in class
 - Under each dropdown show table with student name, percentage and raw mark
- Return to menu button

3.2.1.9 Admin page

Must allow a teacher to manage all aspects of the class (add/remove user, class and homework)

Logout

Admin:

Add user to class:	Class: <input type="button" value="Dropdown ▼"/>	Username: <input type="text"/>	<input type="button" value="Submit"/>
Remove user from class:	Class: <input type="button" value="Dropdown ▼"/>	Username: <input type="button" value="Dropdown ▼"/>	<input type="button" value="REMOVE"/>
Create class:	<input type="text"/>		<input type="button" value="Submit"/>
Delete class:	Class: <input type="button" value="Dropdown ▼"/>		<input type="button" value="DELETE"/>
Create homework:	<input type="text"/>	Due Date: <input type="button" value="Calendar ▼"/>	<input type="button" value="Submit"/>
Delete homework:	Class: <input type="button" value="Dropdown ▼"/>		<input type="button" value="DELETE"/>

[Return to menu](#)

- Logout button
- Add user to class
 - Class select dropdown
 - Input box for username
 - Submit button
- Remove user from class
 - Class select dropdown
 - User select dropdown
 - Submit button
- Create class
 - Input box for class name
 - Submit button
- Delete class
 - Class select dropdown

- Submit button
- Create homework
 - Input box for homework name
 - Due date select calendar
 - Submit button
- Delete homework
 - Homework select dropdown
 - Submit button
- Return to menu button

3.2.1.10 Account management page

Must allow a user to change non-vital account details

Logout

New First Name	New Last Name
First Name	Last Name
Old Password	
Username	
New Password	
Password	
Verify New Password	
Verify Password	
Submit	

Return to menu

- Logout button
- Input box for new first name
- Input box for new surname
- Input box for password (to verify user making changes is the correct user)
- Input box for new password
- Input box for password verification
- Submit button
- Return to menu button

3.2.1.11 Template quiz page

Must allow a user to navigate between questions, show them a question, allow them to select an answer and submit a response, receiving relevant feedback

[Logout](#)

Question x/y:

[Previous Question](#)

[Next Question](#)

Question text.....?:

A B
 C D

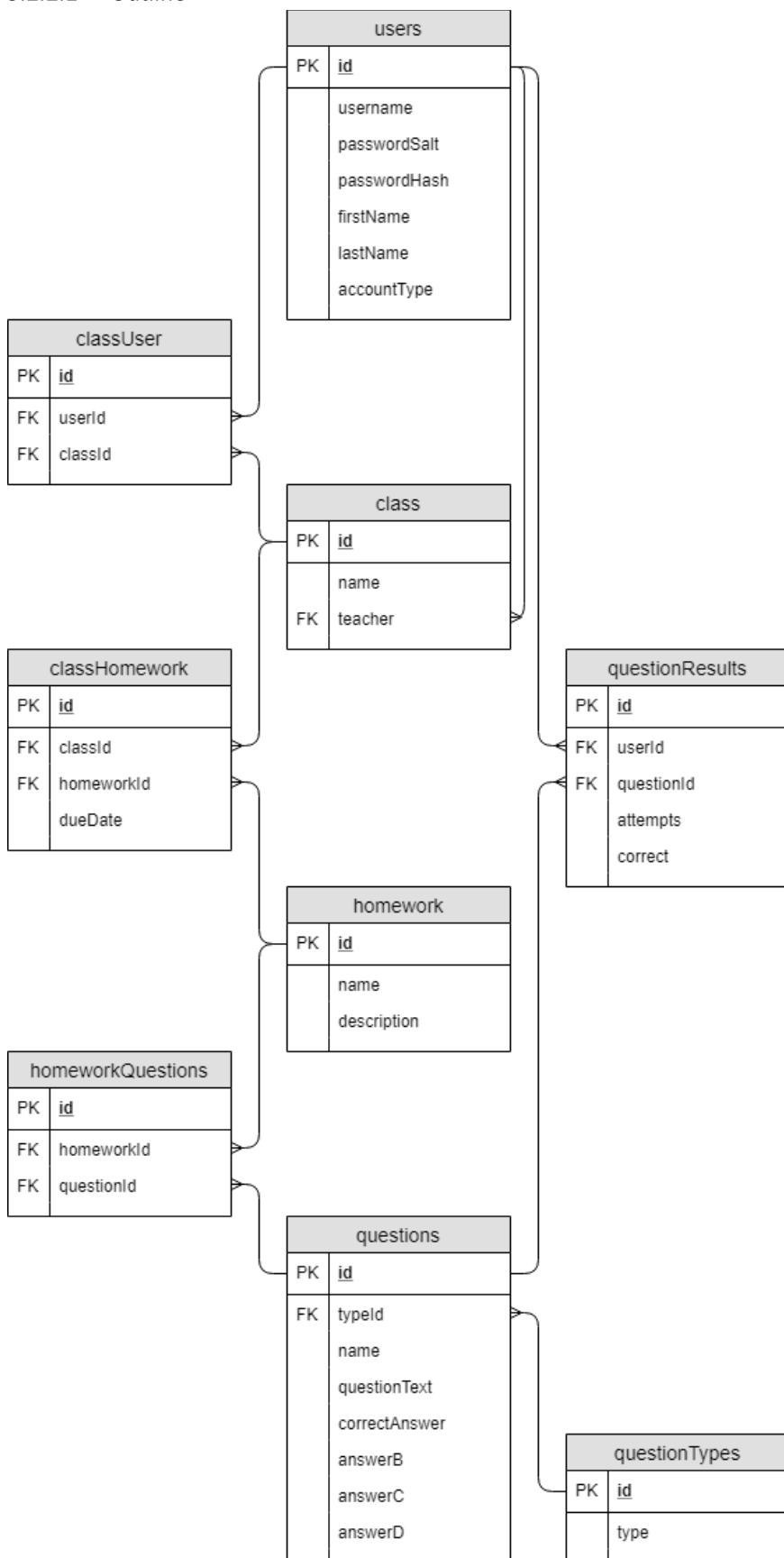
[Submit](#)

[Return to menu](#)

- Logout button
- Previous question button
- Next question button
- Question text area
- Radio buttons to select response (answer a, answer b, answer c, answer d)
- Submit answer button
- Return to menu button

3.2.2 Data Structure

3.2.2.1 Outline



3.2.2.2 Validation

3.2.2.2.1 users

Attribute	Key Type	Data Type	Further Properties	Explanation	Test data (student)	Test data (teacher)
Id	Primary	Integer	Not null Unique	Identifier to uniquely identify a single entity in the table	1 – 100	101 – 200
Username		Text	Not null Unique	Unique identifier which is more memorable for a user than the id	student1 – student100	teacher1 – teacher100
passwordSalt		Text	Not null Unique	Unique string to be appended to the end of any password entered for that user used to increase the uniqueness of the hash (prevent lookup tables being used if database is compromised)		
passwordHash		Text	Not null	Password stored in an irreversibly encrypted form to keep the password unknown if the database is compromised		
firstName		Text	Not null	User's first name	first1 – first100	first1 – first100
lastName		Text	Not null	User's last name	last1 – last100	last1 – last100
accountType		Text	Not null	Stores the user's type (student or teacher)	s	t

Unhashed passwords: password

3.2.2.2.2 class

Attribute	Key Type	Data Type	Further Properties	Explanation
id	Primary	Integer	Not null Unique	Identifier to uniquely identify a single entity in the table
name		Text	Not null	Class's name
teacher	Foreign (references user(id))	Integer	Not null	Links to the user data of the class's teacher (class admin)

3.2.2.2.3 classUser

Attribute	Key Type	Data Type	Further Properties	Explanation
id	Primary	Integer	Not null Unique	Identifier to uniquely identify a single entity in the table
classId	Foreign (references class(id))	Integer	Not null	Links to the class the user is a member of
userId	Foreign (references user(id))	Integer	Not null	Links to the user that is a member of the class

3.2.2.2.4 homework

Attribute	Key Type	Data Type	Further Properties	Explanation	Test Data
Id	Primary	Integer	Not null Unique	Identifier to uniquely identify a single entity in the table	1
Name		Text	Not null	Name of homework task	name1
Description		Text		Description of homework task	description1

3.2.2.2.5 classHomework

Attribute	Key Type	Data Type	Further Properties	Explanation
Id	Primary	Integer	Not null Unique	Identifier to uniquely identify a single entity in the table
classId	Foreign (references class(id))	Integer	Not null	Links to the class to which the homework relates
homeworkId	Foreign (references homework(id))	Integer	Not null	Links to the homework task
dueDate		String	Not null	Expiry date for homework, beyond which it will be inaccessible

3.2.2.2.6 questionTypes

Attribute	Key Type	Data Type	Further Properties	Explanation	Test Data
id	Primary	Integer	Not null Unique	Identifier to uniquely identify a single entity in the table	1

Type		Text	Not null Unique	Human understandable question type identifier	type1
------	--	------	--------------------	-----------------------------------------------	-------

3.2.2.2.7 questions

Attribute	Key Type	Data Type	Further Properties	Explanation
Id	Primary	Integer	Not null Unique	Identifier to uniquely identify a single entity in the table
Name		Text	Not null	Name of question
typeld	Foreign (references questionTypes(id))	Integer	Not null	Links to question type of question
questionText		Text	Not null	Text to be displayed as a question
correctAnswer		Text	Not null	Correct answer to question
answer		Text	Not null	A unique incorrect answer to the question
answer		Text	Not null	A unique incorrect answer to the question
answerD		Text	Not null	A unique incorrect answer to the question

3.2.2.2.8 homeworkQuestions

Attribute	Key Type	Data Type	Further Properties	Explanation
Id	Primary	Integer	Not null Unique	Identifier to uniquely identify a single entity in the table
homeworkId	Foreign (references homework(id))	Integer	Not null	Links to the homework to which the homework entry relates
questionId	Foreign (references questions(id))	Integer	Not null	Links to the question to which the homework entry relates

3.2.2.2.9 questionResults

Attribute	Key Type	Data Type	Further Properties	Explanation
id	Primary	Integer	Not null Unique	Identifier to uniquely identify a single entity in the table
userId	Foreign (references users(id))	Integer	Not null	Links to the user to which the question score relates
questionId	Foreign (references questions(id))	Integer	Not null	Links to the question to which the question score relates
attempts		Integer	Not null	Total number of times the question has been attempted by a user

correct		Boolean	Not null	Stores True or False depending on whether a correct answer has been submitted
---------	--	---------	----------	-------------------------------------------------------------------------------

3.2.2.3 Normalisation

All tables have been verified to be in Third Normal Form. This means that each non-prime attribute in a table is dependent only on the primary key, wholly by the key and all fields contain atomic values (i.e. store only a single value, no lists). In most cases this was most easily achieved by including an id attribute in each table with the ‘not null’ and ‘unique’ properties, meaning each record in the table will always have at least one attribute with which it can be uniquely identified and can therefore act as a primary key. Using an id column also simplifies development as only integer values need to be considered when referencing items in the database and so functions can be shared between various functions relating to using various different files compared to if more complex compound string based primary keys were used. Keeping fields atomic has been achieved by including tables to link instances in one table to instances in another, with each entry in the table simply including two foreign keys referencing one item in one table and one in another. This means additional data can be linked together by inserting a new entry into the connecting table rather than requiring lists of data to be stored as in a single table solution. Examples of this are the classUser, classHomework and homeworkQuestions tables which allow respectively a user to be linked to many classes, a homework to be linked to many classes and a question to be linked to many homework without listing each user id in the class table, each homework id in the class table and each question id in the homework table.

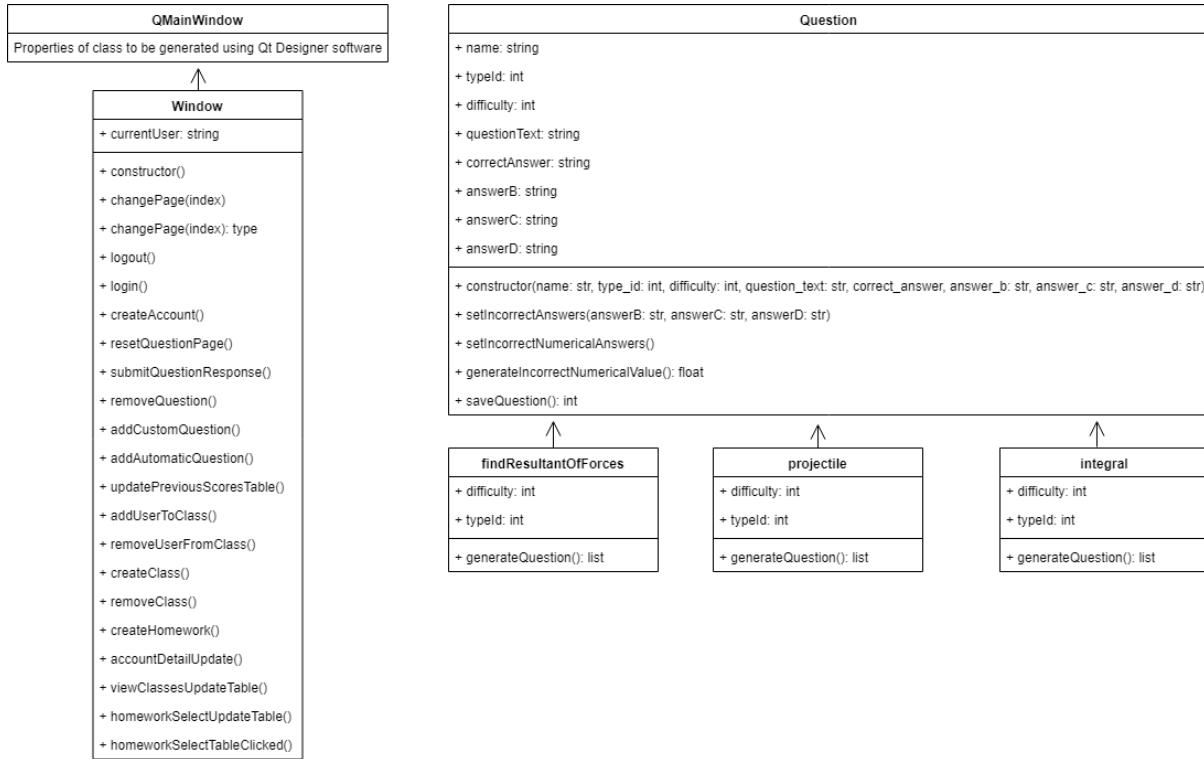
3.2.3 Class structure

3.2.3.1 Summary

The main program is to be contained within a window class based off PyQt’s built in QMainWindow class. This means a large proportion of the workings of the UI, though not specifically how it is laid out and what each element in the UI does, is already prewritten (i.e. do not need to define how a button works, simply what it does) through the use of inheritance from this parent class. Methods can then be developed to define what each interaction with the UI itself is to do (see UML diagrams below). By containing these within a window class, it also means certain variables can be easily stored as class attributes and used by methods without needing to be manually passed through each proceeding function such as the current page id or currently logged in username and user id (for specifics see UML).

Additionally, questions are to be generated using a base question class which stores all the data required to be stored in the database as class attributes and methods to intelligently generate data as needed depending on how much data was initially supplied to the class (e.g. generate incorrect answers if only a single correct numerical answer was provided). This simplifies the production of functions to generate questions of various more specific types as all that needs to be done is to generate question text and a single correct answer, with the rest able to be handled by then passing all this data to the question class.

3.2.3.2 UML



3.2.4 Algorithms to develop

3.2.4.1 Database interaction

3.2.4.1.1 Check if a user exists [checkUserExists(name)]

3.2.4.1.1.1 Pseudocode

function checkUserExists(name):

```

if (count from users where username = name) > 0:
    return True
else:
    return False
end if
  
```

end function

3.2.4.1.1.2 Explanation

Takes in a username and queries the database to find the number of occurrences of it in the database. If the query returns 0, False is returned as the username does not exist in the database whereas if the query returns 1 the username is in the database and True is returned.

3.2.4.1.1.3 Test Data

Test Number	Test	Parameter values	Data types	Expected Outcome
1	Valid data	test	String (irrelevant)	Returns True
2	Invalid data	qwerty	String (irrelevant)	Return False

3.2.4.1.2 Find a salt for a given username [findSalt(userId)]

3.2.4.1.2.1 Pseudocode

```
function findSalt(userId):
```

```
    return (passwordSalt from users where id = userId)
```

```
end function
```

3.2.4.1.2.2 Explanation

Takes in a user id (assumes this has already been validated) and queries the database to find the password salt for the matching user then returns this value.

3.2.4.1.2.3 Test data

Test Number	Test	Parameter values	Data types	Expected Outcome
1	Valid data	Test	String (irrelevant)	Returns 6a29c0b26e104282a42aab2f9c55d2b8
2	Invalid data	N/A (validation done before using function)	N/A	N/A

3.2.4.1.3 Generate salt [generateSalt]

3.2.4.1.3.1 Pseudocode

```
function generateSalt:
```

```
    return randomString(length = 32)
```

```
end function
```

3.2.4.1.3.2 Explanation

Returns a randomly generated 32-character string to be used as a salt. Python contains inbuilt libraries to do this

3.2.4.1.3.3 Test data

Test Number	Test	Parameter values	Data types	Expected Outcome
1	Returns a unique valid salt	N/A	N/A	Returns a unique salt each of three times it is run

3.2.4.1.4 Generate hash [generateHash(password, salt)]

3.2.4.1.4.1 Pseudocode

```
function generateHash(password, salt):
```

```
    return sha256(password + salt)
```

```
end function
```

3.2.4.1.4.2 Explanation

Takes in a password and a salt as strings, appends the salt to the end of the string and generates the sha256 hash for this string (python has inbuilt libraries to do this). The hashed value is then returned.

3.2.4.1.4.3 Test data

Test Number	Test	Parameter values	Data types	Expected Outcome
1	Password 1	test, generateSalt()	String, string	Returns string which is the sha256 hash of test and generateSalt() concatenated
2	Password 2	abc123, generateSalt()	String, string	Returns string which is the sha256 hash of abc123 and generateSalt() concatenated
3	Password 3	password, generateSalt()	String, string	Returns string which is the sha256 hash of password and generateSalt() concatenated

3.2.4.1.5 Check password against hashed password [checkPassword(userId, password)]

3.2.4.1.5.1 Pseudocode

```
function checkPassword(userId, password):
```

```
    return checkHash(userId, password, findSalt(userId))
```

```
end function
```

3.2.4.1.5.2 Explanation

Takes in a user ID and a password to check. The salt is then found for the given user and the user id, password and the salt are passed to the checkHash function. The result of the checkHash function (True or False depending on if password matches) is then returned.

3.2.4.1.5.3 Test data

Test Number	Test	Parameter values	Data types	Expected Outcome
1	Valid data	test, test12345	String, string	Returns True
2	Invalid username	qwerty, test12345	String (irrelevant), string	Returns False
3	Invalid password	test, qwerty	String, string (irrelevant)	Returns False
4	Invalid username and password	qwerty, qwerty	String (irrelevant), string (irrelevant)	Returns False

3.2.4.1.6 Check hashed password matches stored hashed password [checkHash(userId, password, salt)]

3.2.4.1.6.1 Pseudocode

```
function checkHash(userId, password, salt):
```

```
    generatedHash = generateHash(password, salt)
```

```
    if (count from users where id = userId and passwordHash = generatedHash) > 0:
```

```

    return True
else:
    return False
end if
end function

```

3.2.4.1.6.2 Explanation

Takes in a user id, an inputted password and a salt provided. These are validated through previous functions. A hash is generated using the pre-written function from the password and salt. The database is then queried and if both the user id and hashed password match an entry in the users table an item is found and 1 is returned so True is returned to indicate the password is valid. If 0 is returned the hash does not match and the password is invalid so False is returned.

3.2.4.1.6.3 Test data

Test Number	Test	Parameter values	Data types	Expected Outcome
1	Valid data	test, test12345, findSalt(test)	String, string, string	Returns True
2	Invalid username	qwerty, test12345, findSalt(test)	String (irrelevant), string	Returns False
3	Invalid password	test, qwerty, findSalt(qwerty)	String, string (irrelevant), string	Returns False
4	Invalid username and password	qwerty, qwerty, findSalt(qwerty)	String (irrelevant), string (irrelevant), string	Returns False

3.2.4.1.7 Get user id [getUserId(username)]

3.2.4.1.7.1 Pseudocode

```
function getUserId(username):
```

```
    return(id from users where username = username)
```

```
end function
```

3.2.4.1.7.2 Explanation

Takes in a username and queries the users table of the database to find the entry with a matching name value. The value in the id column (the primary key for the user) is then returned. Assumes username pre-validated with the checkUserExists function.

3.2.4.1.7.3 Test data

Test Number	Test	Parameter values	Data types	Expected Outcome
1	Valid data	test	String	Returns 1
2	Invalid data	N/A (validation done before using function)	N/A	N/A

3.2.4.1.8 Get account type [getAccountType(userId)]

3.2.4.1.8.1 Pseudocode

```
function getAccountType(userId):
```

```
    return(type from users where id = userId)
```

```
end function
```

3.2.4.1.8.2 Explanation

Takes in a user id and queries the users table of the database to find the entry with a matching id value. The value in the type column is then returned. Assumes pre-validated id passed

3.2.4.1.8.3 Test data

Test Number	Test	Parameter values	Data types	Expected Outcome
1	Valid data	1	Integer	Returns "s"
2	Invalid data	N/A (validation done before using function)	N/A	N/A

3.2.4.1.9 Get username [getUserName(userId)]

3.2.4.1.9.1 Pseudocode

```
function getUserName(userId):
```

```
    return(username from users where id = userId)
```

```
end function
```

3.2.4.1.9.2 Explanation

Takes in a user id and queries the users table of the database to find the entry with a matching id value. The value in the username column is then returned. Assumes pre-validated id passed

3.2.4.1.9.3 Test data

Test Number	Test	Parameter values	Data types	Expected Outcome
1	Valid data	1	Integer	Returns "test"
2	Invalid data	N/A (validation done before using function)	N/A	N/A

3.2.4.1.10 Get first name [getFirstName(userId)]

3.2.4.1.10.1 Pseudocode

```
function getFirstName(userId):
```

```
    return(firstName from users where id = userId)
```

```
end function
```

3.2.4.1.10.2 Explanation

Takes in a user id and queries the users table of the database to find the entry with a matching id value. The value in the firstname column is then returned. Assumes pre-validated id passed

3.2.4.1.10.3 Test data

Test Number	Test	Parameter values	Data types	Expected Outcome
1	Valid data	1	Integer	Returns "te"
2	Invalid data	N/A (validation done before using function)	N/A	N/A

3.2.4.1.11 Get last name [getLastName(userId)]

3.2.4.1.11.1 Pseudocode

```
function getLastName(userId):
```

```
    return(lastName from users where id = userId)
```

```
end function
```

3.2.4.1.11.2 Explanation

Takes in a user id and queries the users table of the database to find the entry with a matching id value. The value in the lastname column is then returned. Assumes pre-validated id passed

3.2.4.1.11.3 Test data

Test Number	Test	Parameter values	Data types	Expected Outcome
1	Valid data	1	Integer	Returns "st"
2	Invalid data	N/A (validation done before using function)	N/A	N/A

3.2.4.1.12 Create account

3.2.4.1.12.1.1 Pseudocode

```
procedure create_account(username, password, firstName, lastName, accountType):
```

```
    salt = generateSalt()
```

```
    insert into users(username=username, passwordSalt=salt, passwordHash=
generateHash(password, salt), firstName=firstName, lastName=lastName, type=accountType)
```

```
end procedure
```

3.2.4.1.12.1.2 Explanation

Takes in a username, password, firstname, lastname and account type. A unique salt is then generated to be used to hash the password before storing it. All values are then inserted into the users table as needed with a hashed password design generated in place of storing the password as string by passing the password with the salt into the generateHash function.

3.2.4.1.12.2 Test data

Test Number	Test	Parameter values	Data types	Expected Outcome
1	Valid data	a, b, c, d, s	String, string, string, string, string	Data correctly inserted into database

2	Invalid data	N/A (validation done before using function)	N/A	N/A
---	--------------	---------------------------------------------	-----	-----

3.2.4.1.13 Update first name in database [updateFirstName(userId, newFirstName)]

3.2.4.1.13.1 Pseudocode

procedure updateFirstName(userId, newFirstName):

```
set first_name = newFirstName in users where id = userID
```

end procedure

3.2.4.1.13.2 Explanation

Takes in a user ID and a new first name. Finds the entry in the users table within the database for the given user id and sets the first name value to the new first name entered

3.2.4.1.13.3 Test data

Test Number	Test	Parameter values	Data types	Expected Outcome
1	Valid user id and new first name	67, test	int, string	Updates first name in database for given user to given name
2	Invalid user id			N/A, should be verified in advance
3	Invalid new first name			N/A, should be verified in advance

3.2.4.1.14 Update last name in database [updateLastName(userId, newLastName)]

3.2.4.1.14.1 Pseudocode

procedure updateLastName(userId, newLastName):

```
set last_name = newLastName in users where id = userID
```

end procedure

3.2.4.1.14.2 Explanation

Takes in a user ID and a new last name. Finds the entry in the users table within the database for the given user id and sets the last name value to the new last name entered

3.2.4.1.14.3 Test data

Test Number	Test	Parameter values	Data types	Expected Outcome
1	Valid user id and new last name	134, test	int, string	Updates last name in database for given user (134) to given name (test)
2	Invalid user id			N/A, should be verified in advance
3	Invalid new last name			N/A, should be verified in advance

3.2.4.1.15 Update password in database [updatePassword(userId, newPassword)]

3.2.4.1.15.1 Pseudocode

procedure updatePassword(userId, newPassword):

```
set password_hash = generateHash(newPassword, findSalt(userID)) in users where id = userID
```

end procedure

3.2.4.1.15.2 *Explanation*

Takes a new password as a string and a user ID. Finds the entry in the users table within the database for the given user id and sets the hashed password value to a hash of the given password using the relevant user's salt.

3.2.4.1.15.3 *Test data*

Test Number	Test	Parameter values	Data types	Expected Outcome
1	Valid user id and new password	45, test	int, string	Updates password in database for given user (45) to given password (test)
2	Invalid user id			N/A, should be verified in advance
3	Invalid new password			N/A, should be verified in advance

3.2.4.1.16 Get list of classes for given user [getClassesOfStudent(*userId*)]

3.2.4.1.16.1 *Pseudocode*

*function getClassesOfStudent(*userId*):*

return id and name in classes where id from users = userID

end function

3.2.4.1.16.2 *Explanation*

Takes in a user ID and returns a list of all class ids and names related to that user

3.2.4.1.16.3 *Test data*

Test Number	Test	Parameter values	Data types	Expected Outcome
1	Valid user id	5	int	Returns list of all classes user is in (will be checked manually before running test)
2	Invalid user id			N/A, should be verified in advance

3.2.4.1.17 Get list of homework for given class [getHomeworkOfClass(*userId*)]

3.2.4.1.17.1 *Pseudocode*

*function getHomeworkOfClass(*classId*):*

join homework with class_homework on id = homework_id

return id and name in homework where class_id from class_homework = classId

end function

3.2.4.1.17.2 *Explanation*

Takes in a class ID and returns a list of all homework ids and names related to that class

3.2.4.1.17.3 *Test data*

Test Number	Test	Parameter values	Data types	Expected Outcome

1	Valid class id	123	int	Returns list of all homework class contains (will be checked manually before running test)
---	----------------	-----	-----	--------------------------------------------------------------------------------------------

3.2.4.1.18 Get score of homework [getHomeworkScore(userId, homeworkId, classId)]

3.2.4.1.18.1 Pseudocode

```
function getHomeworkScore(userId, homeworkId, classId):
    scoreList = (attempts and correct) from question_results in (homework join
    homework_questions on homework.id = homework_questions.homework_id join questions on
    homework_questions.question_id = questions.id join question_results on questions.id =
    question_results.question_id) where (user_id from question_results = userID and id from homework =
    homeworkId)

    questionCount = 0
    score = 0
    for i = 0 to length(scoreList):
        questionCount = questionCount + 1
        if eachScore[1] is True:
            score = score + (1/eachScore[0])
    result = round(score / questionCount *100)
    nameAndDueDate = getHomeworkNameAndDueDate(homeworkId, classId)
    return nameAndDueDate[0], result, nameAndDueDate[1]

end function
```

3.2.4.1.18.2 Explanation

Takes in a user id, homework id and class id and retrieves for the relevant class, user and homework the correct status and attempt count for each question in the homework. Iterates through all question results gathered and adds 1 to the question count for each question and (1/number of attempts) to the score count for each question with a correct status of 1 (only awards marks for correct questions, and marks awarded decreases from 1 for correct first time with each incorrect attempt recorded). The result for the homework is then calculated as a percentage as the score/number of questions multiplied by 100 (i.e. proportion of questions correct with a bias against questions which required multiple attempts). The name of the homework and its due date are then retrieved and returned along with the score

3.2.4.1.18.3 Test Data

Test Number	Test	Parameter values	Data types	Expected Outcome
1	Valid data	87, 50, 778	Int, int, int	Returns ('name50', 28, '2017-9-4')
2	Invalid data (e.g. homework does not exist in class)	87. 49, 778	Int, int, int	Returns ('name49', 'N/A', '2018-5-21')

3.2.4.2 UI Programming

Defined below are all methods which will be required alongside UI code within the Window class

3.2.4.2.1 Constructor method [new()]

3.2.4.2.1.1 Pseudocode

public procedure new():

```
super.new()  
setupUi()  
loginPage = 0  
createAccountPage = 1  
studentMainMenuPage = 2  
teacherMainMenuPage = 3  
questionPage = 4  
homeworkSelectPage = 5  
previousScoresPage = 6  
setHomeworkPage = 7  
adminPage = 8  
accountManagementPage = 9  
viewClassesPage = 10  
changePage(loginPage)  
buttonSetup()  
currentUser = 1
```

end procedure

3.2.4.2.1.2 Explanation

Inherits all properties of the predefined window class provided with PyQt. SetupUi function is then called from PyQt to initialise the window. Constants are then declared to more easily identify what page number in Qt contains what screen to make understanding code easier. The changePage function is then called with loginPage as a parameter to load the landing page on program start-up (this should occur by default as it is page 0 though this is still done in case of unexpected behaviour). buttonSetup is then called to ensure all buttons in the program will run the associated scripts when clicked. Current user is set to a placeholder value which will be replaced with the user id of the user once logged in.

3.2.4.2.2 Change page method [changePage()]

3.2.4.2.2.1 Pseudocode

Public procedure changePage(index):

```

    resetPage(Index)
    window.setIndex(index)

```

end procedure

3.2.4.2.2.2 Explanation

Sets the page index to the desired index to change the viewed page and resets page layout to default.

3.2.4.2.2.2.1 Test Data

Test Number	Test	Parameter values	Data types	Expected Outcome
1	Load login page	0	int	Resets and shows page
2	Load create account page	1	int	Resets and shows page
3	Load student menu page	2	int	Resets and shows page
4	Load teacher menu page	3	int	Resets and shows page
5	Load quiz page	4	int	Resets and shows page
6	Load homework select page	5	int	Resets and shows page
7	Load student results page	6	int	Resets and shows page
8	Load homework management page	7	int	Resets and shows page
9	Load class management page	8	int	Resets and shows page
10	Load edit account page	9	int	Resets and shows page
11	Load view class results page	10	int	Resets and shows page

3.2.4.2.3 Logout method [logout()]

3.2.4.2.3.1 Pseudocode

private procedure logout():

```

    currentUser = 1
    changePage(loginPage)
    hideLogoutButton()

```

end procedure

3.2.4.2.3.2 Explanation

Resets the value of current user back to a placeholder value. The login page is then loaded.

3.2.4.2.3.3 Test Data

Test Number	Test	Parameter values	Data types	Expected Outcome
1	Press logout button	N/A	N/A	Returns to login page and sets user status to loggedout

3.2.4.2.4 Login Page

3.2.4.2.4.1 Login to account [login()]

3.2.4.2.4.1.1 Pseudocode

private procedure login():

```

if loginScripts.checkUserExists(loginUsernameInput.text):

    userId = loginScripts.getUserId(loginUsernameInput.text)

    if loginScripts.checkPassword(userId, loginPasswordInput.text):

        loginSuccessOutput.setText("Success")

        currentUser = userId

        if loginScripts.getAccountType(currentUser) = 't':

            changePage(teacherMainMenuPage)

        else if loginScripts.getAccountType(currentUser) = 's':

            changePage(studentMainMenuPage)

        end if

    else:

        loginSuccessOutput.setText("Invalid password")

        loginPasswordInput.setText("")

    end if

else:

    loginSuccessOutput.setText("Invalid username")

end if

end procedure

```

3.2.4.2.4.1.2 Explanation

Runs script from login scripts to check if the username entered is an existing user.

If true

- Gets the user id for that user and stores it in the temporary variable userId. Next the checkPassword script from the login scripts is run with the user id and the inputted password and checked if valid.

- If password valid
 - Outputs “Success” in output box. currentUser is set to the id of the now logged in user. The account type is then checked using the getAccountType function.
 - If a teacher
 - Loads the teacher main menu page
 - If a student
 - Loads the student main menu page
- If password invalid
 - Outputs “Invalid password” in the output box and clears the password input box.

If false

- Outputs “Invalid username” in output box

3.2.4.2.4.1.3 Test Data

Test Number	Test	Parameter values	Data types	Expected Outcome
1	Login to student account	Student1, password	Str, str	Loads student main menu page and displays correct first name (First1)
2	Login to teacher account	Teacher1, password	Str, str	Loads teacher main menu page and displays correct first name (First1)
3	Invalid username	qwertyuiop, password	Str, str	Displays incorrect username message
4	Invalid password	Student1, qwertyuiop	Str, str	Displays incorrect password message
5	No username	BLANK, password	Str, str	Displays incorrect username message
6	No password	BLANK, password	Str, str	Displays incorrect password message

3.2.4.2.5 Create Account Page

3.2.4.2.5.1 Function to create an account

3.2.4.2.5.1.1 Pseudocode

private procedure createAccount():

```

if usernameInput == ":
    successOutput = "Invalid Username"
else if checkUserExists(usernameInput):
    successOutput = "Username taken"
else if length(firstNameInput) > 100 or length(firstNameInput) < 1:
    successOutput = "Invalid First Name"
else if length(lastNameInput) > 100 or length(lastNameInput) < 1:
    successOutput = "Invalid Last Name"
else if length(passwordInput) < 8:

```

```

successOutput = "Password must be at least 8 characters"

else if passwordInput != passwordVerifyInput:

    successOutput = "Passwords must match"

else if not(accountTypeSelected = 's' or accountTypeSelected = 't'):

    successOutput = "Account type not selected"

else:

    createAccount(usernameInput, firstNameInput, lastNameInput,
getAccountType(currentUser))

    successOutput = "Account Created"

end procedure

```

3.2.4.2.5.1.2 Explanation

Takes the inputs for username, first name, last name, password and password verification (see UI design). If all validation is passed (see test data for conditions), calls function to create account by inserting data into database and outputs a success message. If validation fails, outputs cause of failure and does not create account.

3.2.4.2.5.1.3 Test data

Test Number	Test	Parameter values	Data types	Expected Outcome
1	Valid student account creation	Student12345, password, stu, dent, student	Str, str, str, str, bool	Creates a student account by verifying all inputs are valid and outputs relevant success message. User is added to database
2	Valid teacher account creation	Teacher12345, password, password, tea, cher, teacher	Str, str, str, str, bool	Creates a teacher account by verifying all inputs are valid and outputs relevant success message. User is added to database
3	No username given	BLANK, password, password, stu, dent, student	Str, str, str, str, bool	Does not create account and outputs no username error message
4	Existing username given	Student1, password, password, password, stu, dent, student	Str, str, str, str, bool	Does not create account and outputs username taken error message
5	No first name	Student1, password, password, BLANK, dent, student	Str, str, str, str, bool	Does not create account and outputs no first name error message
6	First name greater than 100 characters	Student1, password, password, q x110, dent, student	Str, str, str, str, bool	Does not create account and outputs invalid first name error message
7	No last name	Student1, password, password, stu, BLANK, student	Str, str, str, str, bool	Does not create account and outputs no last name error message

8	Last name greater than 100 characters	Student1, password, password, stu, q x110, student	Str, str, str, str, bool	Does not create account and outputs invalid last name error message
9	Password less than 8 characters	Student1, qwertyu, qwertyu, stu, dent, student	Str, str, str, str, bool	Does not create account and outputs invalid password error message
10	Non-Matching password verification	Student1, password, qwerty, stu, dent, student	Str, str, str, str, bool	Does not create account and outputs non-matching password error message

3.2.4.2.6 Question Page

3.2.4.2.6.1 Function to set up page

3.2.4.2.6.1.1 Pseudocode

private procedure resetQuestionPage(questionId):

```

radioA.checked = True
radioB.checked = False
radioC.checked = False
radioD.checked = False
textOutput = getQuestionTextOfQuestion(questionId)
correctAnswer = getCorrectAnswerOfQuestion(questionId)
incorrectAnswers = getIncorrectAnswersOfQuestion
correctAnswerLocation = randomInteger(1 to 4)
if correctAnswerLocation = 1:
    radioA.text = correctAnswer
    radioB.text = incorrectAnswers[0]
    radioC.text = incorrectAnswers[1]
    radioD.text = incorrectAnswers[2]
else if correctAnswerLocation = 2:
    radioA.text = incorrectAnswers[0]
    radioB.text = correctAnswer
    radioC.text = incorrectAnswers[1]
    radioD.text = incorrectAnswers[2]
else if correctAnswerLocation = 3:
    radioA.text = incorrectAnswers[0]
    radioB.text = incorrectAnswers[1]
```

```

radioC.text = correctAnswer
radioD.text = incorrectAnswers[2]

else if correctAnswerLocation = 4:
    radioA.text = incorrectAnswers[0]
    radioB.text = incorrectAnswers[1]
    radioC.text = incorrectAnswers[2]
    radioD.text = correctAnswer

end if

end procedure

```

3.2.4.2.6.1.2 Explanation

Selected answer is reset to default (first answer). Fetches for the current question id the question text, the correct answer and each stored incorrect answer from the database. The correct answer is then randomly placed in one of the four answer slots with its position stored for use when verifying a response. The remaining three slots are filled with the remaining incorrect answers in a randomised order.

3.2.4.2.6.1.3 Test data

Test Number	Test	Parameter values	Data types	Expected Outcome
1	Load a question	N/A	N/A	Loading a question page correctly outputs question text and answers in a randomised order

3.2.4.2.6.2 Function to submit question response

3.2.4.2.6.2.1 Pseudocode

```
private procedure submitQuestionResponse():
```

```

uiScripts.incrementUserAttemptAtQuestion(currentUser, questionId)

if (correctAnswerLocation = 1 and radioA = True) or (correctAnswerLocation = 2 and radioB = True) or (correctAnswerLocation = 3 and radioC = True) or (correctAnswerLocation = 4 and radioD = True):
    uiScripts.markQuestionAsCorrect(currentUser, questionId)
    submitButton.enabled = False
    questionFeedbackOutput = "Correct"

else:
    submitButton.enabled = True
    questionFeedbackOutput = "Incorrect"

end procedure

```

3.2.4.2.6.2.2 *Explanation*

Increments attempt count for current user and question as an attempt has been made. If the selected answer radio at time of submission matches position of correct answer, submit button is disabled and user is informed they were correct. The question is also marked as correctly answered by the current user in the database. If the condition is not met, the user is instead told they were incorrect.

3.2.4.2.6.2.3 *Test data*

Test Number	Test	Parameter values	Data types	Expected Outcome
1	Incorrect response	Select incorrect response for loaded question	N/A	Outputs that response was incorrect when submitted
2	Correct response	Select correct response for loaded question	N/A	Outputs that response was correct when submitted and correct status in database is update

3.2.4.2.7 Set Homework Page

3.2.4.2.7.1 *Function to remove question*

3.2.4.2.7.1.1 *Pseudocode*

private procedure removeQuestion():

```

questionId =questions[questionComboBox.index()]
homeworkId = homework[homeowrkComboBox.index()]
uiScripts.removeQuestionFromHomework(questionId, homeworkId)
homeworkRemovedOutput = ("Question removed")

```

end procedure

3.2.4.2.7.1.2 *Explanation*

Gets the question id and homework id from of the selected question and homework in the UI. Runs script to remove question-homework relationship from database using these values. Outputs that homework removal was a success.

3.2.4.2.7.1.3 *Test data*

Test Number	Test	Parameter values	Data types	Expected Outcome
1	Removing from homework which contains questions	Remove button click	Boolean	Question-homework relationship is successfully removed, and success message outputted
2	Removing from homework which contains no questions	Remove button click	Boolean	No question to remove message outputted

3.2.4.2.7.2 Function to add custom question

3.2.4.2.7.2.1 Explanation

See UI design for page layout. Uses script to add question written in DB scripts to insert question into database using relevant input fields in UI. Errors should be raised if any inputs are left blank and database entries not applied. Two out of three incorrect answer fields can be left blank as they are not required though preferred (multiple choice question can be out of two possible answers)

3.2.4.2.7.2.2 Test data

Test Number	Test	Parameter values	Data types	Expected Outcome
1	One incorrect answer given	Qwerty, asdf, a, b, NONE, NONE	String, String, String, String, NONE, NONE	Informs user adding question was a success and inserts it into database
2	Three incorrect answers given	Qwerty, asdf, a, b, c, d	String, String, String, String, String, String	Informs user adding question was a success and inserts it into database
3	No name given	None, asdf, a, b, c, d	NONE, String, String, String, String, String	Informs user that a question name is required
4	No question text given	Qwerty, NONE, a, b, c, d	String, NONE, String, String, String, String	Informs user that question text is required
5	No correct answer given	Qwerty, asdf, NONE, b, c, d	String, String, NONE, String, String, String	Informs user that a correct answer is required
6	No incorrect answers given	Qwerty, asdf, a, NONE, NONE, NONE	String, String, String, NONE, NONE, NONE	Informs user that an incorrect answer is required
7	Matching answers given	Qwerty, asdf, a, a, NONE, NONE	String, String, String, String, NONE, NONE	Informs user that all potential answers should be unique

3.2.4.2.7.3 Function to add automatic question

3.2.4.2.7.3.1 Explanation

Calls relevant function to generate a question of the selected type (see UI design) and writes to database as in database scripts

3.2.4.2.7.3.2 Test data

Test Number	Test	Parameter values	Data types	Expected Outcome
1	Add question	Submit button	Boolean	Generates and inserts question of selected topic into homework and informs user adding question was successful

3.2.4.2.8 Previous Scores Page

3.2.4.2.8.1 Function to fill scores table

3.2.4.2.8.1.1 Pseudocode

```
private procedure updateTable():
```

```
    table.clear()
```

```
    for eachHomework in
```

```
        uiScripts.getHomeworkOfClass(currentClasses[classComboBox.Index()]):
```

```
            data = uiScripts.getHomeworkScore(currentUser, eachHomework[0], self.  
currentClasses[classComboBox.Index()]])
```

```
            table.insertRow(data[0], data[1], data[2])
```

```
        end for
```

```
    end procedure
```

3.2.4.2.8.1.2 Explanation

For the selected class, each homework in the class is fetched from the database using the getHomeworkOfClass function. For each homework, homework data for the current user in the current class is fetched using the getHomeworkScore function and a row is inserted into the table containing homework name, user score and homework date.

3.2.4.2.8.1.3 Test data

Test Number	Test	Parameter values	Data types	Expected Outcome
1	Load page	N/A	N/A	Loading page lists all homework for the given class along with the date and the correctly calculated score (for detail on score calculation see database scripts section)

3.2.4.2.9 Admin Page

3.2.4.2.9.1 Function to add user to class

3.2.4.2.9.1.1 Explanation

Calls function written earlier in database scripts to add user to class according to selected items on page (see UI design)

3.2.4.2.9.1.2 Test data

Test Number	Test	Parameter values	Data types	Expected Outcome
1	Add user not in class	student1	String	Adds given student to class and informs user of success
2	Add non-existent user	NONE	NONE	Informs user that no such student exists
3	Add user already in class	student1	String	Informs user that student is already in class

3.2.4.2.9.2 Function to remove user from class

3.2.4.2.9.2.1 Explanation

Calls function written earlier in database scripts to remove user according to selected items on page (see UI design)

3.2.4.2.9.2.2 Test data

Test Number	Test	Parameter values	Data types	Expected Outcome
1	Remove user in class	User from dropdown	String	User selected is removed from class

3.2.4.2.9.3 Function to create class

3.2.4.2.9.3.1 Explanation

Calls function written earlier in database scripts to create class according to input box on page (see UI design)

3.2.4.2.9.3.2 Test data

Test Number	Test	Parameter values	Data types	Expected Outcome
1	Create valid class	Testclass	String	Creates new class with given name under current teacher
2	No class name given	NONE	NONE	Informs user that class to be added must be named

3.2.4.2.9.4 Function to remove class

3.2.4.2.9.4.1 Explanation

Calls function written earlier in database scripts to remove class according to selected items on page (see UI design)

3.2.4.2.9.4.2 Test data

Test Number	Test	Parameter values	Data types	Expected Outcome
1	Remove class	Class from dropdown selected	String	Removes selected class from database

3.2.4.2.9.5 Function to create homework

3.2.4.2.9.5.1 Explanation

Calls function written earlier in database scripts to remove create homework according to selected items on page (see UI design)

3.2.4.2.9.5.2 Test data

Test Number	Test	Parameter values	Data types	Expected Outcome
1	Add valid homework	Qwerty, asdf, 1/1/2020	String, string, date	Inserts new homework into class with given name, description and due date

2	No homework name given	NONE, asdf, 1/1/2020	NONE, string, date	Informs user that a name is required
3	No description given	Qwerty, NONE, 1/1/2020	String, NONE, date	Informs user that a description is required
4	Past due date	Qwerty, asdf, DAY BEFORE CURRENT DATE AT TIME OF TESTING	String, string, date	Informs user date must be in the future

3.2.4.2.9.6 Function to remove homework

3.2.4.2.9.6.1 Explanation

Calls function written earlier in database scripts to remove homework according to selected items on page (see UI design)

3.2.4.2.9.6.2 Test data

Test Number	Test	Parameter values	Data types	Expected Outcome
1	Remove valid homework	Homework selected from dropdown	String	Removes selected homework from class

3.2.4.2.10 Account Management Page

3.2.4.2.10.1 Function to update account details

3.2.4.2.10.1.1 Explanation

See UI design for page layout. Uses pre-written scripts (see DB scripts) to verify existing password matches that in database. For any input box with data, scripts in DB scripts are used to change the database entry and leaves it alone if the input is blank. For password to be changed, the value in verify password must match the new password

3.2.4.2.10.1.2 Test data

Test Number	Test	Parameter values	Data types	Expected Outcome
1	No new first name given	NONE, b, password, c, c	NONE, string, string, string, string	Leaves first name unchanged
2	No new last name given	a, NONE, password, c, c	string, NONE, string, string, string	Leaves last name unchanged
3	No new password given	a, b, password, NONE, NONE	string, string, string, NONE, NONE	Leaves password unchanged
4	New first name	Abc, Def, password, qwertyuiop, qwertyuiop	string, string, string, string, string	Updates first name to Abc
5	New last name	Abc, Def, password, qwertyuiop, qwertyuiop	string, string, string, string, string	Updates last name to Def

6	New password	Abc, Def, password, qwertyuiop, qwertyuiop	string, string, string, string, string	Updates password to qwertyuiop
7	Incorrect current password	Abc, Def, abcdef, qwertyuiop, qwertyuiop	string, string, string, string, string	Refuses to apply any changes and informs user password is incorrect
8	Mismatching password verification	Abc, Def, password, qwertyuiop, abc	string, string, string, string, string	Informs user new passwords do not match and does not update password
9	Invalid new password (less than 8 characters)	Abc, Def, password, qwertyu, qwertyu	string, string, string, string, string	Informs user password is invalid

3.2.4.2.11 View Classes Page

3.2.4.2.11.1 Function to update results table

3.2.4.2.11.1.1 Pseudocode

```
private procedure viewClassesUpdateTable():
```

```



```

3.2.4.2.11.1.2 Explanation

Clears any existing data from the table, gets score data using the function previously defined to get the all student results for the currently selected class and homework. Unless there are no scores, the list is then iterated through with each score being placed in the table as a new row.

3.2.4.2.11.1.3 Test data

Test Number	Test	Parameter values	Data types	Expected Outcome
1	Class view selected	Class view selected from dropdown	String	Calculates and outputs all required score data for selected class and homework
2	Student view selected	Student view selected from dropdown	String	Calculates and outputs all required score data for selected class and student

3.2.4.2.12 Homework Select Page

3.2.4.2.12.1 Function to update homework list

3.2.4.2.12.1.1 Explanation

Calls function written earlier in database scripts to remove retrieve homework data according to selected items on page (see UI design)

3.2.4.2.12.1.2 Test data

Test Number	Test	Parameter values	Data types	Expected Outcome
1	Load page	N/A	N/A	Loads all homework for selected class with future due date

3.2.5 Algorithms to be used

3.2.5.1 Libraries

- PyQt5 library
 - QMainWindow
 - Pre-set class which interacts with a QT .ui file to setup a class containing all relevant procedures and constants needed to initialise and interact with a QT GUI
 - QtChart
 - Allows for the creation and output of graphs
- Math library
 - sin function
 - Takes in a value and returns the sine of that value
 - cos function
 - Takes in a value and returns the cosine of that value
 - tan function
 - Takes in a value and returns the tangent of that value
 - asin function
 - Takes in a value and returns the arc sine of that value
 - acos function
 - Takes in a value and returns the arc cosine of that value
 - atan function
 - Takes in a value and returns the arc tangent of that value
 - degrees function
 - Takes in a value in radians and returns the same value in degrees
 - Returns $(value * 180 / \pi)$
 - radians function
 - Takes in a value in degrees and returns the same value in radians
 - Returns $(value * \pi / 180)$
- SymPy
 - Allow for the expression of mathematical functions in terms of a variable and easy use of this equation (e.g. evaluate for a given value, solve, or integrate/differentiate)

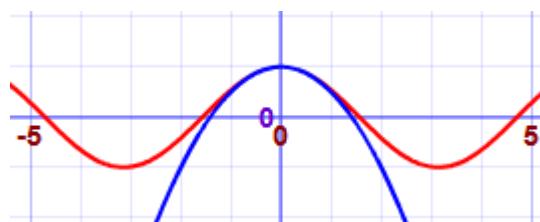
3.2.5.2 Mathematics

3.2.5.2.1 Taylor expansion

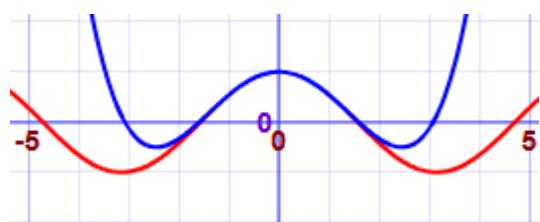
An algorithm to perform a Taylor expansion of a given expression as it allows for the relatively simple approximation of the value of complex mathematical function which cannot easily be evaluated such as sine and cosine waves (or any other function which can be differentiated infinitely, referred to in mathematics as an “infinitely differentiable” function). Additionally, it is included within the Further Maths specification and so could have questions generated about it for quizzes.

A Taylor expansion works by taking an expression in terms of a variable (to be referred to from now on as x) and expressing it as an infinite series of less complex terms in terms powers of x multiplied by fractions. The more terms that are found in the infinite series, the more accurate the approximation of the function for a given value of x as more turning points are introduced to the curve defined by the function, allowing it to be brought closer to the infinite number of curves on the true function. The nature of a Taylor expansion means that it is more accurate when the parameter passed is closer to the centre of expansion as terms are generated getting further and further away from this point as eventually the generated finite series runs out of turning points to keep it approximating the complex original curve, with more and more terms needing to be generated before an accurate approximation can be found for values of larger magnitude (in theory an infinite number of terms would be needed to accurately find a value infinitely far away from the centre of expansion). Below is shown images of a graphical representation of the Taylor expansion of $\cos(x)$ with the blue line showing the approximation, showing how as more terms are added the approximation approaches the true line of the true value of the function (red) further and further from the centre of expansion:

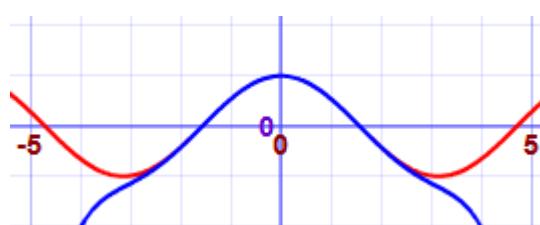
1 term in expansion



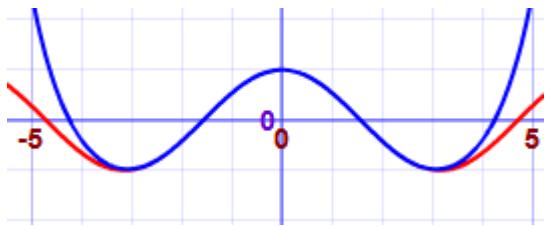
2 terms in expansion



5 terms in expansion



4 terms in expansion



Performing an expansion is done using the following process:

- 1) Take the initial function and substitute the centre of expansion (from here this will be expressed as a) into this to get an integer value for the first term
- 2) To get each successive term (term number will be expressed as n from here), differentiate the function used in the previous term, substitute in the previous term and multiply by $((x-a)^n)/n!$
- 3) Each successive term can be added to the sum of all the previous terms to get an equation in x for which an approximate value of the original function will be given when a value of x is substituted in

This can be represented with the following pseudocode which takes in a function in terms of x and the number of terms in the expansion to be generated (SymPy library will be used to evaluate functions and perform differentiation):

```
function TaylorExpansion(function, centre, terms):
```

```

    finalEquation = 0
    previousFunction = function
    for n = (0 to terms):
        f = evaluate(previousFunction at x = centre)
        finalEquation = finalEquation + (f*((x-centre)^terms)/terms!))
        previousFunction = derivative(previousFunction)
    end for
    return finalEquation

```

```
end function
```

There is a case where a Taylor expansion is invalid, that being when the value of each derivative at $x=a$ is 0 (e.g. the derivative of $e^{(x^2)}$ is $2xe^{(x^2)}$ which equals 0 when $x=a=0$ and will continue to equal 0 as each further derivative will involve multiplication by 0 when evaluated) and so the series will terminate prematurely. This can be addressed by picking a value for the centre of expansion where the value of each derivative is not 0 and so the expansion will not terminate. Relevant error checking will be needed to avoid this problem.

3.2.5.2.2 Maclaurin expansion

A Maclaurin series is just a special case of the Taylor series for which the centre of expansion is 0. This means the series is simpler to compute as each calculated addition to the series is only in terms of a single power of x and so further processing to expand $(x-a)^n$ and to simplify the expression is not needed as each term will simply be in terms of x^n .

This can be represented by the following pseudocode:

```

function MaclaurinExpansion(function, terms):
    finalEquation = 0
    previousFunction = function
    for n = (0 to terms):
        f = evaluate(previousFunction at x = 0)
        finalEquation = finalEquation + (f*((x^terms)/terms!))
        previousFunction = derivative(previousFunction)
    end for
    return finalEquation
end function

```

When programming this, as the code is so like a Taylor expansion, object-oriented programming could be used to create a Maclaurin class as a child of a Taylor class and through polymorphism the centre of expansion can be automatically set to 0 when a Maclaurin instance is initialised.

The same caveat applies to a Maclaurin expansion as to a Taylor expansion. This means that, as the centre of expansion cannot be moved certain functions simply cannot be expanded and so error checking will need to be included to ensure that functions entered for a Maclaurin expansion are valid before an attempt is made (an example of one such function is given in the explanation of the limitations of Taylor expansion)

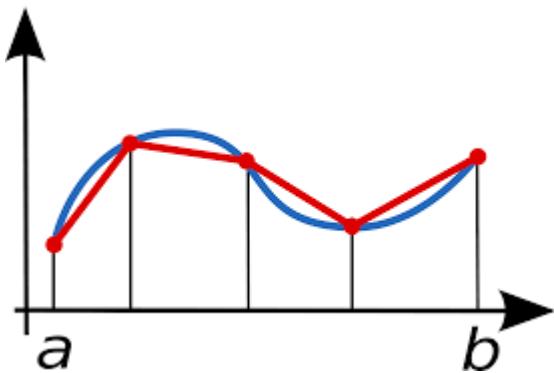
3.2.5.2.2.1 Sine and cosine functions

One example of a place where a Maclaurin expansion is valid is with sine and cosine functions as each are the derivative of the other (derivative of $\sin(x)$ is $\cos(x)$ and derivative of $\cos(x)$ is $-\sin(x)$) and, while $\sin(0) = 0$, $\cos(0) = 1$ and so terms in the Maclaurin series can be generated for every other derivative as the values of sine and cosine are known from the definition of them. The above definition of a Maclaurin expansion can therefore be used to approximate the values of each function for any value of x for which the values of $\sin(x)$ and $\cos(x)$ are not known. In addition to this, as sine and cosine waves are repeating functions every 2π values of x , precision can be ensured as multiples of 2π can be taken off any value of x given to get it within the range 0 to 2π , an area within which the Maclaurin expansion can be much more accurately approximated than further away from the centre.

If this is implemented, the math library is no longer needed to find values of $\sin(x)$ and $\cos(x)$ as my own code will be usable.

3.2.5.2.3 Trapezium rule

The trapezium rule is a method of approximating an integral by approximating a curve as a series of straight lines of fixed width and calculating the area underneath these straight lines, essentially by working out the areas of trapezia:



The area of a single trapezium is equal to the mean vertical side length multiplied by the width ($0.5 * (y_1 + y_2) * \text{change in } x$). The area under a curve with given limits a and b is therefore approximately equal to the sum of all trapezia under it between $x=a$ and $x=b$. This can be simplified into a single easier equation as follows which simplifies the area of all trapezia into a single equation (assuming width is kept constant):

$$\int_{x_0}^{x_n} f(x) dx = \frac{1}{2} h [(y_0 + y_n) + 2(y_1 + y_2 + \dots + y_{n-1})]$$

In the above equation, h is strip width and can be calculated by taking the lower limit from the upper limit and dividing by the desired number of strips. Each y value is then calculated by, starting with the lower limit, evaluating the equation of the curve at a x values increasing each time by the strip width up until the upper limit is reached. The first and last terms are used once in the equation as they each only appear as the first and last side length of the first and last trapezia respectively, whereas each other y value is both the last side length of one trapezium and the first of the other and so is added twice.

This equation can be expressed as the following pseudocode:

```

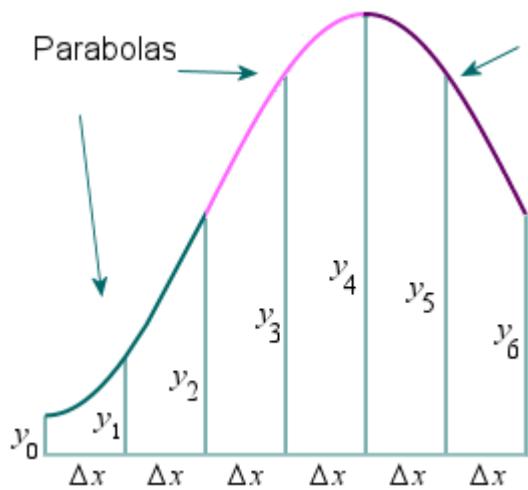
function TrapeziumRule(function, lowerLimit, upperLimit, stripCount):
    h = (upperLimit - lowerLimit) / stripCount
    ySum = evaluate(function at x = lowerLimit) + evaluate(function at x = upperLimit)
    for n = 1 to stripCount - 1:
        ySum = ySum + 2 * evaluate(function at x = lowerLimit + n * h)
    end for
    return ySum * 0.5 * h
end function

```

3.2.5.2.4 Simpson's rule

Simpson's rule is somewhat like the trapezium rule in that it splits a curve into multiple regions of even width and finds the area under these simpler regions. The difference however is that a straight line is not approximated between each y value, but a parabola through a set of three y values (this means that an even number of strips are required for the function to work however). This means

that between y values, the approximated area of the curve is closer to the true value as the curved line can better fit to the line of the curve of the true function.



The area under each parabola is calculated by taking the x and y co-ordinates of three points on the line horizontally equidistant (with the separation between each x value being defined in the same way as in the Trapezium rule as $(\text{upper limit} - \text{lower limit})/\text{number of strips}$), and using them to solve the equation $y=ax^2+bx+c$ for a , b and c by substituting in each pair of x and y values to construct three equations which can be solved simultaneously. Putting these values of a , b and c back into the equation $y=ax^2+bx+c$ gives the equation of the parabola passing through all these points (see pink line above). The area under this parabola within the range of the three x values substituted as the integral of the parabola equation with upper limit of the highest of the x values (middle x value + h , where h is strip width) used and lower limit of the lowest limit (middle x value + h). This integral can easily be calculated to be of the general form $(ax^3)/3 + (bx^2)/2 + cx + d$, where a , b and c are the same as in the original equation. When the definite integral with the given limits is calculated, this gives $((a(x+h)^3)/3 + (b(x+h))/2 + c(x+h)) - ((a(x-h)^3)/3 + (b(x-h))/2 + c(x-h))$ where x is the middle x value. This simplifies to $h/3(2ah^2+6c)$. This process can be repeated for each set of three x values to get the total approximate area under the integral. All of this can be simplified to give a single equation to find the total area under the entire region in one go:

$$\int_a^b y dx \approx \frac{h}{3} (y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 + \dots + 2y_{n-2} + 4y_{n-1} + y_n)$$

Where h is strip width.

This can be expressed as the following pseudocode:

```
function SimpsonsRule(function, lowerLimit, upperLimit, stripCount):
```

```
    h = (upperLimit - lowerLimit) / stripCount
```

```
    ySum = evaluate(function at x = lowerLimit) + evaluate(function at x = upperLimit)
```

```
    for n = 1 to stripCount - 1:
```

```
        if n MOD 2 = 0:
```

```
            ySum = ySum + 2 * evaluate(function at x = lowerLimit + n * h)
```

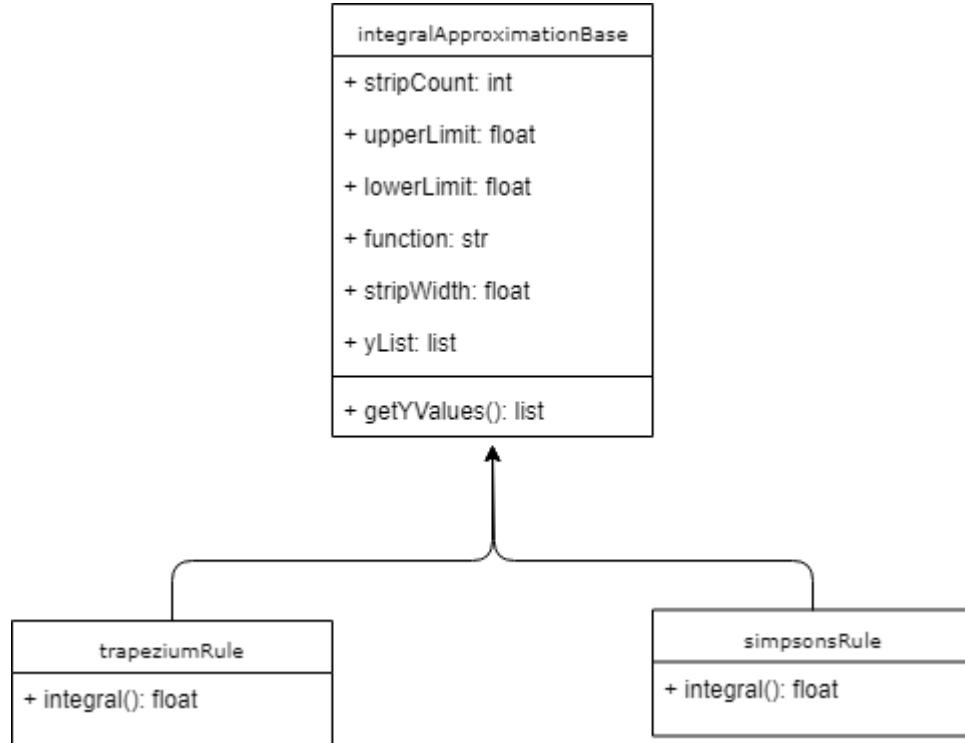
```

else:
    ySum = ySum + 4 * evaluate(function at x = lowerLimit + n * h)
end if
end for
return ySum * (h/3)

end function

```

3.2.5.3 Proposed class structures



3.3 USABILITY FEATURES

- Sufficiently large text
 - Easy to read, even from beyond a typical screen distance and by those who may have a slight visual impairment (more advanced disability features such as text to speech are however beyond the scope of this program), though not so large as to be obstructive of other elements
- Large buttons
 - Easy to select with a typical cursor or even a finger if used on a device with a touchscreen. Large buttons also ensure that not too much precision is needed to select elements, for example for those with more limited fine motor skills (e.g. those with a movement impairing disability)
- High contrast between elements
 - All text is kept dark and backgrounds light, with text and background also of different colour, to maximise readability even for those with visual impairments such as colour blindness
- Clear indication of destructive features

- Features to indicate if a button press will delete data to cause hesitation before pressing by colouring the button red and using capital letters
- Radio buttons
 - Allows for the easy selection of an answer from a set of possible answers
 - Superior to a checkbox for required implementation of a multiple choice quiz with only one correct answer as only a single radio can be selected at a time whereas more than one checkbox can be selected, avoiding the need for extra validation to ensure only a single checkbox is selected by making an invalid selection impossible
- Password reset
 - Allows a user to reset their password using a teacher of one of their classes in order to reduce the chance of complete loss of access to an account in the event of a password being forgotten
- Tables to present results
 - Data from the database can be pulled as needed and represented in a table format where user scores are to be viewed within the program to clearly and space-efficiently present many results on screen at once without compromising the ease with which the output can be understood. This is ensured by keeping all related data, and only related data, within the same row in the table, and all columns titled appropriately and with only data matching that field included within the column.

3.4 APPROACH TO TESTING

3.4.1 During Development

After the development of each function it will be tested individually to ensure it always behaves as expected. This will be done by passing through all possibilities of valid data and ensuring the correct result is produced. Following this, the function will then be fed invalid data of various types to ensure it handles the errors correctly. If the function fails at any of these stages, it will be tweaked and then re-tested again at all stages. This will repeat until all errors are eliminated. Testing criteria will be based on the test data defined below.

By this process when larger functions are developed utilising other pre-tested functions, it can be almost certainly ensured that any errors which occur in the testing of the newer function are in fact an error in the way it has been developed and not the result of a function it is dependent on, making it much less likely for hard to track problems surfacing from functions many layers deep.

All testing will be logged, as well as every occurrence of errors and how they were fixed.

How each function will be tested has been included alongside relevant pseudocode. Test data is either as stated in the data structure or, where not stated, will be generated in compliance with the data structure at the start of development when populating the database.

3.4.2 Post Development

3.4.2.1 Overview

All objectives will be checked and verified to still be functional after testing during development. More general objectives about the usability of the program will then be assessed and tweaks made to the program where necessary.

Stakeholders, a teacher and student, will also be consulted to assess that the program meets their requirements and any suggestions for improvements will be noted for evaluation.

Post development, all testing will be limited to what can be directly tested from within the program itself, and not individual functions by themselves in a contained environment as these will be assumed to be still working since testing during development. The focus of testing here is for usability and direct functionality.

3.4.2.2 Tests

3.4.2.2.1 Robustness

Test Number	Test	Test method	Expected outcome
1	Pages change within a reasonable amount of time	All page transition buttons will be rapidly clicked when used and ensured to still complete in a reasonable amount of time to the stakeholder each time	Pages change near instantly any time transitions should occur
2	Deleting database while in use	Delete file using file explorer whilst a student is attempting a quiz	File cannot be removed

3.4.2.2.2 Usability

Test Number	Test	Expected outcome	Stakeholder to perform test	Decomposition of problem area covered
1.	Go to create account screen	Loads create account screen	Either	1d
2.	Create teacher account	Teacher account permanently created	Teacher	6ai, 6aii, 6aiii, 6aiv, 6av
3.	Create student account	Student account permanently created	Student	
4.	Return from account creation to login screen	Loads login screen	Either	6e
5.	Teacher login	Loads teacher main menu as relevant teacher	Teacher	1a, 1b, 1c, 6d
6.	Student login	Loads student main menu as relevant student	Student	3
7.	Logout button	Returns to login screen and unselects current user	Either	2a, 2b, 5a
8.	Go to account management	Loads account management page	Either	7ai

9.	Set new first name, last name and password	Sets new first name and last name and only new password works when attempting to login with the edited account	Either	7ai1, 7ai2, 7ai3, 7ai4, 7ai5
10.	Home button	When clicked returns to main menu	Either	5b
11.	Create class	Enter a class name and new class is created	Teacher	7cii1a, 7cii1b
12.	Add user to class	Enter username and adds given username to class	Teacher	7cii2ai, 7cii2av
13.	Remove user from class	Select student from list and removes user from class	Teacher	7cii2bi, 7cii2bii
14.	Add homework to class	Enter homework name, description and due date and inserts homework into class	Teacher	7cii2ci, 7cii2cii, 7cii2ciii, 7cii2civ, 7cii2cv
15.	Remove homework from class	Select from list of homework in class and remove homework from database and class	Teacher	7cii2di, 7cii2dii, 7cii2diii
16.	Delete class	Select from list of classes and remove from database	Teacher	7cii3a, 7cii3b
17.	Add automatic question to homework	Select class and homework to add question to and question topic and difficulty to add an automatically generated question to the class	Teacher	7ciii1, 7ciii2, 7ciii3a, 7ciii3b, 7ciii3c, 7ciii3d

18.	Add custom question to homework	Select class and homework to add question to and enter question name, text, answer and incorrect answers and add to homework	Teacher	7ciii4a, 7ciii4b, 7ciii4c, 7ciii4d, 7ciii4e
19.	Remove question from homework	Select question to remove from list, show question details and when remove selected remove from database	Teacher	7ciii5a, 7ciii5b, 7ciii5c
20.	Select homework	Select class to do homework from and select homework to do from a list of homework, showing the due date. Go to quiz when selected and show first question	Student	7bii1, 7bii2, 7bii3, 7bii4, 8a
21.	Navigate quiz	Next question loads next question and previous loads previous	Student	8bi, 8bii
22.	Question output	Must show question text and four potential answers, allowing only 1 to be selected, submitted and a response given	Student	8c, 8di, 8dii, 8eii1, 8eii2
23.	View previous homework results	Show list of homework with name, score and date	Student	7bi1a, 7bi1b, 7bi1c
24.	View class scores	Select class view, select class, select homework and output a list of results showing for each	Teacher	7ci1, 7ci2, 7ci3, 7ci4a, 7ci4b, 7ci4c, 7ci4d, 7ci4e

		student: first name, last name, raw score, weighted percentage and number of attempts		
--	--	---------------------------------------------------------------------------------------	--	--

3.5 STAKEHOLDER RESPONSE TO DESIGN

Stakeholders were allowed to view the design documentation and asked to give any relevant feedback

3.5.1 Troy Garvin, teacher of Mathematics

The UI mock-ups given seem reasonably intuitive, though perhaps a few are a little cluttered and could be tidied up visually. It also seems as if a little more feedback could be given when performing actions in the program as to whether it has been successful or not. Functionality wise, everything seems great as I can view a lot of data about how my students are performing, and I will never have to worry about having to reuse the same questions

3.5.2 Tommy Clair, Year 12 student currently studying A Level Maths and Further Maths

The quiz seems well laid out and simple to use and navigate. I like that it is multiple choice as many programs we've used in the past have been inconsistent with the formatting of your inputs, losing marks for correct answers, so it's good that I won't be penalised for that. Selecting a quiz also seems as straightforward as it could be, and I really like that I'll be able to look at a list of every homework I've ever done and be able to see where I can improve. This should all be very helpful.

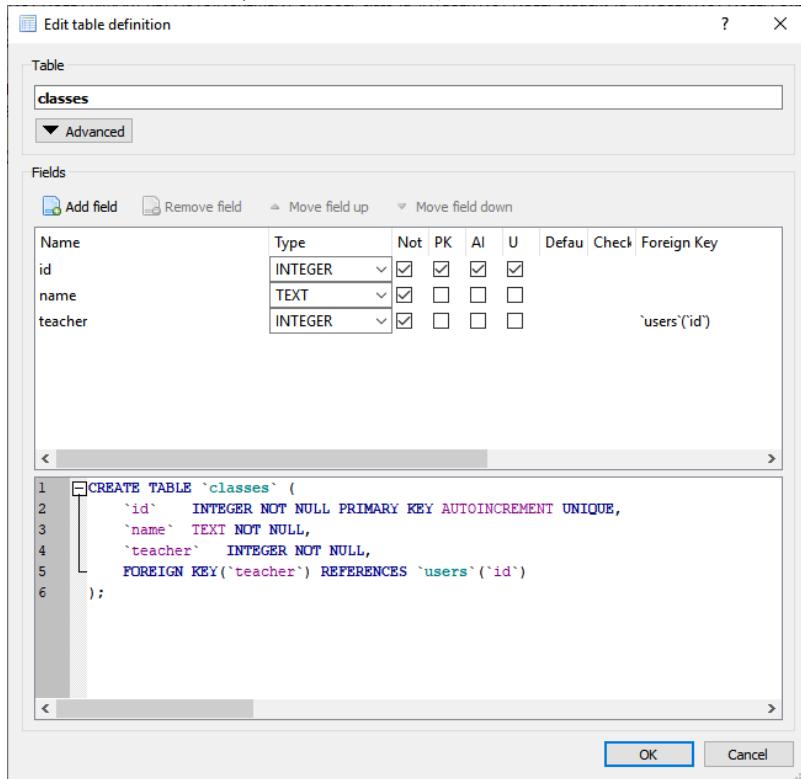
3.5.3 Analysis of responses

Overall it seems like the program's design has been created in a way that is suitable for the intended end users. Some criticism has been given to the cluttered nature of some pages in the program, however this should be simple to fix by separating functionalities into separate labelled tabs, so they are not all shown at once and it is clearer how to use them (this is mostly relevant on the teacher side of the program). Extra text outputs will also be included in the program to help clear up any areas where it may be unclear whether an action has been completed successfully or not

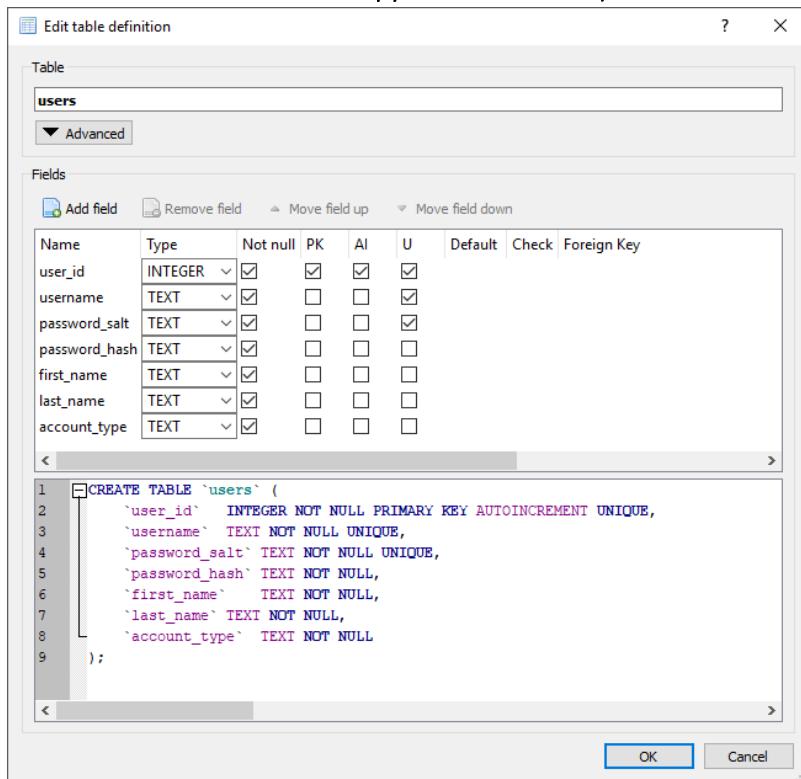
4 DEVELOPMENT OF SOLUTION

4.1 DATABASE SETUP

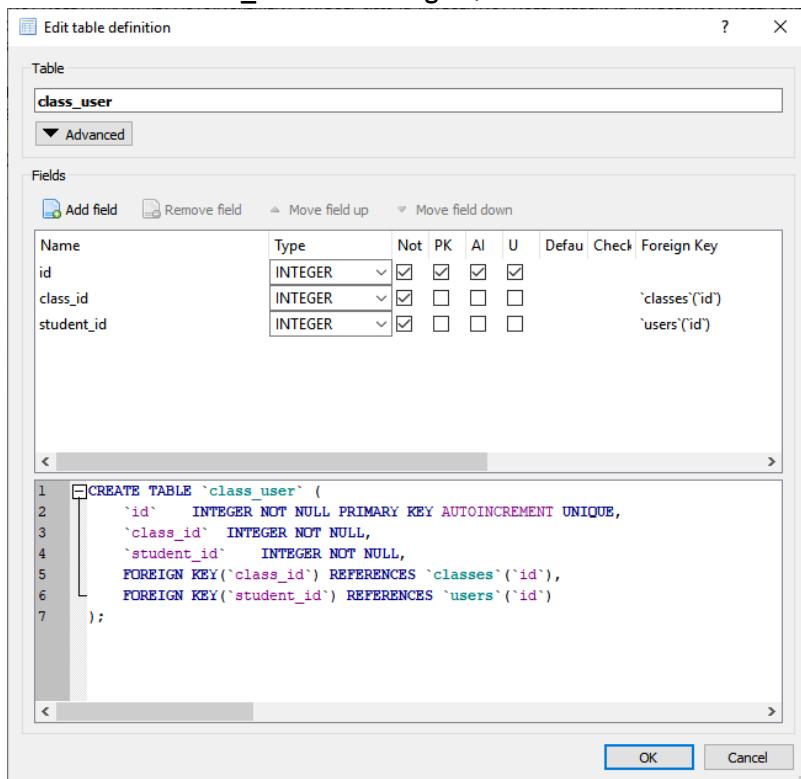
- 4.1.1 Create classes table using SQLite DB Browser GUI (renamed from class in design as plural is more descriptive of contents)



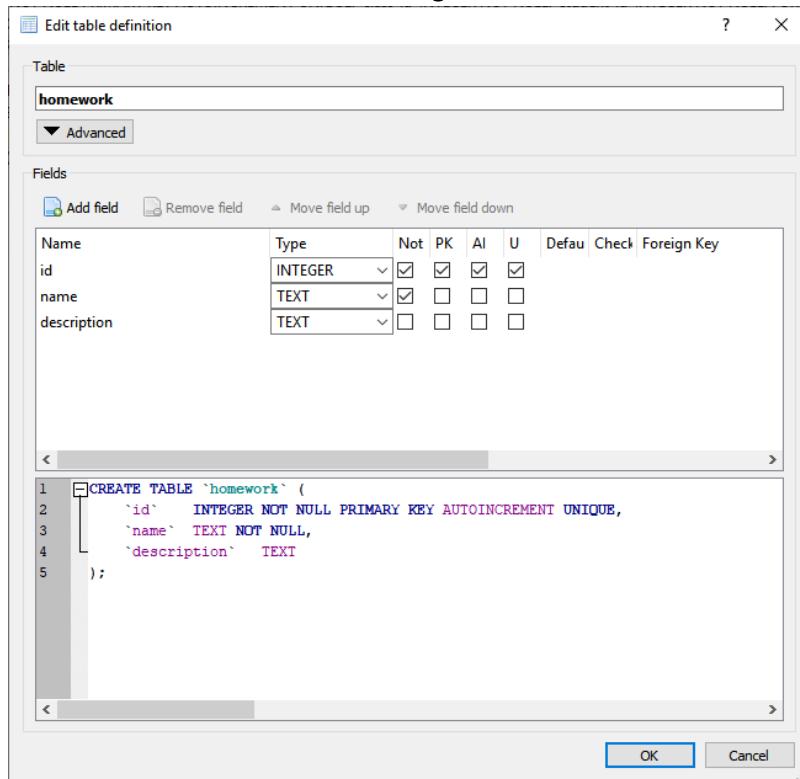
4.1.2 Create users table using SQLite DB Browser GUI (Code generated can later be used to automate creation of database in python if not found)



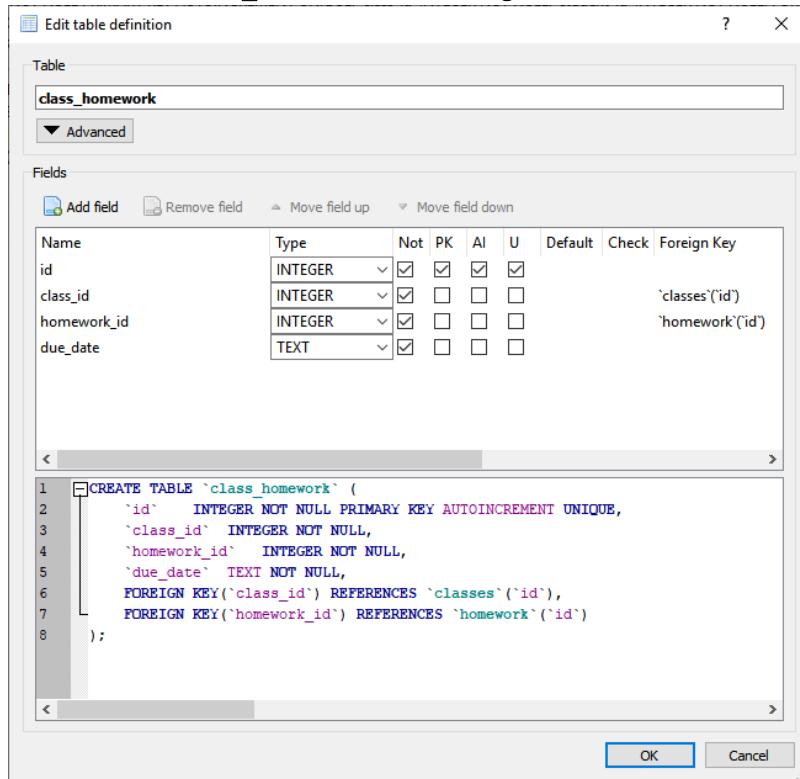
4.1.3 Create class_user table using SQLite DB Browser GUI



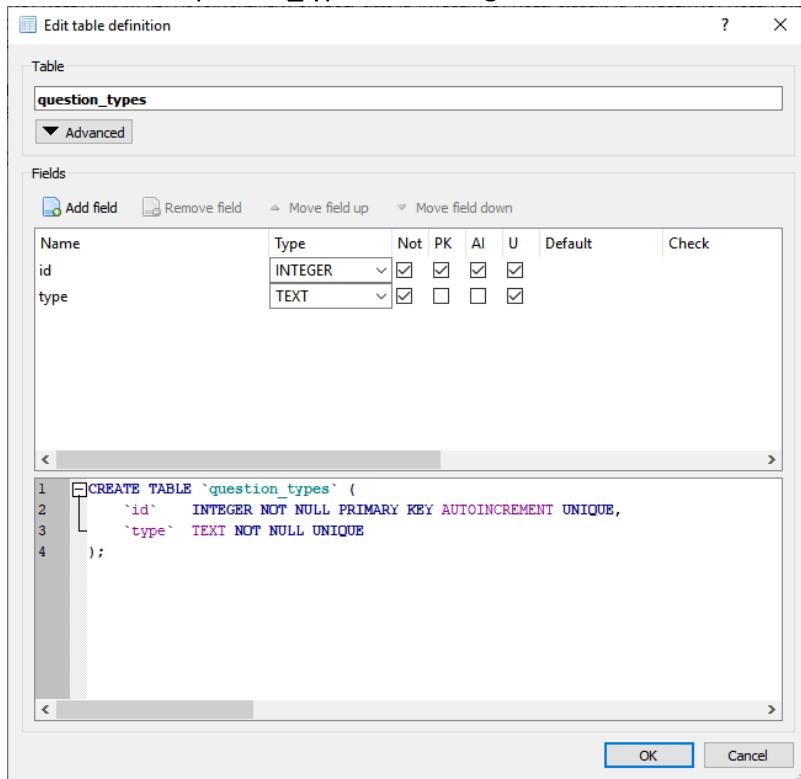
4.1.4 Create homework table using SQLite DB Browser GUI



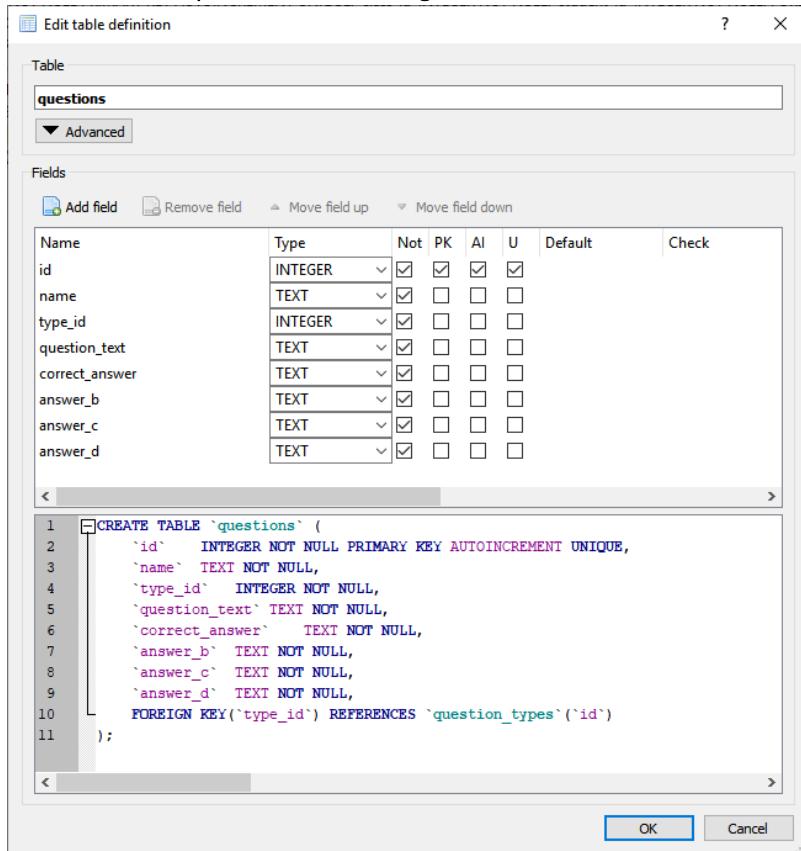
4.1.5 Create class_homework table using SQLite DB Browser GUI



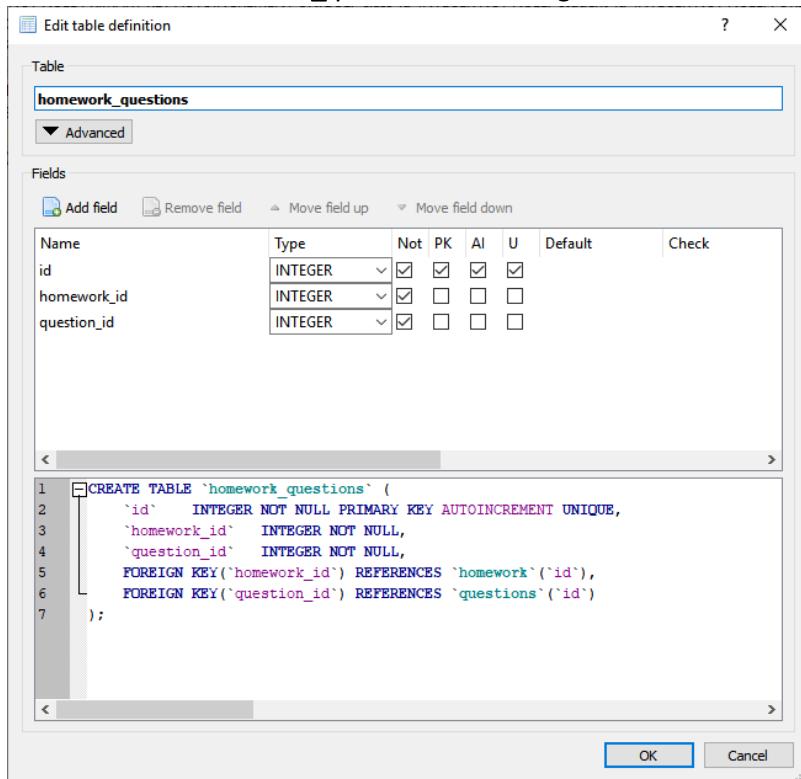
4.1.6 Create question_types table using SQLite DB Browser GUI



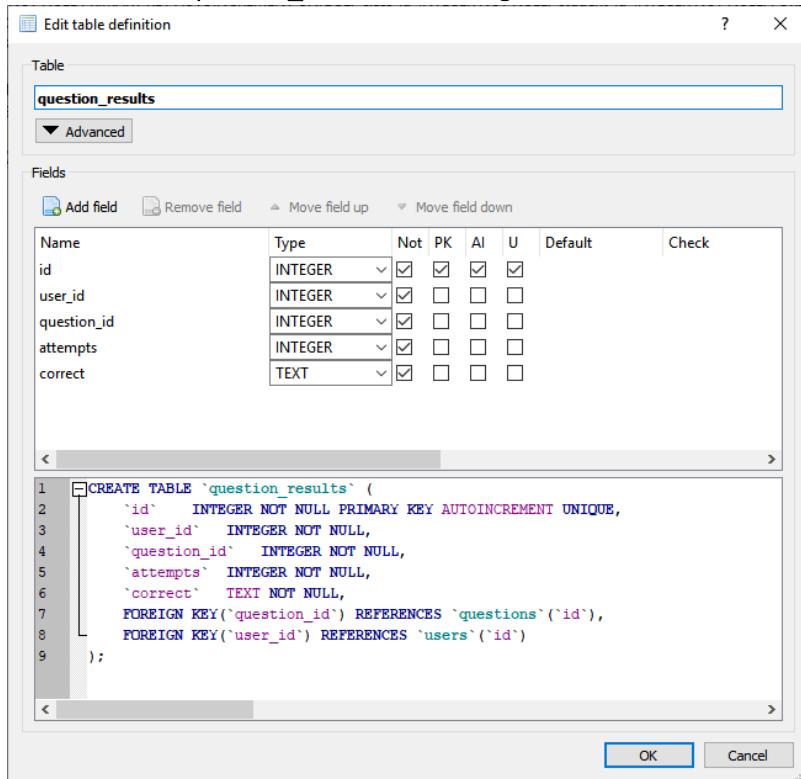
4.1.7 Create questions table using SQLite DB Browser GUI



4.1.8 Create homework_questions table using SQLite DB Browser GUI



4.1.9 Create question_results table using SQLite DB Browser GUI



4.2 POPULATING DATABASE (TESTING.DB_POPULATE)

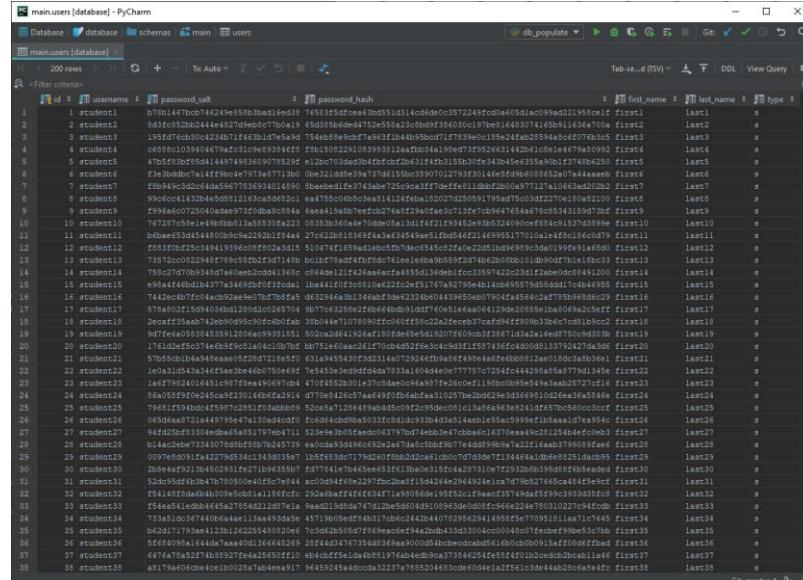
All code written below is designed to format each field in the way described in the data structure and all relationships are designed in accordance with the rules of the database (if rules are contradicted SQLite will prevent database changes made from being committed)

4.2.1 Populate users table

4.2.1.1 Code

```
def user():
    # Creates 100 unique valid entries in user table with type student
    identity = 1
    while identity <= 100:
        username = 'student{}'.format(identity)
        password_salt = scripts.db_scripts.generate_salt()
        password_hash = scripts.db_scripts.generate_hash('password', password_salt)
        first_name = 'first{}'.format(identity)
        last_name = 'last{}'.format(identity)
        user_type = 's'
        sql = 'INSERT INTO users(username, password_salt, password_hash, first_name, last_name, type)' \
              'VALUES(?, ?, ?, ?, ?, ?);'
        c.execute(sql, (username, password_salt, password_hash, first_name, last_name, user_type))
        identity += 1
    # Creates 100 unique valid entries in user table with type teacher
    identity = 1
    while identity <= 100:
        username = 'teacher{}'.format(identity)
        password_salt = scripts.db_scripts.generate_salt()
        password_hash = scripts.db_scripts.generate_hash('password', password_salt)
        first_name = 'first{}'.format(identity)
        last_name = 'last{}'.format(identity)
        user_type = 't'
        sql = 'INSERT INTO users(username, password_salt, password_hash, first_name, last_name, type)' \
              'VALUES(?, ?, ?, ?, ?, ?);'
        c.execute(sql, (username, password_salt, password_hash, first_name, last_name, user_type))
        identity += 1
    conn.commit()
```

4.2.1.2 Result



```
main.users [database] - PyCharm
Database Database schemas main users
mamusers [database]
200 rows
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 379 380 381 382 383 384 385 386 387 388 389 389 390 391 392 393 394 395 396 397 398 399 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 459 460 461 462 463 464 465 466 467 468 469 469 470 471 472 473 474 475 476 477 478 479 479 480 481 482 483 484 485 486 487 488 489 489 490 491 492 493 494 495 496 497 498 499 499 500 501 502 503 504 505 506 507 508 509 509 510 511 512 513 514 515 516 517 518 519 519 520 521 522 523 524 525 526 527 528 529 529 530 531 532 533 534 535 536 537 538 539 539 540 541 542 543 544 545 546 547 548 549 549 550 551 552 553 554 555 556 557 558 559 559 560 561 562 563 564 565 566 567 568 569 569 570 571 572 573 574 575 576 577 578 579 579 580 581 582 583 584 585 586 587 588 589 589 590 591 592 593 594 595 596 597 598 598 599 599 600 601 602 603 604 605 606 607 608 609 609 610 611 612 613 614 615 616 617 618 619 619 620 621 622 623 624 625 626 627 628 629 629 630 631 632 633 634 635 636 637 638 639 639 640 641 642 643 644 645 646 647 648 649 649 650 651 652 653 654 655 656 657 658 659 659 660 661 662 663 664 665 666 667 667 668 669 669 670 671 672 673 674 675 676 677 677 678 679 679 680 681 682 683 684 685 686 687 688 689 689 690 691 692 693 694 695 696 697 697 698 699 699 700 699 700 701 702 703 704 705 706 707 708 709 709 710 711 712 713 714 715 715 716 717 718 718 719 719 720 721 722 723 724 725 726 727 728 729 729 730 731 732 733 734 735 735 736 737 737 738 738 739 739 740 740 741 742 743 744 744 745 745 746 746 747 747 748 748 749 749 750 750 751 752 752 753 753 754 754 755 755 756 756 757 757 758 758 759 759 760 760 761 761 762 762 763 763 764 764 765 765 766 766 767 767 768 768 769 769 770 770 771 771 772 772 773 773 774 774 775 775 776 776 777 777 778 778 779 779 779 780 780 781 781 782 782 783 783 784 784 785 785 786 786 787 787 788 788 789 789 789 790 790 790 791 791 792 792 793 793 794 794 795 795 796 796 797 797 798 798 799 799 799 800 800 801 801 802 802 803 803 804 804 805 805 806 806 807 807 808 808 809 809 810 810 811 811 812 812 813 813 814 814 815 815 816 816 817 817 818 818 819 819 820 820 821 821 822 822 823 823 824 824 825 825 826 826 827 827 828 828 829 829 830 830 831 831 832 832 833 833 834 834 835 835 836 836 837 837 838 838 839 839 840 840 841 841 842 842 843 843 844 844 845 845 846 846 847 847 848 848 849 849 850 850 851 851 852 852 853 853 854 854 855 855 856 856 857 857 858 858 859 859 860 860 861 861 862 862 863 863 864 864 865 865 866 866 867 867 868 868 869 869 870 870 871 871 872 872 873 873 874 874 875 875 876 876 877 877 878 878 879 879 880 880 881 881 882 882 883 883 884 884 885 885 886 886 887 887 888 888 889 889 890 890 891 891 892 892 893 893 894 894 895 895 896 896 897 897 898 898 899 899 900 900 901 901 902 902 903 903 904 904 905 905 906 906 907 907 908 908 909 909 910 910 911 911 912 912 913 913 914 914 915 915 916 916 917 917 918 918 919 919 920 920 921 921 922 922 923 923 924 924 925 925 926 926 927 927 928 928 929 929 930 930 931 931 932 932 933 933 934 934 935 935 936 936 937 937 938 938 939 939 940 940 941 941 942 942 943 943 944 944 945 945 946 946 947 947 948 948 949 949 950 950 951 951 952 952 953 953 954 954 955 955 956 956 957 957 958 958 959 959 960 960 961 961 962 962 963 963 964 964 965 965 966 966 967 967 968 968 969 969 970 970 971 971 972 972 973 973 974 974 975 975 976 976 977 977 978 978 979 979 980 980 981 981 982 982 983 983 984 984 985 985 986 986 987 987 988 988 989 989 990 990 991 991 992 992 993 993 994 994 995 995 996 996 997 997 998 998 999 999 1000 1000 1001 1001 1002 1002 1003 1003 1004 1004 1005 1005 1006 1006 1007 1007 1008 1008 1009 1009 1010 1010 1011 1011 1012 1012 1013 1013 1014 1014 1015 1015 1016 1016 1017 1017 1018 1018 1019 1019 1020 1020 1021 1021 1022 1022 1023 1023 1024 1024 1025 1025 1026 1026 1027 1027 1028 1028 1029 1029 1030 1030 1031 1031 1032 1032 1033 1033 1034 1034 1035 1035 1036 1036 1037 1037 1038 1038 1039 1039 1040 1040 1041 1041 1042 1042 1043 1043 1044 1044 1045 1045 1046 1046 1047 1047 1048 1048 1049 1049 1050 1050 1051 1051 1052 1052 1053 1053 1054 1054 1055 1055 1056 1056 1057 1057 1058 1058 1059 1059 1060 1060 1061 1061 1062 1062 1063 1063 1064 1064 1065 1065 1066 1066 1067 1067 1068 1068 1069 1069 1070 1070 1071 1071 1072 1072 1073 1073 1074 1074 1075 1075 1076 1076 1077 1077 1078 1078 1079 1079 1080 1080 1081 1081 1082 1082 1083 1083 1084 1084 1085 1085 1086 1086 1087 1087 1088 1088 1089 1089 1090 1090 1091 1091 1092 1092 1093 1093 1094 1094 1095 1095 1096 1096 1097 1097 1098 1098 1099 1099 1099 1099 1100 1100 1101 1101 1102 1102 1103 1103 1104 1104 1105 1105 1106 1106 1107 1107 1108 1108 1109 1109 1110 1110 1111 1111 1112 1112 1113 1113 1114 1114 1115 1115 1116 1116 1117 1117 1118 1118 1119 1119 1120 1120 1121 1121 1122 1122 1123 1123 1124 1124 1125 1125 1126 1126 1127 1127 1128 1128 1129 1129 1130 1130 1131 1131 1132 1132 1133 1133 1134 1134 1135 1135 1136 1136 1137 1137 1138 1138 1139 1139 1140 1140 1141 1141 1142 1142 1143 1143 1144 1144 1145 1145 1146 1146 1147 1147 1148 1148 1149 1149 1150 1150 1151 1151 1152 1152 1153 1153 1154 1154 1155 1155 1156 1156 1157 1157 1158 1158 1159 1159 1160 1160 1161 1161 1162 1162 1163 1163 1164 1164 1165 1165 1166 1166 1167 1167 1168 1168 1169 1169 1170 1170 1171 1171 1172 1172 1173 1173 1174 1174 1175 1175 1176 1176 1177 1177 1178 1178 1179 1179 1180 1180 1181 1181 1182 1182 1183 1183 1184 1184 1185 1185 1186 1186 1187 1187 1188 1188 1189 1189 1190 1190 1191 1191 1192 1192 1193 1193 1194 1194 1195 1195 1196 1196 1197 1197 1198 1198 1199 1199 1199 1199 1200 1200 1201 1201 1202 1202 1203 1203 1204 1204 1205 1205 1206 1206 1207 1207 1208 1208 1209 1209 1210 1210 1211 1211 1212 1212 1213 1213 1214 1214 1215 1215 1216 1216 1217 1217 1218 1218 1219 1219 1220 1220 1221 1221 1222 1222 1223 1223 1224 1224 1225 1225 1226 1226 1227 1227 1228 1228 1229 1229 1230 1230 1231 1231 1232 1232 1233 1233 1234 1234 1235 1235 1236 1236 1237 1237 1238 1238 1239 1239 1240 1240 1241 1241 1242 1242 1243 1243 1244 1244 1245 1245 1246 1246 1247 1247 1248 1248 1249 1249 1250 1250 1251 1251 1252 1252 1253 1253 1254 1254 1255 1255 1256 1256 1257 1257 1258 1258 1259 1259 1260 1260 1261 1261 1262 1262 1263 1263 1264 1264 1265 1265 1266 1266 1267 1267 1268 1268 1269 1269 1270 1270 1271 1271 1272 1272 1273 1273 1274 1274 1275 1275 1276 1276 1277 1277 1278 1278 1279 1279 1280 1280 1281 1281 1282 1282 1283 1283 1284 1284 1285 1285 1286 1286 1287 1287 1288 1288 1289 1289 1290 1290 1291 1291 1292 1292 1293 1293 1294 1294 1295 1295 1296 1296 1297 1297 1298 1298 1299 1299 1299 1299 1300 1300 1301 1301 1302 1302 1303 1303 1304 1304 1305 1305 1306 1306 1307 1307 1308 1308 1309 1309 1310 1310 1311 1311 1312 1312 1313 1313 1314 1314 1315 1315 1316 1316 1317 1317 1318 1318 1319 1319 1320 1320 1321 1321 1322 1322 1323 1323 1324 1324 1325 1325 1326 1326 1327 1327 1328 1328 1329 1329 1330 1330 1331 1331 1332 1332 1333 1333 1334 1334 1335 1335 1336 1336 1337 1337 1338 1338 1339 1339 1340 1340 1341 1341 1342 1342 1343 1343 1344 1344 1345 1345 1346 1346 1347 1347 1348 1348 1349 1349 1350 1350 1351 1351 1352 1352 1353 1353 1354 1354 1355 1355 1356 1356 1357 1357 1358 1358 1359 1359 1360 1360 1361 1361 1362 1362 1363 1363 1364 1364 1365 1365 1366 1366 1367 1367 1368 1368 1369 1369 1370 1370 1371 1371 1372 1372 1373 1373 1374 1374 1375 1375 1376 1376 1377 1377 1378 1378 1379 1379 1380 1380 1381 1381 1382 1382 1383 1383 1384 1384 1385 1385 1386 1386 1387 1387 1388 1388 1389 1389 1390 1390 1391 1391 1392 1392 1393 1393 1394 1394 1395 1395 1396 1396 1397 1397 1398 1398 1399 1399 1399 1399 1400 1400 1401 1401 1402 1402 1403 1403 1404 1404 1405 1405 1406 1406 1407 1407 1408 1408 1409 1409 1410 1410 1411 1411 1412 1412 1413 1413 1414 1414 1415 1415 1416 1416 1417 1417 1418 1418 1419 1419 1420 1420 1421 1421 1422 1422 1423 1423 1424 1424 1425 1425 1426 1426 1427 1427 1428 1428 1429 1429 1430 1430 1431 1431 1432 1432 1433 1433 1434 1434 1435 1435 1436 1436 1437 1437 1438 1438 1439 1439 1440 1440 1441 1441 1442 1442 1443 1443 1444 1444 1445 1445 1446 1446 1447 1447 1448 1448 1449 1449 1450 1450 1451 1451 1452 1452 1453 1453 1454 1454 1455 1455 1456 1456 1457 1457 1458 1458 1459 1459 1460 1460 1461 1461 1462 1462 1463 1463 1464 1464 1465 1465 1466 1466 1467 1467 1468 1468 1469 1469 1470 1470 1471 1471 1472 1472 1473 1473 1474 1474 1475 1475 1476 1476 1477 1477 1478 1478 1479 1479 1480 1480 1481 1481 1482 1482 1483 1483 1484 1484 1485 1485 1486 1486 1487 1487 1488 1488 1489 1489 1490 1490 1491 1491 1492 1492 1493 1493 1494 1494 1495 1495 1496 1496 1497 1497 1498 1498 1499 1499 1499 1499 1500 1500 1501 1501 1502 1502 1503 1503 1504 1504 1505 1505 1506 
```

4.2.1.3 Evaluation

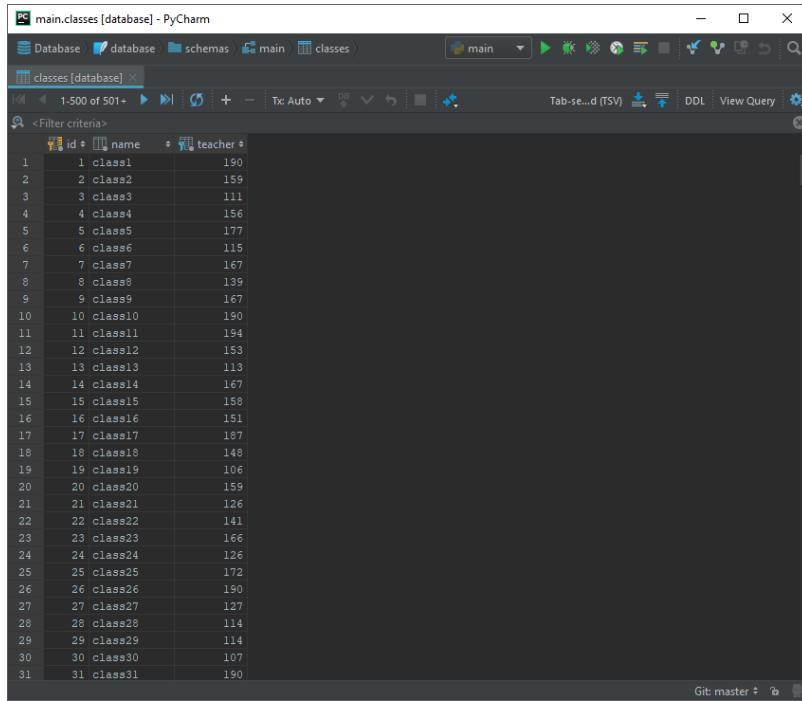
Table successfully filled with data of valid form and in accordance to data structure and design

4.2.2 Populate classes table

4.2.2.1 Code

```
def classes():
    # Creates 1000 unique valid entries in classes table with a randomly assigned
teacher
    identity = 1
    while identity <= 1000:
        name = 'class{}'.format(identity)
        teacher = random.randint(101, 200)
        sql = 'INSERT INTO classes(name, teacher) ' \
              'VALUES (?, ?);'
        c.execute(sql, (name, teacher))
        identity += 1
    conn.commit()
```

4.2.2.2 Result



The screenshot shows the PyCharm Database tool interface. The title bar says "main.classes [database] - PyCharm". The left sidebar shows "Database", "schemas", "main", and "classes". The main area is titled "classes [database]" and contains a table with 31 rows. The columns are "id" (1-31), "name" (e.g., class1, class2, ..., class31), and "teacher" (e.g., 190, 159, ..., 190). The table has a header row with column names.

	id	name	teacher
1	1	class1	190
2	2	class2	159
3	3	class3	111
4	4	class4	156
5	5	class5	177
6	6	class6	115
7	7	class7	167
8	8	class8	139
9	9	class9	167
10	10	class10	190
11	11	class11	194
12	12	class12	153
13	13	class13	113
14	14	class14	167
15	15	class15	158
16	16	class16	151
17	17	class17	187
18	18	class18	148
19	19	class19	106
20	20	class20	159
21	21	class21	126
22	22	class22	141
23	23	class23	166
24	24	class24	126
25	25	class25	172
26	26	class26	190
27	27	class27	127
28	28	class28	114
29	29	class29	114
30	30	class30	107
31	31	class31	190

4.2.2.3 Evaluation

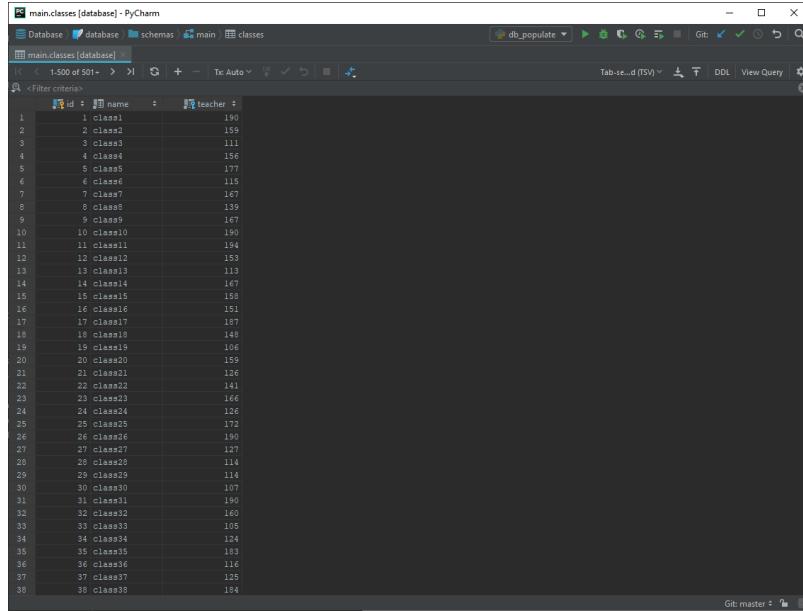
Table successfully filled with data of valid form and in accordance to data structure and design

4.2.3 Populate class_user table

4.2.3.1 Code

```
def class_user():
    # Creates 10000 unique valid entries in class_user table
    # with a randomly chosen student assigned to a randomly chosen class
    identity = 1
    while identity <= 10000:
        class_id = random.randint(1, 1000)
        student_id = random.randint(1, 100)
        sql = 'INSERT INTO class_user(class_id, student_id) ' \
              'VALUES (?, ?);'
        c.execute(sql, (class_id, student_id))
        identity += 1
    conn.commit()
```

4.2.3.2 Result



	id	name	teacher
1	1	class1	190
2	2	class2	159
3	3	class3	111
4	4	class4	156
5	5	class5	177
6	6	class6	115
7	7	class7	167
8	8	class8	139
9	9	class9	167
10	10	class10	190
11	11	class11	194
12	12	class12	153
13	13	class13	113
14	14	class14	167
15	15	class15	158
16	16	class16	151
17	17	class17	187
18	18	class18	148
19	19	class19	106
20	20	class20	159
21	21	class21	126
22	22	class22	141
23	23	class23	166
24	24	class24	126
25	25	class25	172
26	26	class26	190
27	27	class27	127
28	28	class28	114
29	29	class29	114
30	30	class30	107
31	31	class31	190
32	32	class32	160
33	33	class33	105
34	34	class34	124
35	35	class35	183
36	36	class36	116
37	37	class37	125
38	38	class38	184

4.2.3.3 Evaluation

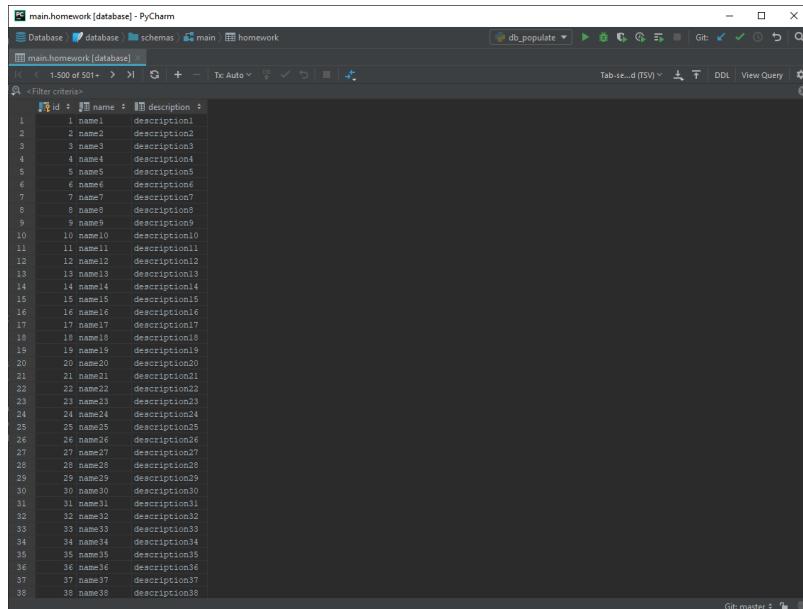
Table successfully filled with data of valid form and in accordance to data structure and design

4.2.4 Populate homework table

4.2.4.1 Code

```
def homework():
    # Creates 10000 unique valid entries in homework table
    identity = 1
    while identity <= 10000:
        name = 'name{}'.format(identity)
        description = 'description{}'.format(identity)
        sql = 'INSERT INTO homework(name, description)' \
              'VALUES (?, ?);'
        c.execute(sql, (name, description))
        identity += 1
    conn.commit()
```

4.2.4.2 Result



	id	name	description
1	1	name1	description1
2	2	name2	description2
3	3	name3	description3
4	4	name4	description4
5	5	name5	description5
6	6	name6	description6
7	7	name7	description7
8	8	name8	description8
9	9	name9	description9
10	10	name10	description10
11	11	name11	description11
12	12	name12	description12
13	13	name13	description13
14	14	name14	description14
15	15	name15	description15
16	16	name16	description16
17	17	name17	description17
18	18	name18	description18
19	19	name19	description19
20	20	name20	description20
21	21	name21	description21
22	22	name22	description22
23	23	name23	description23
24	24	name24	description24
25	25	name25	description25
26	26	name26	description26
27	27	name27	description27
28	28	name28	description28
29	29	name29	description29
30	30	name30	description30
31	31	name31	description31
32	32	name32	description32
33	33	name33	description33
34	34	name34	description34
35	35	name35	description35
36	36	name36	description36
37	37	name37	description37
38	38	name38	description38

4.2.4.3 Evaluation

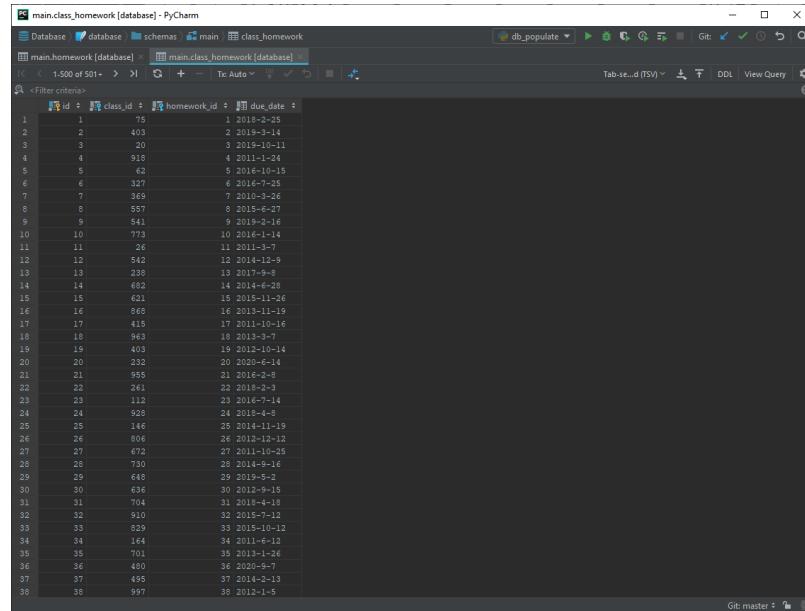
Table successfully filled with data of valid form and in accordance to data structure and design

4.2.5 Populate class_homework table

4.2.5.1 Code

```
def class_homework():
    # Creates 10000 unique valid entries in class_homework table
    # with each homework in the homework table assigned to a random task and with a
    random due date
    identity = 1
    while identity <= 10000:
        class_id = random.randint(1, 1000)
        homework_id = identity
        due_date = '{}-{}-{}'.format((random.randint(2010, 2020)), (random.rand-
int(1, 12)), (random.randint(1, 28)))
        sql = 'INSERT INTO class_homework(class_id, homework_id, due_date)' \
              'VALUES (?, ?, ?);'
        c.execute(sql, (class_id, homework_id, due_date))
        identity += 1
    conn.commit()
```

4.2.5.2 Result



The screenshot shows a PyCharm interface with a database tool window. The left pane lists databases: 'main.homework [database]' and 'main.class_homework [database]'. The right pane displays the 'main.class_homework' table with the following schema: id, class_id, homework_id, due_date. The table contains 501 rows of data, with the first few rows shown below:

		id	class_id	homework_id	due_date
1	1	1	75	1	2018-2-25
2	2	2	403	2	2019-3-14
3	3	3	20	3	2018-3-11
4	4	4	915	4	2001-3-24
5	5	5	62	5	2016-3-15
6	6	6	327	6	2016-3-25
7	7	7	369	7	2010-3-26
8	8	8	557	8	2015-4-27
9	9	9	541	9	2019-3-16
10	10	10	773	10	2016-1-14
11	11	11	26	11	2011-3-7
12	12	12	542	12	2014-12-9
13	13	13	238	13	2017-9-8
14	14	14	692	14	2011-3-28
15	15	15	621	15	2018-1-26
16	16	16	568	16	2013-3-19
17	17	17	415	17	2011-10-16
18	18	18	963	18	2013-3-7
19	19	19	403	19	2012-10-14
20	20	20	232	20	2020-6-14
21	21	21	955	21	2016-6-8
22	22	22	261	22	2018-3-3
23	23	23	112	23	2016-7-14
24	24	24	928	24	2018-4-8
25	25	25	148	25	2013-1-19
26	26	26	506	26	2012-2-24
27	27	27	672	27	2011-3-25
28	28	28	730	28	2014-3-16
29	29	29	648	29	2019-3-2
30	30	30	636	30	2012-3-15
31	31	31	704	31	2018-4-18
32	32	32	910	32	2015-3-12
33	33	33	829	33	2015-10-12
34	34	34	164	34	2011-6-12
35	35	35	701	35	2013-3-26
36	36	36	491	36	2020-9-7
37	37	37	495	37	2014-1-13
38	38	38	597	38	2012-1-5

4.2.5.3 Evaluation

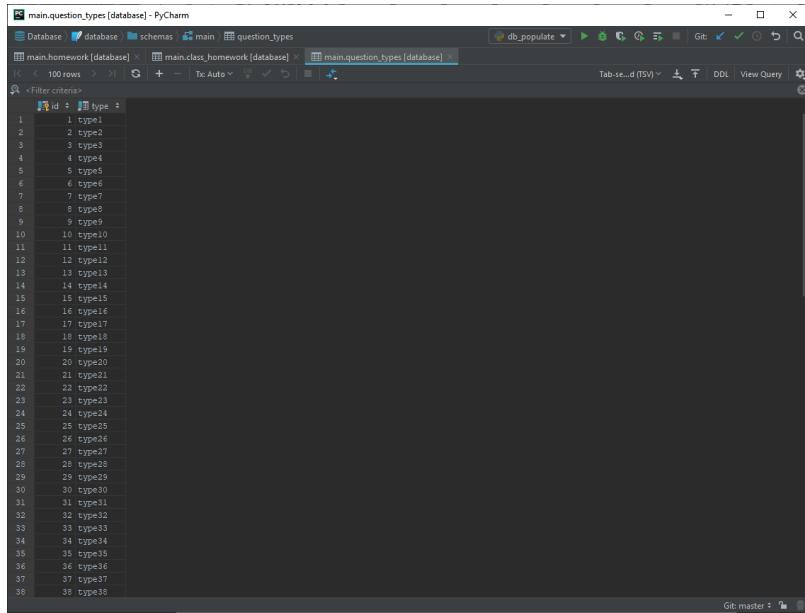
Table successfully filled with data of valid form and in accordance to data structure and design

4.2.6 Populate question_types table

4.2.6.1 Code

```
def question_types():
    # Creates 100 unique valid entries in question_types table
    identity = 1
    while identity <= 100:
        question_type = 'type{}'.format(identity)
        sql = 'INSERT INTO question_types(type)' \
              'VALUES (?)'
        c.execute(sql, (question_type,))
        identity += 1
    conn.commit()
```

4.2.6.2 Result



The screenshot shows a PyCharm interface with a database tool window open. The current database is 'main.question_types'. The table has two columns: 'id' and 'type'. The data consists of 38 rows, each containing a unique ID from 1 to 38 and a corresponding type name. The types are: type1, type2, type3, type4, type5, type6, type7, type8, type9, type10, type11, type12, type13, type14, type15, type16, type17, type18, type19, type20, type21, type22, type23, type24, type25, type26, type27, type28, type29, type30, type31, type32, type33, type34, type35, type36, type37, and type38.

id	type
1	type1
2	type2
3	type3
4	type4
5	type5
6	type6
7	type7
8	type8
9	type9
10	type10
11	type11
12	type12
13	type13
14	type14
15	type15
16	type16
17	type17
18	type18
19	type19
20	type20
21	type21
22	type22
23	type23
24	type24
25	type25
26	type26
27	type27
28	type28
29	type29
30	type30
31	type31
32	type32
33	type33
34	type34
35	type35
36	type36
37	type37
38	type38

4.2.6.3 Evaluation

Table successfully filled with data of valid form and in accordance to data structure and design

4.2.7 Populate questions table

4.2.7.1 Code

```
def questions():
    # Creates 100000 unique valid entries in homework table
    identity = 1
    while identity <= 100000:
        name = 'question{}'.format(identity)
        type_id = random.randint(1, 100)
        question_text = 'question text {}'.format(identity)
        correct_answer = 'correct answer {}'.format(identity)
        answer_b = 'b {}'.format(identity)
        answer_c = 'c {}'.format(identity)
        answer_d = 'd {}'.format(identity)
        sql = 'INSERT INTO questions(name, type_id, question_text, correct_answer,
answer_b, answer_c, answer_d) \
            VALUES(?, ?, ?, ?, ?, ?, ?);'
        c.execute(sql, (name, type_id, question_text, correct_answer, answer_b, an-
swer_c, answer_d))
        identity += 1
    conn.commit()
```

4.2.7.2 Result

	id	name	type_id	question_text	correct_answer	answer_b	answer_c	answer_d
1	question1	39 question text 1	1	correct answer 1 b 1	c 1	d 1		
2	question2	85 question text 2	2	correct answer 2 b 2	c 2	d 2		
3	question3	51 question text 3	3	correct answer 3 b 3	c 3	d 3		
4	question4	75 question text 4	4	correct answer 4 b 4	c 4	d 4		
5	question5	20 question text 5	5	correct answer 5 b 5	c 5	d 5		
6	question6	60 question text 6	6	correct answer 6 b 6	c 6	d 6		
7	question7	14 question text 7	7	correct answer 7 b 7	c 7	d 7		
8	question8	68 question text 8	8	correct answer 8 b 8	c 8	d 8		
9	question9	5 question text 9	9	correct answer 9 b 9	c 9	d 9		
10	question10	84 question text 10	10	correct answer 10 b 10	c 10	d 10		
11	question11	84 question text 11	11	correct answer 11 b 11	c 11	d 11		
12	question12	45 question text 12	12	correct answer 12 b 12	c 12	d 12		
13	question13	7 question text 13	13	correct answer 13 b 13	c 13	d 13		
14	question14	95 question text 14	14	correct answer 14 b 14	c 14	d 14		
15	question15	60 question text 15	15	correct answer 15 b 15	c 15	d 15		
16	question16	9 question text 16	16	correct answer 16 b 16	c 16	d 16		
17	question17	26 question text 17	17	correct answer 17 b 17	c 17	d 17		
18	question18	14 question text 18	18	correct answer 18 b 18	c 18	d 18		
19	question19	45 question text 19	19	correct answer 19 b 19	c 19	d 19		
20	question20	2 question text 20	20	correct answer 20 b 20	c 20	d 20		
21	question21	90 question text 21	21	correct answer 21 b 21	c 21	d 21		
22	question22	60 question text 22	22	correct answer 22 b 22	c 22	d 22		
23	question23	57 question text 23	23	correct answer 23 b 23	c 23	d 23		
24	question24	45 question text 24	24	correct answer 24 b 24	c 24	d 24		
25	question25	61 question text 25	25	correct answer 25 b 25	c 25	d 25		
26	question26	17 question text 26	26	correct answer 26 b 26	c 26	d 26		
27	question27	73 question text 27	27	correct answer 27 b 27	c 27	d 27		
28	question28	94 question text 28	28	correct answer 28 b 28	c 28	d 28		
29	question29	84 question text 29	29	correct answer 29 b 29	c 29	d 29		
30	question30	87 question text 30	30	correct answer 30 b 30	c 30	d 30		
31	question31	88 question text 31	31	correct answer 31 b 31	c 31	d 31		
32	question32	49 question text 32	32	correct answer 32 b 32	c 32	d 32		
33	question33	4 question text 33	33	correct answer 33 b 33	c 33	d 33		
34	question34	5 question text 34	34	correct answer 34 b 34	c 34	d 34		
35	question35	96 question text 35	35	correct answer 35 b 35	c 35	d 35		
36	question36	8 question text 36	36	correct answer 36 b 36	c 36	d 36		
37	question37	42 question text 37	37	correct answer 37 b 37	c 37	d 37		
38	question38	96 question text 38	38	correct answer 38 b 38	c 38	d 38		

4.2.7.3 Evaluation

Table successfully filled with data of valid form and in accordance to data structure and design

4.2.8 Populate homework_questions table

4.2.8.1 Code

```
def homework_questions():
    # Creates 100000 unique valid entries in homework_questions table
    # with each question in the question table being assigned to a random class
    identity = 1
    while identity <= 100000:
        homework_id = random.randint(1, 10000)
        question_id = identity
        sql = 'INSERT INTO homework_questions(homework_id, question_id) ' \
              'VALUES(?, ?);'
        c.execute(sql, (homework_id, question_id))
        identity += 1
    conn.commit()
```

4.2.8.2 Result

	homework_id	question_id
1	7626	1
2	4648	2
3	2706	3
4	7305	4
5	6097	5
6	316	6
7	6357	7
8	5021	8
9	2851	9
10	8660	10
11	85	11
12	172	12
13	6616	13
14	8908	14
15	123	15
16	2433	16
17	5061	17
18	7053	18
19	6324	19
20	7815	20
21	5261	21
22	7046	22
23	1814	23
24	676	24
25	3077	25
26	5523	26
27	641	27
28	3428	28
29	9620	29
30	1316	30
31	1732	31
32	5833	32
33	4027	33
34	2452	34
35	6702	35
36	4496	36
37	9102	37
38	1211	38

4.2.8.3 Evaluation

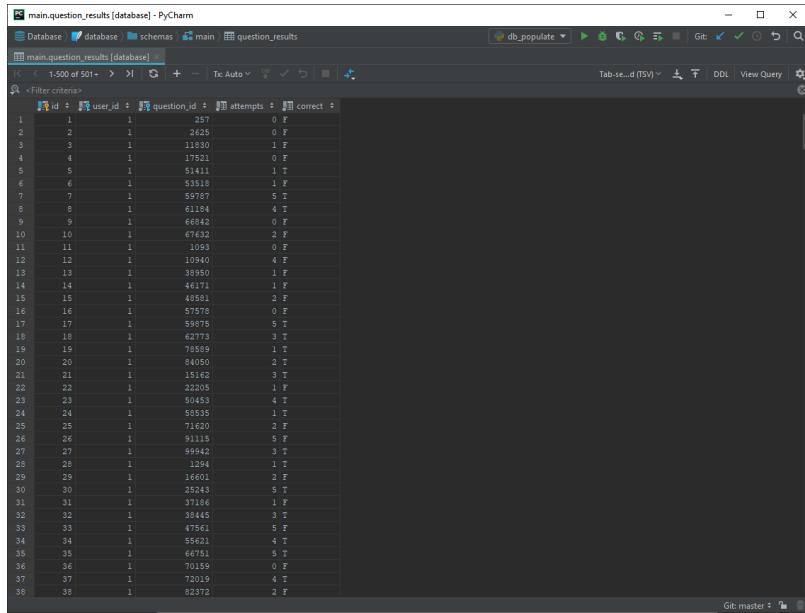
Table successfully filled with data of valid form and in accordance to data structure and design

4.2.9 Populate question_results table

4.2.9.1 Code

```
def question_results():
    # Creates 100 unique valid entries in question_results table
    # with each question applicable to each user given a random attempt count (0 to
5)
    # and whether it has been successfully completed randomly picked from True or
False
    # (If attempts is 0 False is always picked as question cannot be completed)
    identity = 1
    while identity <= 100:
        user_id = identity
        sql = 'SELECT homework_questions.question_id ' \
              'FROM (((homework_questions INNER JOIN homework ON homework.id =
homework_questions.homework_id) ' \
              'INNER JOIN class_homework ON class_homework.homework_id = home-
work.id) ' \
              'INNER JOIN classes ON class_homework.class_id = classes.id) ' \
              'INNER JOIN class_user ON class_user.class_id = classes.id) ' \
              'INNER JOIN users ON users.id = class_user.student_id ' \
              'WHERE users.id = ?'
        c.execute(sql, (user_id,))
        question_ids = c.fetchall()
        for each_question in question_ids:
            attempts = random.randint(0, 5)
            if attempts == 0:
                correct = 'F'
            else:
                correct = random.choice(['T', 'F'])
            sql = 'INSERT INTO question_results(user_id, question_id, attempts,
correct) ' \
                  'VALUES (?, ?, ?, ?)'
            c.execute(sql, (user_id, each_question[0], attempts, correct))
            identity += 1
        conn.commit()
```

4.2.9.2 Result



	id	user_id	question_id	attempts	correct
1	1	1	257	0	F
2	2	1	2625	0	F
3	3	1	11830	1	F
4	4	1	17521	0	F
5	5	1	51411	1	T
6	6	1	53516	1	F
7	7	1	55707	5	T
8	8	1	61184	1	F
9	9	1	66842	0	F
10	10	1	67632	2	F
11	11	1	1093	0	F
12	12	1	10940	4	F
13	13	1	39590	1	F
14	14	1	46171	1	F
15	15	1	48581	2	F
16	16	1	57578	0	F
17	17	1	59875	5	T
18	18	1	62773	3	F
19	19	1	70889	1	F
20	20	1	84050	2	T
21	21	1	15162	3	T
22	22	1	22205	1	F
23	23	1	50453	4	T
24	24	1	58535	1	F
25	25	1	71620	2	F
26	26	1	91115	5	F
27	27	1	99942	3	T
28	28	1	1284	1	T
29	29	1	16601	2	F
30	30	1	25243	1	F
31	31	1	37186	1	F
32	32	1	38445	3	T
33	33	1	47561	5	F
34	34	1	55621	4	T
35	35	1	66751	5	T
36	36	1	70159	0	F
37	37	1	72019	4	T
38	38	1	82372	2	F

4.2.9.3 Evaluation

Table successfully filled with data of valid form and in accordance to data structure and design

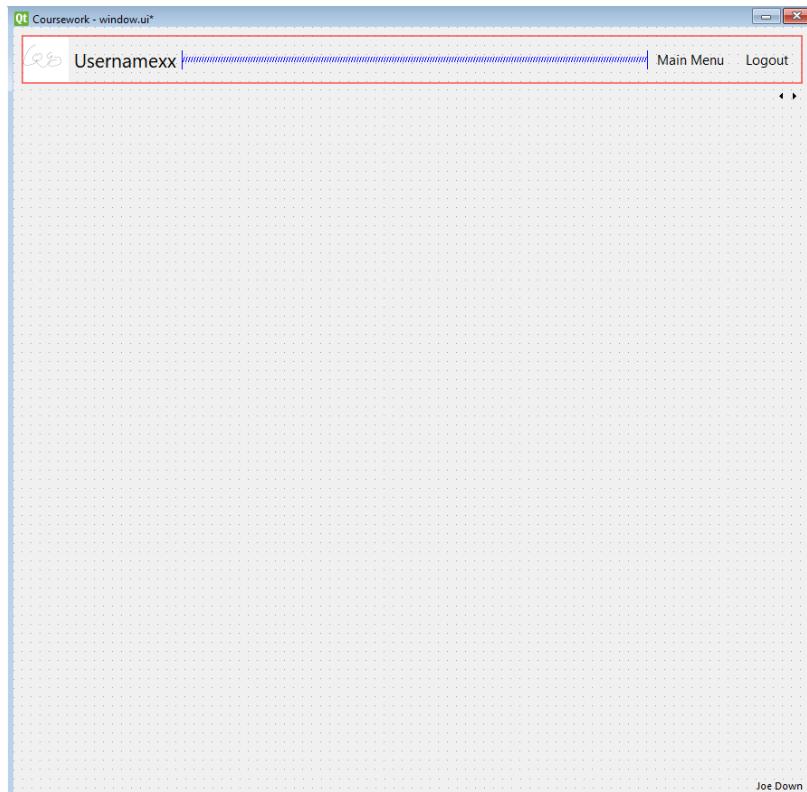
4.3 UI IMPLEMENTATION (WINDOW)

The following UI designs were created using QtDesigner as a .ui file and .qrc resource file, which were then converted to PyQt compatible .py files using the code below to allow them to be used as a library in the rest of the code

```
import os

os.popen('pyuic5 window.ui -o window.py')
os.popen('pyrcc5 window.qrc -o window_rc.py')
```

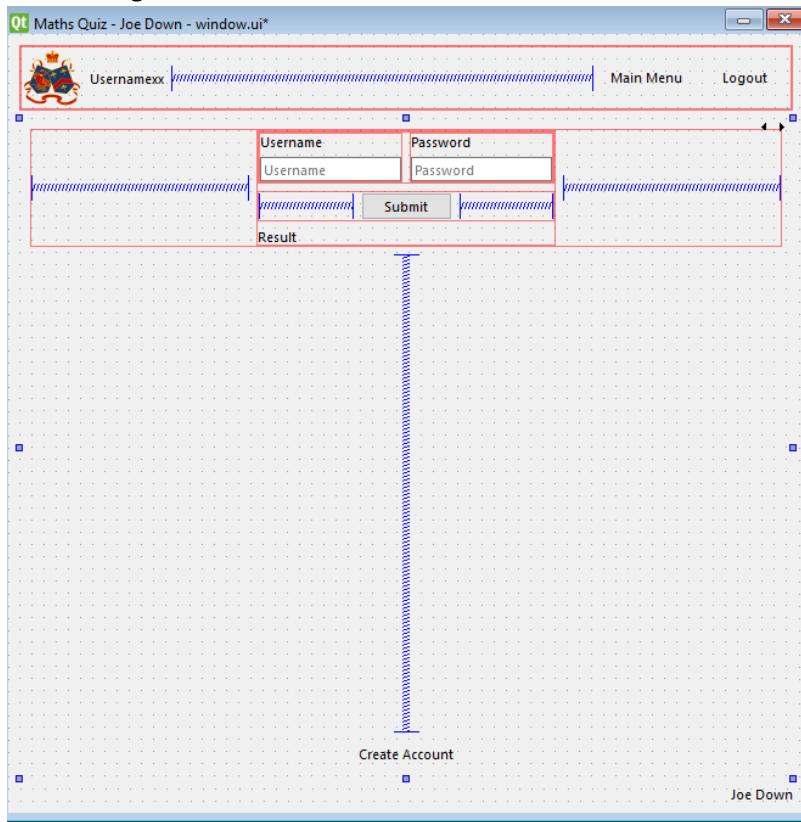
4.3.1 Base UI



Successfully provides navigation buttons to go to main menu or logout (these can be hidden or shown as needed during running of program)

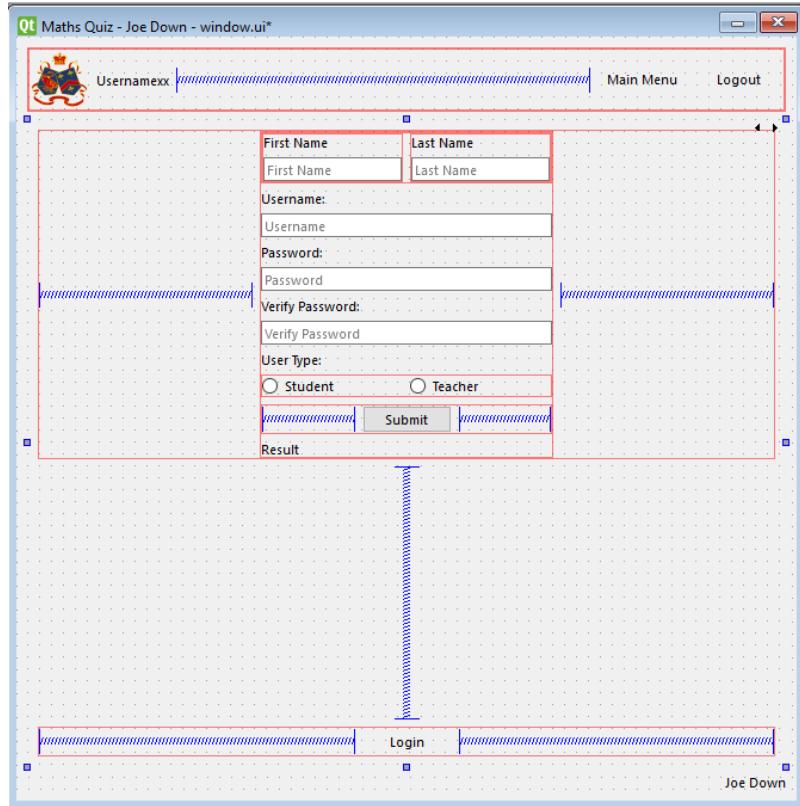
All points in design specification regarding logout buttons and main menu buttons can be disregarded as they have instead been implemented program wide

4.3.2 Login screen



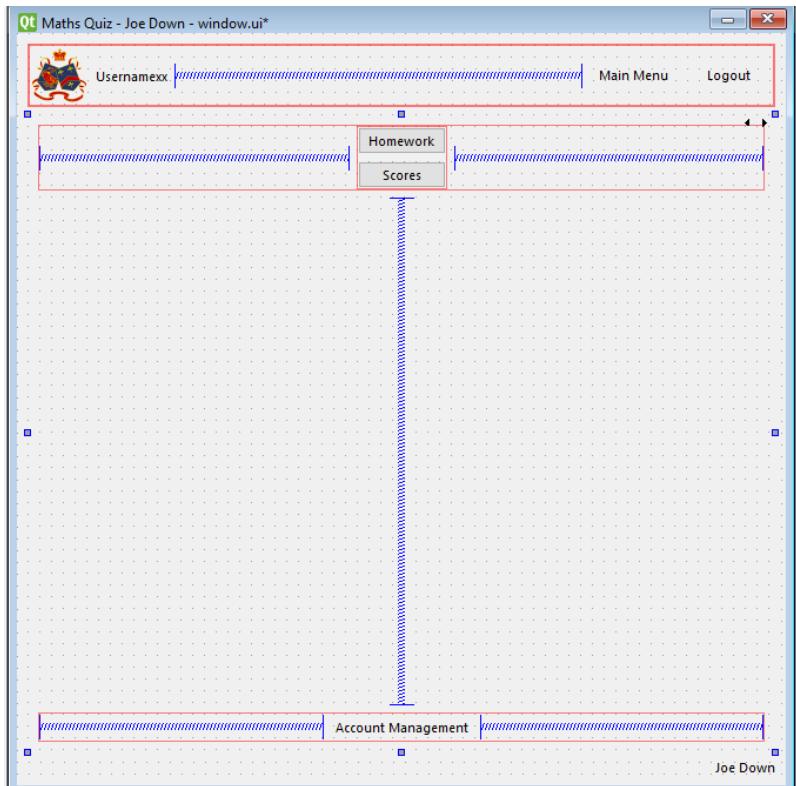
- Input box for username successfully provided (login_username_input)
- Input box for password successfully provided (login_password_input)
- Submit button successfully provided (login_submit_button)

4.3.3 Create account screen



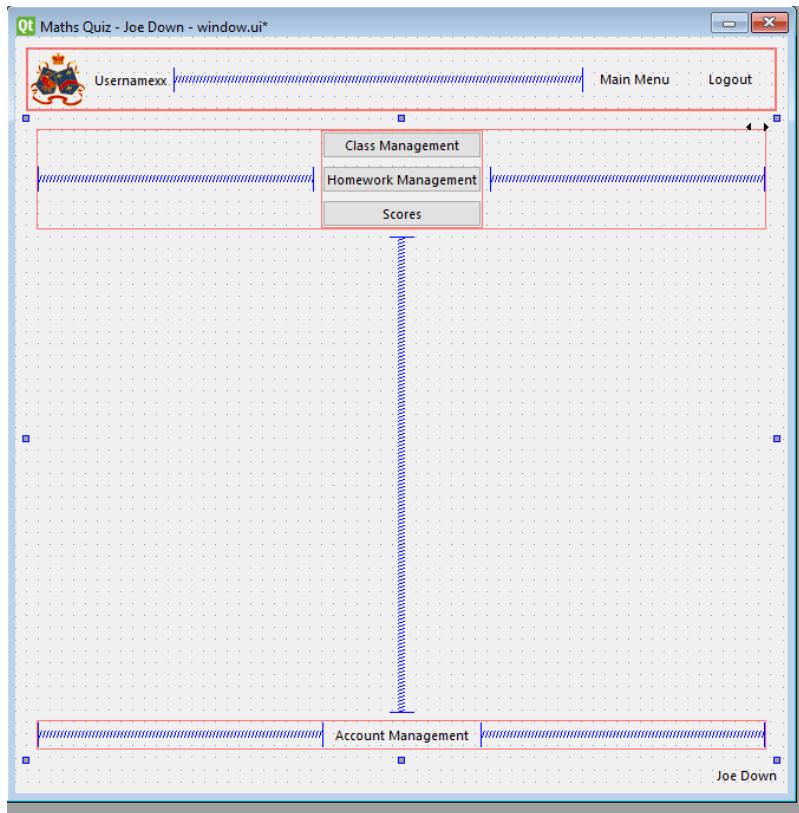
- Input box for first name successfully provided (create_account_first_name_input)
- Input box for surname successfully provided (create_account_last_name_input)
- Input box for username successfully provided (create_account_username_input)
- Input box for password successfully provided (create_account_password_input)
- Input box for password verification successfully provided (create_account_password_verify_input)
- Radio buttons to select account type (student or teacher) successfully provided (create_account_radio_student, create_account_radio_teacher)
- Submit button successfully provided (create_account_submit_button)
- Go to login page button successfully provided (create_account_login_button)
- Result output box added from design to improve user interaction (create_account_success_output)

4.3.4 Student main menu screen



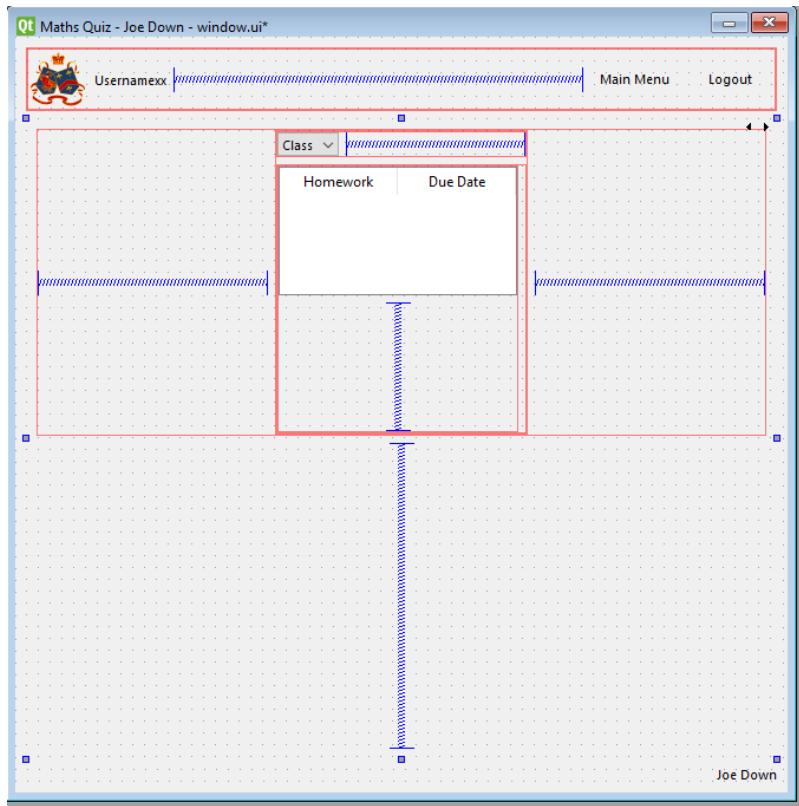
- Go to homework page button successfully provided
(student_main_menu_homework_button)
- Go to view previous scores page button successfully provided
(student_main_menu_previous_scores_button)
- Go to account management page button successfully provided
(student_main_menu_account_management_button)

4.3.5 Teacher main menu screen



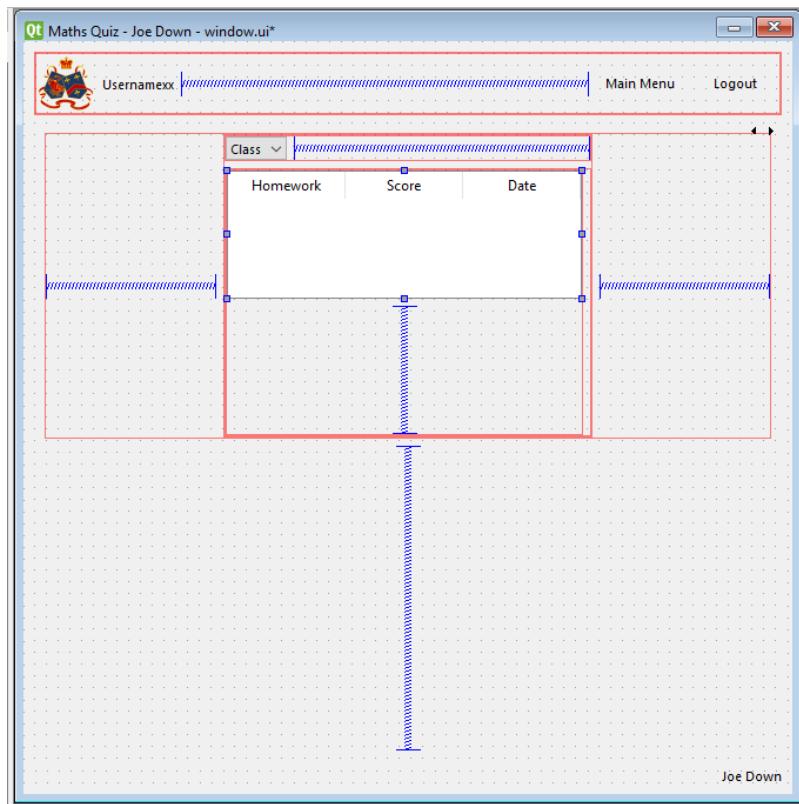
- Go to class admin page button successfully provided (teacher_main_menu_admin_button)
- Go to set homework page button successfully provided (teacher_main_menu_set_homework_button)
- Go to view classes page button successfully provided (teacher_main_menu_view_classes_button)
- Go to account management page button successfully provided (teacher_main_menu_account_management_button)

4.3.6 Homework select screen



- Select class dropdown button successfully provided (homework_select_class_combo_box)
- List of homework, each linking to relevant quiz provided as a table that can be filled during the running of the program as required (homework_select_table)

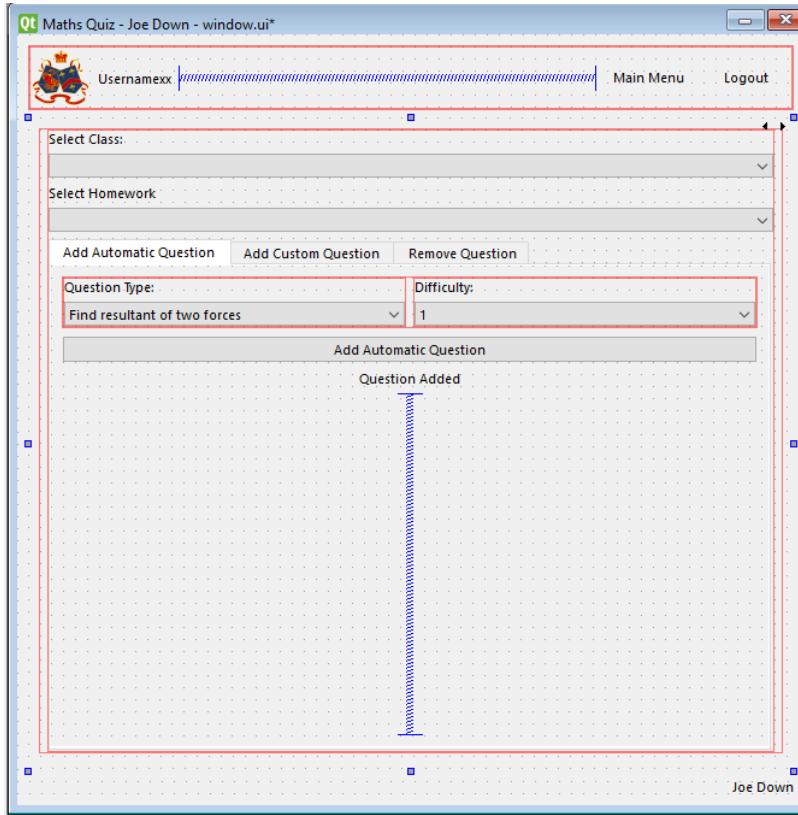
4.3.7 Previous scores screen



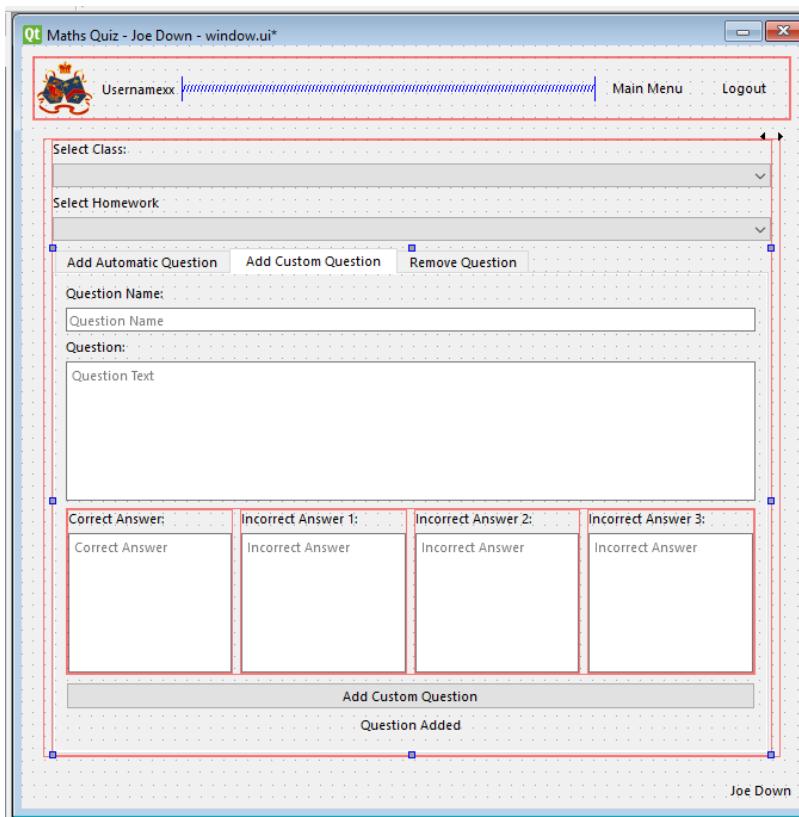
- Dropdown menus for each class student is replaced with dropdown menu to select a class (previous_scores_class_combo_box)
 - List of homework for each class with score and due date provided as a table (previous_scores_table)

4.3.8 Set homework screen

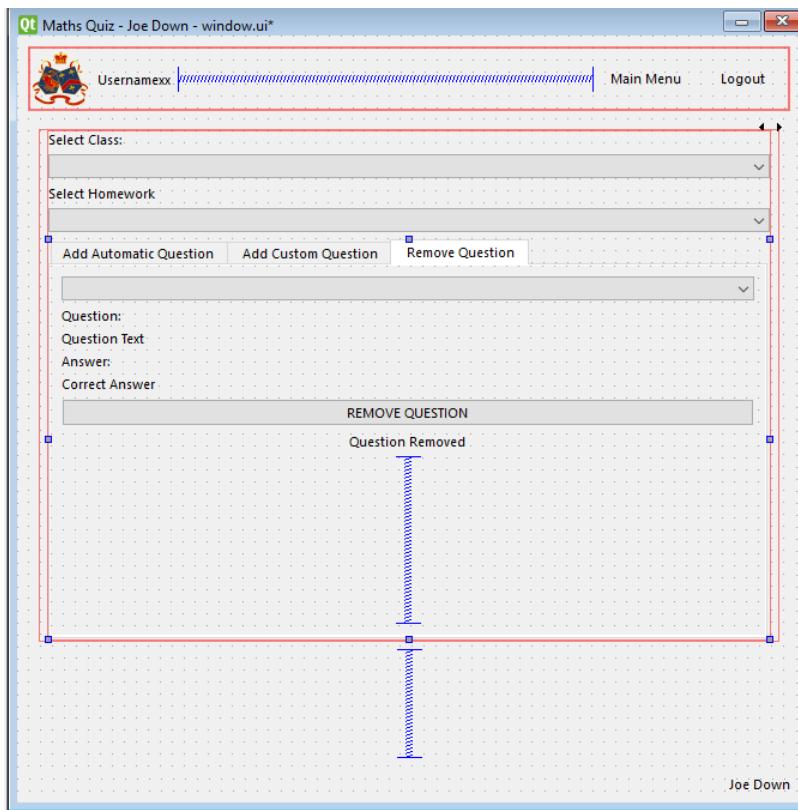
Design of page has been altered to separate each of the main functions of the page into their own separate selectable tab. Otherwise functionality remains unchanged from design.



- Dropdown box to select class successfully provided (set_homework_class_combo_box)
- Dropdown box to select homework successfully provided (set_homework_homework_combo_box)
- Automatic question
 - Dropdown to select topic successfully provided (set_homework_type_combo_box)
 - Dropdown to select difficulty 1-5 successfully provided (set_homework_difficulty_combo_box)
 - Add question button successfully provided (set_homework_add_automatic_question_button)
 - Add question status output added from design to increase usability (set_homework_auto_question_added_output)



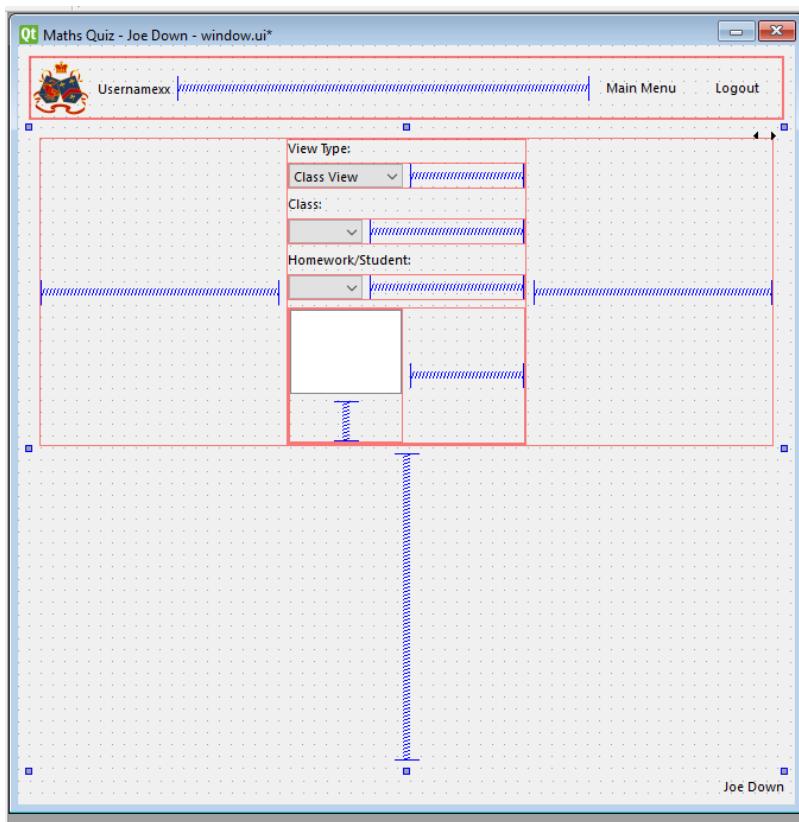
- Custom question
 - Question name input successfully provided (set_homework_question_name_input)
 - Question text input successfully provided (set_homework_question_input)
 - Correct answer input successfully provided (set_homework_correct_answer_input)
 - Incorrect answer 1 input successfully provided (set_homework_answer_b_input)
 - Incorrect answer 2 input successfully provided (set_homework_answer_c_input)
 - Incorrect answer 3 input successfully provided (set_homework_answer_d_input)
 - Add question button successfully provided (set_homework_add_custom_question_button)
 - Add question status output added from design to increase usability (set_homework_custom_question_added_output)



- Remove question
 - Select question dropdown successfully provided (set_homework_question_combo_box)
 - Selected question text output successfully provided (set_homework_question_label)
 - Selected question answer output successfully provided (set_homework_answer_label)
 - Remove question button successfully provided (set_homework_remove_question_button)
 - Remove question status output added from design to increase usability (set_homework_removed_output)

4.3.9 View classes screen

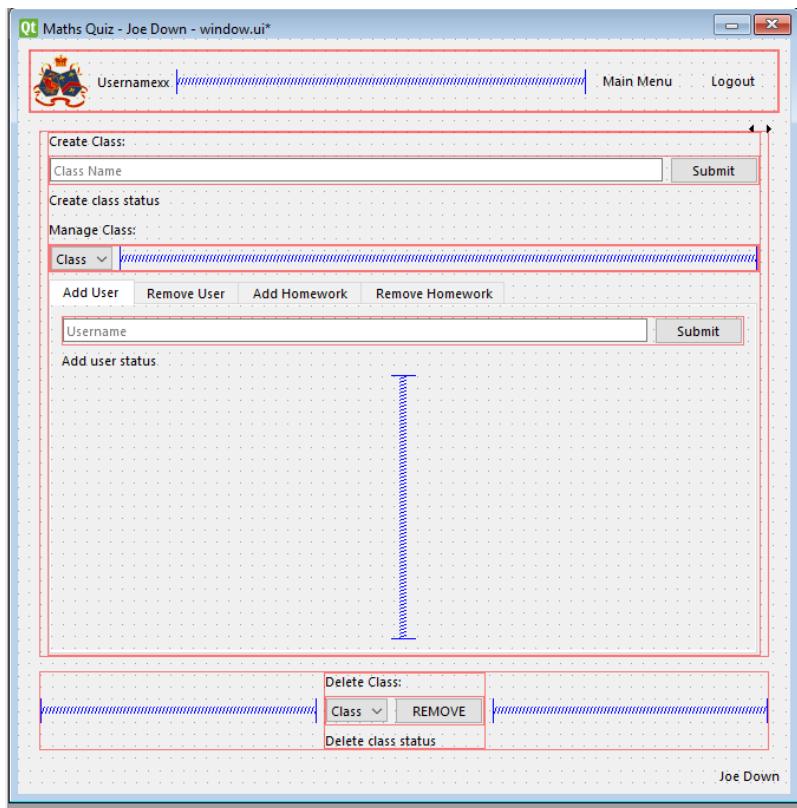
Functionality of page has been updated from design to also allow a teacher to instead view a table for all homework done by a student in their class to allow for more detailed tracking of student progress



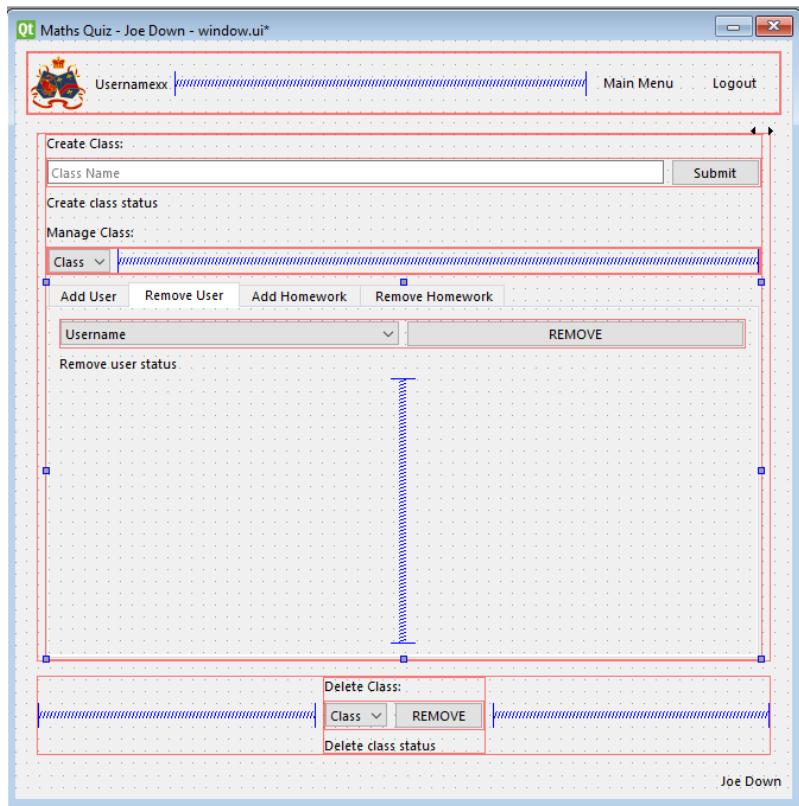
- View type dropdown added since design to allow a teacher to select to view all homework of a student as well as all students of a homework (view_classes_view_type_combo_box)
- Class select dropdown menu successfully provided (view_classes_class_combo_box)
- Dropdown menus for each homework in class replaced with a single dropdown to either allow the selection of a student in the class or homework class depending on status of view type (view_classes_homework_or_student_combo_box)
 - Table to view data will be dynamically added depending on functionality selected in future code (view_classes_score_table)

4.3.10 Admin screen

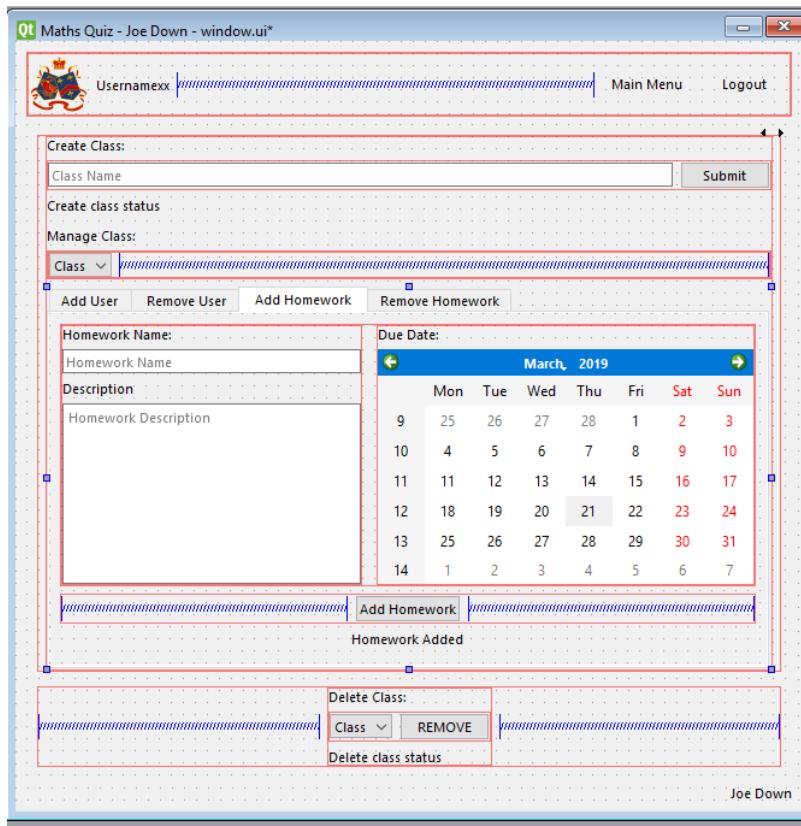
Design of page has been altered to separate each of the main functions of the page into their own separate selectable tab. Otherwise functionality remains unchanged from design except for a single class select button being used across all functions rather than being duplicated as in design.



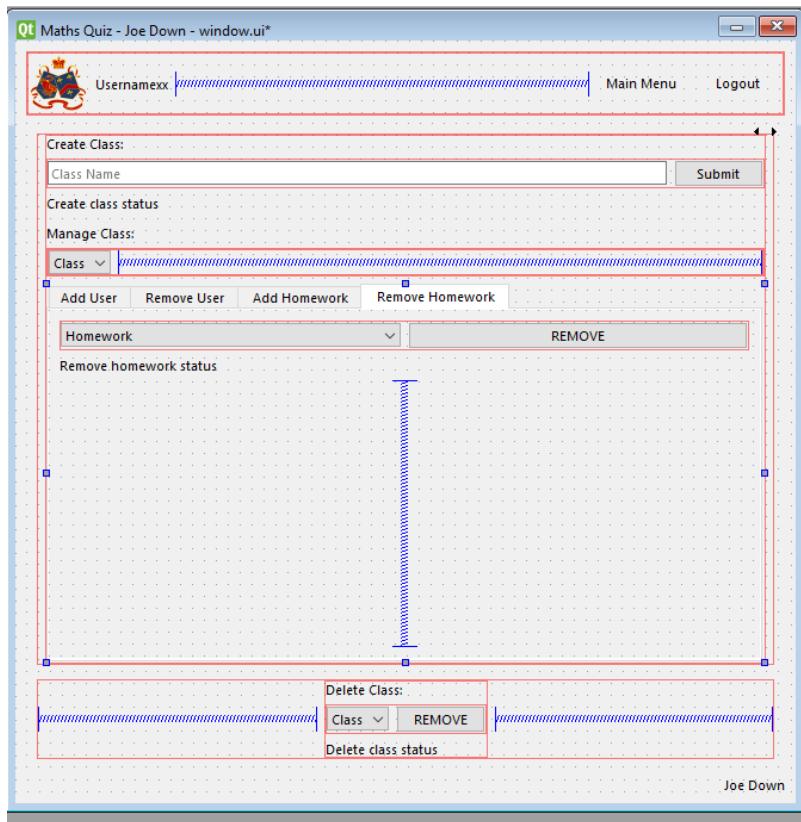
- Create class
 - Input box for class name successfully provided (admin_class_input)
 - Submit button successfully provided (admin_create_class_submit_button)
 - Create class status output added since design to increase usability (admin_create_class_status_label)
- Class select dropdown successfully provided (admin_class_user_combo_box)
- Add user to class
 - Input box for username successfully provided (admin_username_input)
 - Submit button successfully provided (admin_username_submit_button)
 - Add user status output added since design to increase usability (admin_add_user_status_label)
- Delete class
 - Class select dropdown successfully provided (admin_delete_class_combo_box)
 - Submit button successfully provided (admin_remove_class_button)
 - Delete class status output added since design to increase usability (admin_delete_class_status_label)



- Remove user from class
 - User select dropdown successfully provided (admin_username_combo_box)
 - Submit button successfully provided (admin_remove_user_button)
 - Remove user status output added since design to increase usability (admin_remove_user_status_label)

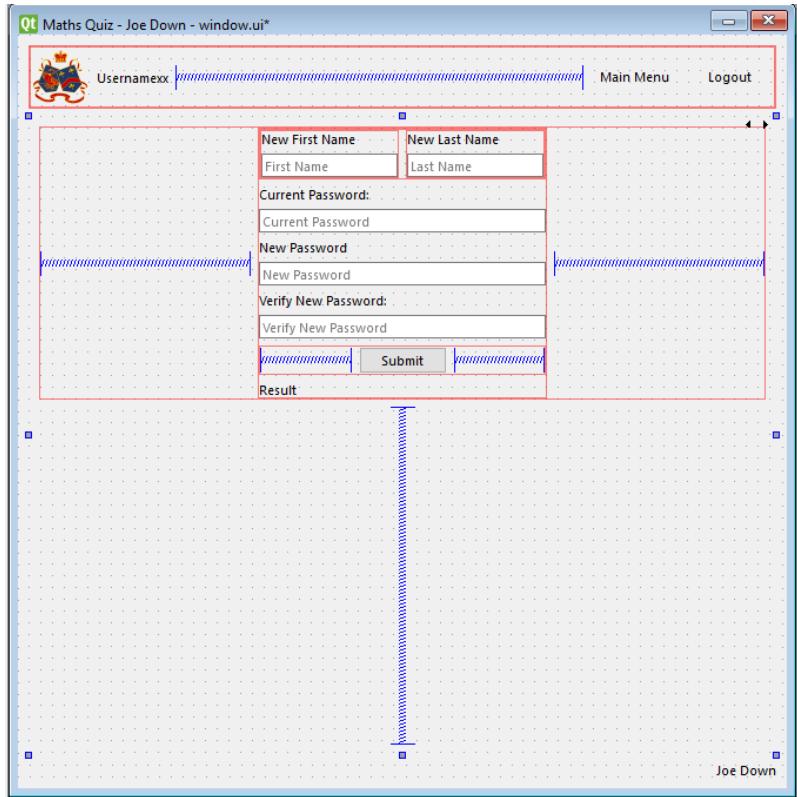


- Create homework
 - Input box for homework name successfully provided (admin_homework_name_input)
 - Input box for homework description added since design as it was missing and is required to satisfy the data structure of the question table (admin_homework_description_input)
 - Due date select calendar successfully provided (admin_due_date_calendar)
 - Submit button successfully provided (admin_add_homework_button)
 - Add homework status output added since design to increase usability (admin_add_homework_status_output)



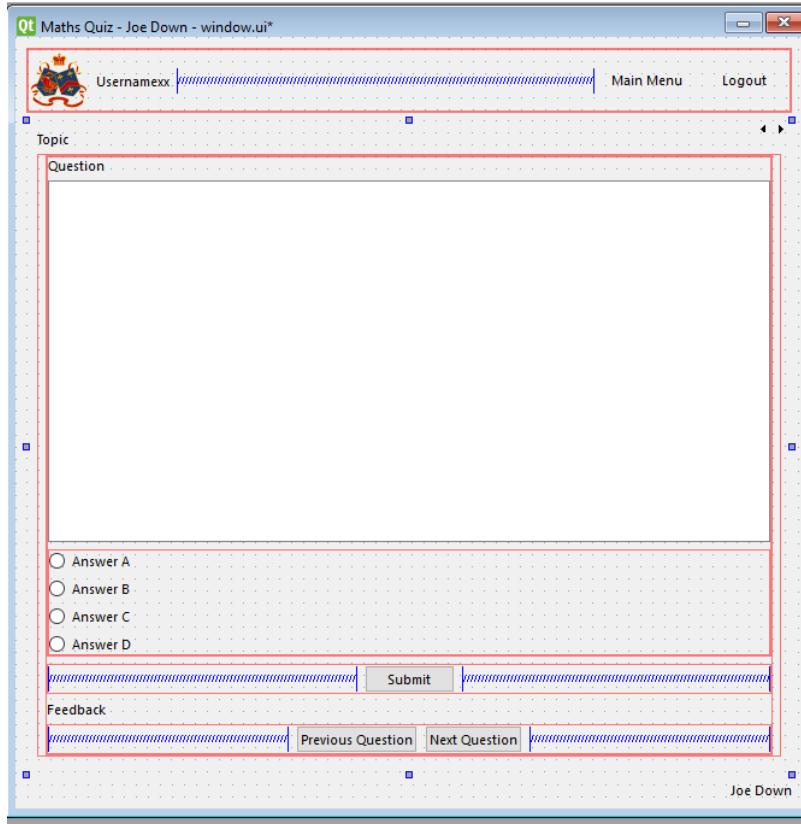
- Delete homework
 - Homework select dropdown successfully provided (admin_remove_homework_combo_box)
 - Submit button successfully provided (admin_remove_homework_button)
 - Remove homework status output added since design to increase usability (admin_remove_homework_status_label)

4.3.11 Account management screen



- Input box for new first name successfully provided (account_management_first_name_input)
- Input box for new surname successfully provided (account_management_last_name_input)
- Input box for password (to verify user making changes is the correct user) successfully provided (account_management_old_password_input)
- Input box for new password successfully provided (account_management_new_password_input)
- Input box for password verification successfully provided (account_management_new_password_verify_input)
- Submit button successfully provided (account_management_submit_button)
- Edit user status output added since design to increase usability (account_management_success_output)

4.3.12 Template quiz screen



- Previous question button successfully provided (question_previous_question_button)
- Next question button successfully provided (question_next_question_button)
- Question text area successfully provided (question_question_output)
- Graph widget added since design to allow for questions to be graphed if appropriate (widget can be hidden when graph is not provided) (question_chart)
- Radio buttons to select response (answer a, answer b, answer c, answer d) successfully provided (question_radio_a, question_radio_b, question_radio_c, question_radio_d)
- Submit answer button successfully provided (question_submit_button)
- Question feedback output added since design to increase usability (question_feedback_output)

4.4 LOGIN SCRIPTS (SCRIPTS.DB_SCRIPTS)

4.4.1 Import required Python libraries to perform hashing algorithms and communicate with an SQLite database

4.4.1.1 Final Code

```
import hashlib
import sqlite3
```

When generating salts later in development the following import was added:

```
import uuid
```

4.4.2 Connect to database

4.4.2.1 Final Code

```
# Connects to database
conn = sqlite3.connect('database.db')
c = conn.cursor()
```

Generic code used

4.4.3 Function to identify if the value passed matches a username stored in the database (True or False should be returned)

4.4.3.1 Issues during development

Initially the below code was written:

```
def check_user_exists(username):
    # Finds the number of occurrences in the users table of the inputted username.
    If 0 returned, username is not in DB
    sql = "SELECT COUNT(1) FROM users WHERE username = ?;"
    c.execute(sql, (username,))
    if c.fetchall() > 0:
        return True
    else:
        return False
```

This produce the following error:

Enter username: test

Traceback (most recent call last):

```
File "C:/Users/Joe/PycharmProjects/Coursework/scripts/db_scripts.py", line 146, in <module>
    user_exists = check_user_exists(un)

File "C:/Users/Joe/PycharmProjects/Coursework/scripts/db_scripts.py", line 15, in
check_user_exists

    if c.fetchall() > 0:

TypeError: '>' not supported between instances of 'list' and 'int'
```

Process finished with exit code 1

This turned out to be due to the fact that an SQL query will always return an array, even if there is only a single value, of the form [(data,)], meaning to reference the single piece of data it must be referred to using c.fetchall()[0][0]

4.4.3.2 Final Code

```
def check_user_exists(username: str) -> bool:
    # Finds the number of occurrences in the users table of the inputted username.
    If 0 returned, username is not in DB
    sql: str = 'SELECT COUNT(1) FROM users WHERE username = ?;'
    c.execute(sql, (username,))
    if c.fetchall()[0][0] > 0:
        return True
    else:
        return False
```

4.4.3.3 Testing

4.4.3.3.1 SQL

```
1   SELECT COUNT(1) FROM users WHERE username = 'test';

1 COUNT(1)
1 1

1 rows returned in 0ms from: SELECT COUNT(1) FROM users WHERE username
= 'test';
```

4.4.3.3.2 Test code

```
if __name__ == '__main__':
    un = input("Enter username: ")
    user_exists = check_user_exists(un)
    print("User exists: {}".format(check_user_exists(un)))
```

4.4.3.3.3 With valid data:

Enter username: test

User exists: True

Process finished with exit code 0

4.4.3.3.4 With Invalid data:

Enter username: qwerty

User exists: False

Process finished with exit code 0

4.4.3.3.5 Evaluation

Function returns correctly either True or False depending on whether a username is present in the database

4.4.4 Function to find the salt for a given username

4.4.4.1 Final Code

```
# Assumes valid username provided
def find_salt(user_id: int) -> str:
    # Finds the salt for the given username
    sql: str = 'SELECT password_salt FROM users WHERE id = ?;'
    c.execute(sql, (user_id,))
    salt: str = c.fetchall()[0][0]
    # Takes the SQL output and formats appropriately to give just a string
    return salt
```

4.4.4.2 Testing

4.4.4.2.1 SQL

The screenshot shows a terminal window with the following content:

```
1 SELECT password_salt FROM users WHERE username = 'test';

password_salt
1 9gu6g8dy9amkUZIzgAC4HenrKyOWG2UGAoKJMExt1...
```

Below the terminal window, the command and its output are displayed again:

```
1 rows returned in 0ms from: SELECT password_salt FROM users WHERE
username = 'test';
```

4.4.4.2.2 Test code

```
if __name__ == '__main__':
    # For testing purposes use username test, password test
    while True:
        un = input("Enter username: ")
        user_exists = check_user_exists(un)
        print("User exists: {}".format(check_user_exists(un)))
        if user_exists:
            print(find_salt(un))
            break
```

4.4.4.2.3 With valid data

Enter username: test

User exists: True

9gu6g8dy9amkUZIzgAC4HenrKyOWG2UGAoKJMExt1FvYbgq90rHaHFhDBiY3aqZK

Process finished with exit code 0

4.4.4.2.4 With invalid data

N/A, any errors should be addressed before calling function by verifying username in advance

4.4.4.2.5 Evaluation

Function correctly returns the salt for the given username

4.4.5 Function to generate a salt

4.4.5.1 Final Code

```
def generate_salt() -> str:  
    # Securely generates a random salt  
    return uuid.uuid4().hex
```

Salt generation code based on example given at stackoverflow.com

4.4.5.2 Testing

4.4.5.2.1 Test code

```
if __name__ == '__main__':  
    print(generate_salt())
```

4.4.5.2.2 Outputs

d4b2cbe839314148a5211a0501ff31b4

Process finished with exit code 0

57892967fcb04f6292042be8974c1f34

Process finished with exit code 0

b10369320f7d4ec193d1f95ba0dd1cee

Process finished with exit code 0

4.4.5.2.3 Evaluation

Successfully and securely produces a unique valid salt every time function is run

4.4.6 Function to hash a password

4.4.6.1 Final Code

```
def generate_hash(password: str, salt: str) -> str:  
    # Combines password and salt and generates SHA256 hash of combined phrase  
    return hashlib.sha256((password + salt).encode('utf-8')).hexdigest()
```

Hashing code based on examples given at docs.python.org

4.4.6.2 Testing

4.4.6.2.1 Test code

```
if __name__ == '__main__':  
    salt = generate_salt()  
    print(salt)  
    pw = input("Enter password: ")  
    print(generate_hash(pw, salt))
```

4.4.6.2.2 Outputs

e1b22680841d48ae9b8aa86939bd9979

Enter password: test

082aaaed539e8a7a7fd6d727f8a684180202e87cc05f13b7ffdf7da4c52f3b9d

Process finished with exit code 0

ede37bd7488a4a708cdcb1959e4a366e

Enter password: abc123

b557bf8bff4914e5641565b29c6fe0be99461c4adeafe2c71de123535a637814

Process finished with exit code 0

6cb73d9f60014032bde66cf4c226183d

Enter password: password

d7b0f8dc737814e449abe26f2cb05658fc4b3840d48dac59f6b7258db58f77fd

Process finished with exit code 0

4.4.6.2.3 Evaluation

`test``e1b22680841d48ae9b8aa86939bd9979`

Treat each line as a separate string

SHA256 Hash of your string:
`082AAAED539E8A7A7FD6D727F8A684180202E87CC05F13B7FFDF7DA4C52F3B9D`

`abc123ede37bd7488a4a708cdcb1959e4a366e`

Treat each line as a separate string

SHA256 Hash of your string:
`B557BF8BFF4914E5641565B29C6FE0BE99461C4ADEAFE2C71DE123535A637814`

`password``6cb73d9f60014032bde66cf4c226183d`

Treat each line as a separate string

SHA256 Hash of your string:
`D7B0F8DC737814E449ABE26F2CB05658FC4B3840D48DAC59F6B7258DB58F77FD`

Function works as intended as all hashes generated at passwordsgenerator.net match hashes generated by program

4.4.7 Function to check if a password hash matches the stored hash

4.4.7.1 Final Code

```
# Assumes valid user id and salt provided
def check_hash(user_id: int, password: str, salt: str) -> bool:
    generated_hash: str = generate_hash(password, salt)
    # Verifies user id and hash are in database by counting number of times they
    occur together
    # This will be either 0 or 1 (not present, present)
    sql: str = 'SELECT COUNT(1) FROM users WHERE id = ? AND password_hash = ?;'
    c.execute(sql, (user_id, generated_hash))
    # If generated hash matched stored hash for given username return true, otherwise return false
    if c.fetchall()[0][0] > 0:
        return True
    else:
        return False
```

4.4.7.2 Testing

4.4.7.2.1 Test code

```
if __name__ == '__main__':
    # For testing purposes use username test, password test12345
    un = input("Enter username: ")
    pw = input("Enter password: ")
    if check_hash(un, pw, find_salt(un)):
        print("Valid")
    else:
        print("Invalid")
```

4.4.7.2.2 With valid data

Enter username: test

Enter password: test12345

Valid

Process finished with exit code 0

4.4.7.2.3 With invalid data

4.4.7.2.3.1 Invalid username

Enter username: qwerty

Enter password: test12345

Invalid

Process finished with exit code 0

4.4.7.2.3.2 Invalid password

Enter username: test

Enter password: qwerty

Invalid

Process finished with exit code 0

4.4.7.2.3.3 Invalid username and password

Enter username: qwerty

Enter password: qwerty

Invalid

Process finished with exit code 0

4.4.7.2.4 Evaluation

Function correctly returns that login data is valid only if both the username and salted password are correct

4.4.8 Function to check if a password is correct for the given username

4.4.8.1 Final Code

```
# Assumes valid username
def check_password(user_id: int, password: str) -> bool:
    # Returns whether password entered for given user id is valid
    return check_hash(user_id, password, find_salt(user_id))
```

4.4.8.2 Testing

4.4.8.2.1 Test code

```
if __name__ == '__main__':
    # For testing purposes use username test, password test12345
    un = input("Enter username: ")
    pw = input("Enter password: ")
    if check_password(un, pw):
        print("Valid")
    else:
        print("Invalid")
```

4.4.8.2.2 With valid data

Enter username: test

Enter password: test12345

Valid

Process finished with exit code 0

4.4.8.2.3 With invalid data

4.4.8.2.3.1 Invalid username

Enter username: qwerty

Enter password: test12345

Invalid

Process finished with exit code 0

4.4.8.2.3.2 Invalid password

Enter username: test

Enter password: qwerty

Invalid

Process finished with exit code 0

4.4.8.2.3.3 Invalid username and password

Enter username: qwerty

Enter password: qwerty

Invalid

Process finished with exit code 0

4.4.8.2.4 Evaluation

Function correctly returns that login data is valid only if both the username and password are correct

4.4.9 Function to create an account (insert valid data into the users table)

4.4.9.1 Issues during development

Initially, the following code was used:

```
def create_account(username: str, password: str, first_name: str, last_name: str,
account_type: str):
    # Generates a unique random salt
    salt: str = generate_salt()
    sql: str = 'INSERT INTO users(username, password_salt, password_hash,
first_name, last_name, type) ' \
              'VALUES(?, ?, ?, ?, ?, ?);'
    # Executes SQL to insert a correctly formatted record into the users table
    c.execute(sql,
              (username.casefold(), salt, generate_hash(password, salt), for-
mat_name(first_name),
               format_name(last_name), account_type))
```

This would result in no changes being made to the database. This was because simply executing an insert statement does permanently make changes to the database on secondary storage, the line 'conn.commit()' must be used to permanently apply the changes made and insert the data

4.4.9.2 Final Code

```
def create_account(username: str, password: str, first_name: str, last_name: str, account_type: str):
    # Generates a unique random salt
    salt: str = generate_salt()
    sql: str = 'INSERT INTO users(username, password_salt, password_hash, first_name, last_name, account_type) \
               VALUES(?, ?, ?, ?, ?, ?);'
    # Executes SQL to insert a correctly formatted record into the users table
    c.execute(sql,
              (username.casefold(), salt, generate_hash(password, salt), format_name(first_name),
               format_name(last_name), account_type))
    conn.commit()
```

4.4.9.3 Testing

4.4.9.3.1 SQL

```
1 INSERT INTO users(username, password_salt, password_hash, first_name, last_name, account_type) VALUES('a', 'b', 'c', 'd', 'e', 'f');
```

Query executed successfully: INSERT INTO users(username, password_salt, password_hash, first_name, last_name, account_type) VALUES('a', 'b', 'c', 'd', 'e', 'f'); (took 1ms, 1 rows affected)

	user_id	username	password_salt	password_hash	first_name	last_name	account_type
	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	test	6a29c0b26e1...	ce005846a2e...	te	st	s
2	2	a	b	c	d	e	f

4.4.9.3.2 Test code

```
if __name__ == '__main__':
    # Area to be used for testing purposes use username test, password test12345
    un = input("Enter username: ")
    pw = input("Enter password: ")
    first = input("Enter first name: ")
    last = input("Enter last name: ")
    user_type = input("Enter user type ('s' or 't'): ")
    create_account(un, pw, first, last, user_type)
```

4.4.9.3.3 With valid data:

Enter username: a

Enter password: b

Enter first name: c

Enter last name: d

Enter user type ('s' or 't'): s

Process finished with exit code 0

	user_id	username	password_salt	password_hash	first_name	last_name	account_type
	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	test	9gu6g8dy9am...	da920855818...	te	st	s
2	2	a	7e370df6ec2a...	5aa448557cb...	c	d	s

4.4.9.3.4 With Invalid data:

N/A, all data should be validated before being passed into the function

4.4.9.3.5 Evaluation

Correctly inserts new data into the database as expected with the originally entered password now in hashed and salted form

4.4.10 Function to get the user id which matches a given username

4.4.10.1 Final Code

```
def get_user_id(username: str) -> int:
    # Returns user id from database for given username
    sql: str = 'SELECT id FROM users WHERE username = ?;'
    c.execute(sql, (username,))
    return c.fetchall()[0][0]
```

4.4.10.2 Testing

4.4.10.2.1 SQL

The screenshot shows the DB Browser for SQLite interface. In the SQL tab, the query `SELECT id FROM users WHERE username = 'student45';` is run, resulting in one row returned with id 45. The results are displayed in a table with columns id, name, commit, last modified, and size.

id	name	commit	last modified	size
45	student45			

4.4.10.2.2 Test code

```
if __name__ == '__main__':
    un = input("Enter username: ")
    print(get_user_id(un))
```

4.4.10.2.3 With valid data:

Enter username: student13

13

Process finished with exit code 0

4.4.10.2.4 With Invalid data:

N/A, any errors should be addressed before calling function by verifying username in advance

4.4.10.2.5 Evaluation

Function returns correctly matching user id for given username if present in database

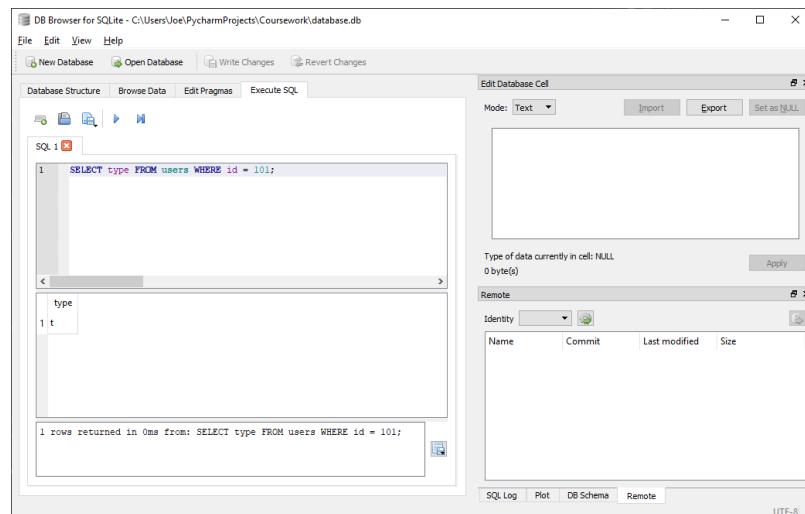
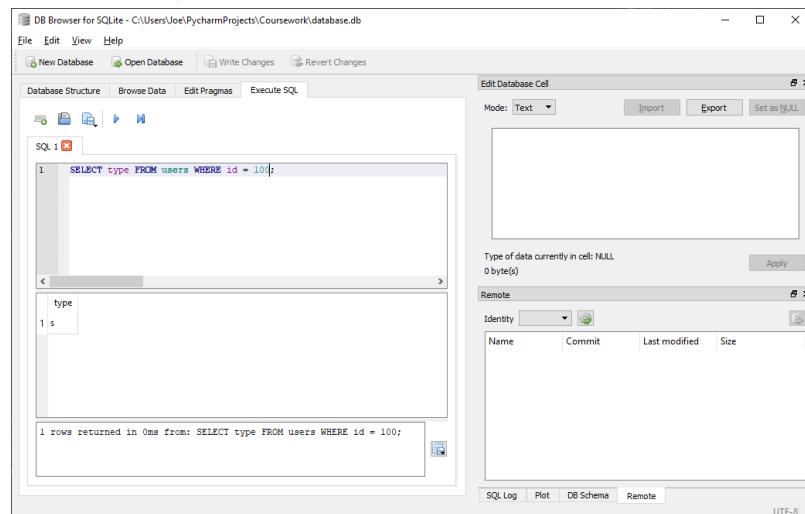
4.4.11 Function to get the account type for a given user id

4.4.11.1 Final Code

```
def get_account_type(user_id: int) -> str:  
    # Returns account type from database for given id  
    sql: str = 'SELECT type FROM users WHERE id = ?;'  
    c.execute(sql, (user_id,))  
    return c.fetchall()[0][0]
```

4.4.11.2 Testing

4.4.11.2.1 SQL



4.4.11.2.2 Test code

```
if name == '__main__':  
    uid = input("Enter user id: ")  
    print(get_account_type(uid))
```

4.4.11.2.3 With valid data:

Enter user id: 100

s

Process finished with exit code 0

Enter user id: 101

t

Process finished with exit code 0

4.4.11.2.4 With Invalid data:

N/A, any errors should be addressed before calling function by verifying user id in advance

4.4.11.2.5 Evaluation

Function returns correctly matching user type for given user id if present in database

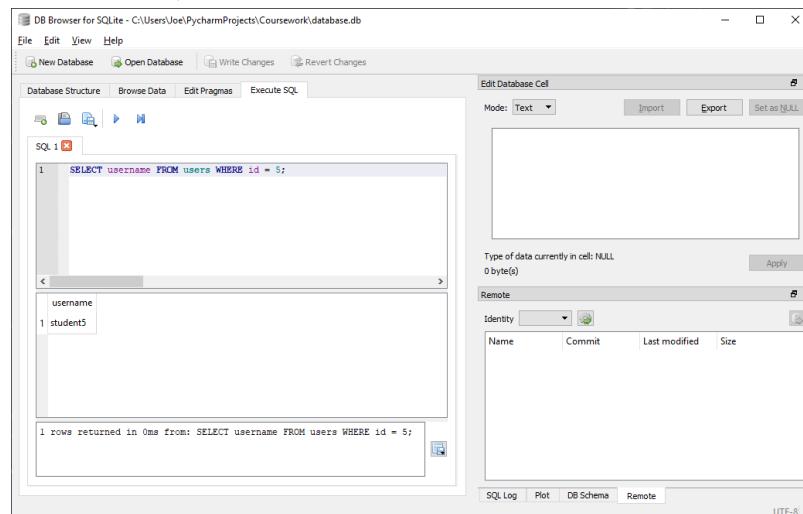
4.4.12 Function to get the username for a given user id

4.4.12.1 Final Code

```
def get_username(user_id: int) -> str:  
    # Returns username from database for given id  
    sql: str = 'SELECT username FROM users WHERE id = ?;'  
    c.execute(sql, (user_id,))  
    return c.fetchall()[0][0]
```

4.4.12.2 Testing

4.4.12.2.1 SQL



4.4.12.2.2 Test code

```
if __name__ == '__main__':  
    uid = input("Enter user id: ")  
    print(get_username(uid))
```

4.4.12.2.3 With valid data:

Enter user id: 134

teacher34

Process finished with exit code 0

4.4.12.2.4 With Invalid data:

N/A, any errors should be addressed before calling function by verifying user id in advance

4.4.12.2.5 Evaluation

Function returns correctly matching username for given user id if present in database

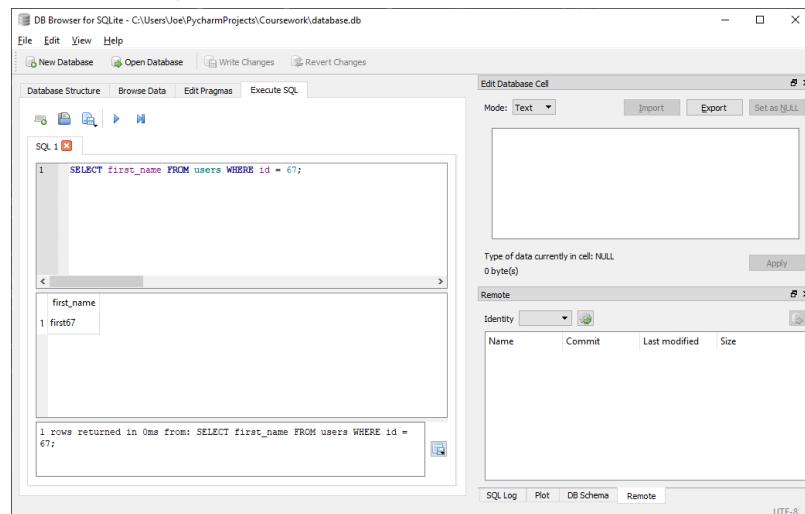
4.4.13 Function to get the first name for a given user id

4.4.13.1 Final Code

```
def get_first_name(user_id: int) -> str:  
    # Returns first name from database for given id  
    sql: str = 'SELECT first_name FROM users WHERE id = ?;  
    c.execute(sql, (user_id,))  
    return c.fetchall()[0][0]
```

4.4.13.2 Testing

4.4.13.2.1 SQL



4.4.13.2.2 Test code

```
if __name__ == '__main__':  
    uid = input("Enter user id: ")  
    print(get_first_name(uid))
```

4.4.13.2.3 With valid data:

Enter user id: 154

first54

Process finished with exit code 0

4.4.13.2.4 With Invalid data:

N/A, any errors should be addressed before calling function by verifying user id in advance

4.4.13.2.5 Evaluation

Function returns correctly matching first name for given user id if present in database

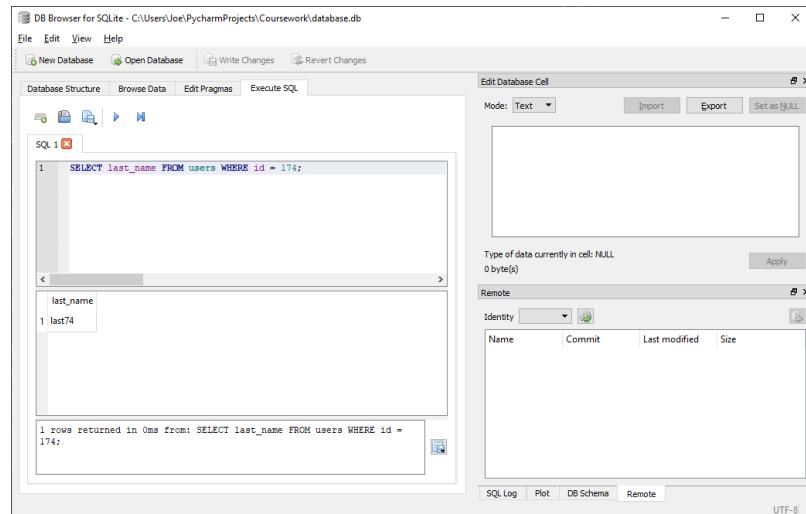
4.4.14 Function to get the last name for a given user id

4.4.14.1 Final Code

```
def get_last_name(user_id: int) -> str:  
    # Returns last name from database for given id  
    sql: str = 'SELECT last_name FROM users WHERE id = ?;  
    c.execute(sql, (user_id,))  
    return c.fetchall()[0][0]
```

4.4.14.2 Testing

4.4.14.2.1 SQL



4.4.14.2.2 Test code

```
if __name__ == '__main__':  
    uid = input("Enter user id: ")  
    print(get_last_name(uid))
```

4.4.14.2.3 With valid data:

Enter user id: 34

last34

Process finished with exit code 0

4.4.14.2.4 With Invalid data:

N/A, any errors should be addressed before calling function by verifying user id in advance

4.4.14.2.5 Evaluation

Function returns correctly matching last name for given user id if present in database

4.4.15 Function to format an inputted first/last name according to database rules (all lowercase)

4.4.15.1 Final Code

```
def format_name(first_name: str) -> str:  
    # Formats a first or last name for the database (all lowercase)  
    return first_name.casefold()
```

4.4.15.2 Testing

4.4.15.2.1 Test code

```
if __name__ == '__main__':  
    name = input("Enter name: ")  
    print(format_name(name))
```

4.4.15.2.2 With valid data:

Enter name: AbCdEfG

abcdefg

Process finished with exit code 0

4.4.15.2.3 With Invalid data:

N/A, any errors should be addressed before calling function by ensuring type passed is string

4.4.15.2.4 Evaluation

Function returns correctly converts all characters to lowercase

4.4.16 Function to update first name in database for a given user id

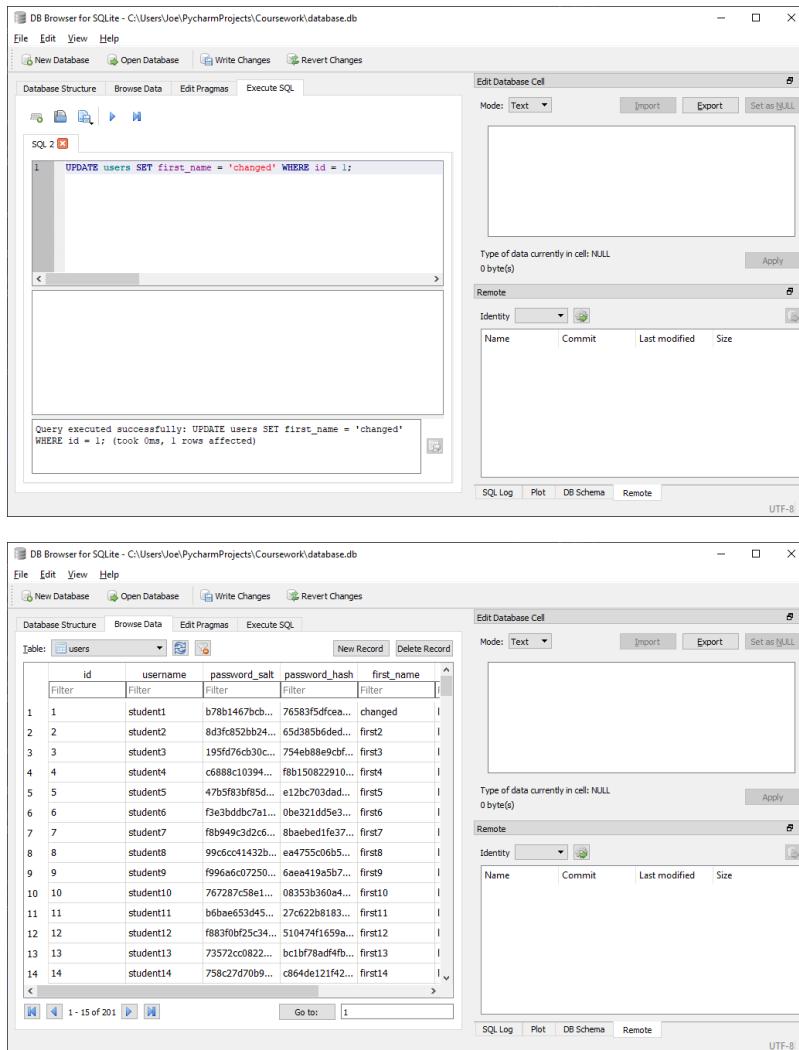
4.4.16.1 Final Code

```
def update_first_name(user_id: int, new_first_name: str):  
    # Updates value for first name in database for given id  
    sql: str = 'UPDATE users SET first_name = ? WHERE id = ?;  
    c.execute(sql, (format_name(new_first_name), user_id))  
    conn.commit()
```

4.4.16.2 Testing

4.4.16.2.1 SQL

	id	username	password_salt	password_hash	first_name
1	1	student1	b78b1467bc...	76583f5dfcea...	first1
2	2	student2	8dfcf652bb24...	65d385b6ded...	first2
3	3	student3	195fd76cb30c...	754eb88e9cbf...	first3
4	4	student4	c6888c10394...	f8b150822910...	first4
5	5	student5	47b5f83bf85d...	e12bc703dad...	first5
6	6	student6	f3e3bdbc7a1...	0be321dd5e3...	first6
7	7	student7	f8b949c3d2c6...	8baebef1f37...	first7
8	8	student8	99cbc41432b...	e4755c06b5...	first8
9	9	student9	f996a6c07250...	6aea419a5b7...	first9
10	10	student10	767287c58e1...	08353b360a4...	first10
11	11	student11	b6bae653d45...	27c622b8183...	first11
12	12	student12	f883f0bf25c34...	510474f1659b...	first12
13	13	student13	73572cc0822...	bc1bf78adff4b...	first13
14	14	student14	758c27d70b9...	c864de121f42...	first14



4.4.16.2.2 Test code

```
if __name__ == '__main__':
    uid = input("Enter user id: ")
    print("First name:", get_first_name(uid))
    update_first_name(uid, 'test')
    print("First name:", get_first_name(uid))
```

4.4.16.2.3 With valid data:

Enter user id: 67

First name: first67

First name: test

Process finished with exit code 0

4.4.16.2.4 With Invalid data:

N/A, any errors should be addressed before calling function by verifying user id in advance

4.4.16.2.5 Evaluation

Function returns correctly changes database value for first name for given user id

4.4.17 Function to update last name in database for a given user id

4.4.17.1 Final Code

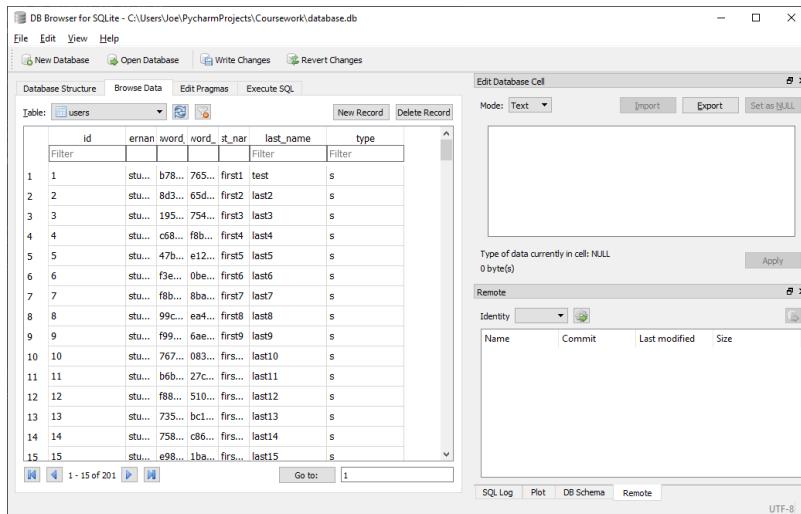
```
def update_last_name(user_id: int, new_last_name: str):
    # Updates value for last name in database for given id
    sql: str = 'UPDATE users SET last_name = ? WHERE id = ?;'
    c.execute(sql, (format_name(new_last_name), user_id))
    conn.commit()
```

4.4.17.2 Testing

4.4.17.2.1 SQL

The screenshot shows the 'DB Browser for SQLite' interface. On the left, the 'users' table is displayed with 15 rows of data. Row 1 has the 'id' column set to 1. An 'Edit Database Cell' dialog is open over this row, specifically for the 'last_name' column. The current value is 'last1'. A tooltip indicates the type is 'Text / Numeric' and the length is '1 char(s)'. The 'Apply' button is visible. On the right side of the interface, there are tabs for 'SQL Log', 'Plot', 'DB Schema', and 'Remote', with 'SQL Log' currently selected.

The screenshot shows the 'DB Browser for SQLite' interface with the 'SQL' tab selected. A single-line query 'UPDATE users SET last_name = 'test' WHERE id = 1;' is entered into the SQL input field. To the right, the 'Edit Database Cell' dialog is still open over the first row of the 'users' table, showing the 'last_name' column with the value 'test'. A tooltip indicates the type is 'NULL' and the length is '0 byte(s)'. Below the SQL input field, a message box displays the result of the query: 'Query executed successfully: UPDATE users SET last_name = 'test' WHERE id = 1; (took 0ms, 1 rows affected)'.



4.4.17.2.2 Test code

```
if __name__ == '__main__':
    uid = input("Enter user id: ")
    print("Last name:", get_last_name(uid))
    update_last_name(uid, 'test')
    print("Last name:", get_last_name(uid))
```

4.4.17.2.3 With valid data:

Enter user id: 134

Last name: last34

Last name: test

Process finished with exit code 0

4.4.17.2.4 With Invalid data:

N/A, any errors should be addressed before calling function by verifying user id in advance

4.4.17.2.5 Evaluation

Function returns correctly changes database value for last name for given user id

4.4.18 Function to update password in database for a given user id

4.4.18.1 Final Code

```
def update_password(user_id: int, password: str):
    # Updates value for hashed password in database for given id
    sql: str = 'UPDATE users SET password_hash = ? WHERE id = ?;'
    # Generates a new password hash to be inserted from salt stored for given
    username
    c.execute(sql, (generate_hash(password, find_salt(user_id)), user_id))
    conn.commit()
```

4.4.18.2 Testing

4.4.18.2.1 SQL

The screenshot shows the 'users' table in DB Browser for SQLite. The table has columns: id, username, password_salt, password_hash, and first_name. The data consists of 14 rows of student records. An 'Edit Database Cell' dialog is open over the first row (id=1), showing the current value 'first1' in the 'password_hash' column.

	id	username	password_salt	password_hash	first_name
1	student1	b78b1467bcb...	76583f5dcea...	first1	
2	student2	8dfcfb52bb24...	65d385b6ded...	first2	
3	student3	195fd76cb30c...	754eb88e9cbf...	first3	
4	student4	c6888c10394...	f8b150822910...	first4	
5	student5	47b5f83bf85d...	e12bc703dad...	first5	
6	student6	f3e3bdbcb7a1...	0be321dd5e3...	first6	
7	student7	f8b949c3d2e6...	8baebed1fe37...	first7	
8	student8	99c6cc41432b...	ea4755c06b5...	first8	
9	student9	f996ad6c07250...	6aae419a5b7...	first9	
10	student10	767287c58e1...	08353b360a4...	first10	
11	student11	b6bae653d45...	27c622b08183...	first11	
12	student12	f883fb25c34...	510474f1659a...	first12	
13	student13	73572cc0822...	bc1b7f8ad4fb...	first13	
14	student14	758c27d70b9...	c864de121f42...	first14	

The screenshot shows the 'users' table with 14 rows of student data. An 'Edit Database Cell' dialog is open over the first row (id=1), showing the current value 'first1' in the 'password_hash' column. A SQL log window shows the execution of the following update query:

```
1 UPDATE users SET password_hash = 'test' WHERE id = 1;
```

The log message indicates: "Query executed successfully: UPDATE users SET password_hash = 'test' WHERE id = 1; (took 0ms, 1 rows affected)".

The screenshot shows the 'users' table with 14 rows of student data. The 'password_hash' column for the first row (id=1) has been updated to 'test'. An 'Edit Database Cell' dialog is open over the first row (id=1), showing the new value 'test' in the 'password_hash' column.

	id	username	password_salt	password_hash	first_name
1	student1	b78b1467bcb...	test	first1	
2	student2	8dfcfb52bb24...	65d385b6ded...	first2	
3	student3	195fd76cb30c...	754eb88e9cbf...	first3	
4	student4	c6888c10394...	f8b150822910...	first4	
5	student5	47b5f83bf85d...	e12bc703dad...	first5	
6	student6	f3e3bdbcb7a1...	0be321dd5e3...	first6	
7	student7	f8b949c3d2e6...	8baebed1fe37...	first7	
8	student8	99c6cc41432b...	ea4755c06b5...	first8	
9	student9	f996ad6c07250...	6aae419a5b7...	first9	
10	student10	767287c58e1...	08353b360a4...	first10	
11	student11	b6bae653d45...	27c622b08183...	first11	
12	student12	f883fb25c34...	510474f1659a...	first12	
13	student13	73572cc0822...	bc1b7f8ad4fb...	first13	
14	student14	758c27d70b9...	c864de121f42...	first14	

4.4.18.2.2 Test code

```
if __name__ == '__main__':
    uid = input("Enter user id: ")
    print("Password is test:", check_password(uid, 'test'))
    update_password(uid, 'test')
    print("Password is test:", check_password(uid, 'test'))
```

4.4.18.2.3 With valid data:

Enter user id: 45

Password is test: False

Password is test: True

Process finished with exit code 0

4.4.18.2.4 With Invalid data:

N/A, any errors should be addressed before calling function by verifying user id in advance

4.4.18.2.5 Evaluation

Function returns correctly changes database value for password hash for given user id

4.5 UI SCRIPTS (SCRIPTS.UI_SCRIPTS)

4.5.1 Import required Python libraries to communicate with SQLite database and use scripts from login scripts

4.5.1.1 Final Code

```
import sqlite3
import scripts.db_scripts as db_scripts
```

4.5.2 Connect to database

4.5.2.1 Final Code

```
# Connects to database
conn = sqlite3.connect('database.db')
c = conn.cursor()
```

Generic code used

4.5.3 Function to get a list of all classes for a given user id

4.5.3.1 Final Code

```
def get_classes_of_student(user_id: int) -> list:
    # Takes a user id, finds all classes student is a member of, and returns the
    class_id and name for each relevant
    # class from the classes table by joining all relevant tables appropriately
    sql: str = 'SELECT classes.id, classes.name ' \
               'FROM ((class_user INNER JOIN classes ON class_user.class_id = clas-'
    ses.id) ' \
               'INNER JOIN users ON users.id = class_user.student_id) ' \
               'WHERE users.id = ?;'
    c.execute(sql, (user_id,))
    return c.fetchall()
```

4.5.3.2 Testing

4.5.3.2.1 SQL

The screenshot displays two windows of DB Browser for SQLite. The top window shows the results of a SQL query:

```
SELECT classes.id, classes.name FROM ((class_user INNER JOIN classes ON class_user.class_id = classes.id) INNER JOIN users ON users.id = class_user.student_id) WHERE users.id = 5;
```

The results show 103 rows returned in 7ms, listing various class IDs and names. The bottom window shows the 'class_user' table with 15 rows, where user 5 is listed multiple times.

id	class_id	student_id
1	456	5
2	573	5
3	33	5
4	112	5
5	235	5
6	524	5
7	1083	5
8	1106	5
9	1131	5
10	1251	5
11	1533	5
12	1842	5
13	1888	5
14	1955	5
15	1962	5

Correctly finds all 103 classes user 5 is in and outputs the required data (class id and class name)

4.5.3.2.2 Test code

```
if __name__ == '__main__':
    uid = input("Enter user id: ")
    print(get_classes_of_student(uid))
```

4.5.3.2.3 With valid data:

Enter user id: 67

```
[(232, 'class232'), (26, 'class26'), (8, 'class8'), (32, 'class32'), (150, 'class150'), (404, 'class404'), (588, 'class588'), (103, 'class103'), (964, 'class964'), (575, 'class575'), (366, 'class366'), (482, 'class482'), (982, 'class982'), (690, 'class690'), (881, 'class881'), (867, 'class867'), (41, 'class41'), (903, 'class903'), (269, 'class269'), (187, 'class187'), (150, 'class150'), (140, 'class140'), (307, 'class307'), (298, 'class298'), (833, 'class833'), (590, 'class590'), (567, 'class567'), (559, 'class559'), (236, 'class236'), (685, 'class685'), (800, 'class800'), (243, 'class243'), (148, 'class148'), (776, 'class776'), (811, 'class811'), (115, 'class115'), (525, 'class525'), (845, 'class845'), (413, 'class413'), (259, 'class259'), (742, 'class742'), (290, 'class290'), (486, 'class486'), (539, 'class539'), (897, 'class897'), (841, 'class841'), (119, 'class119'), (623, 'class623'), (657, 'class657'), (808, 'class808'), (347, 'class347'), (171, 'class171'), (192, 'class192'), (33, 'class33'), (680, 'class680'), (880, 'class880'), (790, 'class790'), (227, 'class227'), (282, 'class282'), (143, 'class143'), (265, 'class265'), (512, 'class512'), (831, 'class831'), (554, 'class554'), (476, 'class476'), (104, 'class104'), (955, 'class955'), (40, 'class40'), (513, 'class513'), (157, 'class157'), (400, 'class400'), (128, 'class128'), (539, 'class539'), (973, 'class973'), (92, 'class92'), (969, 'class969'), (341, 'class341'), (185, 'class185'), (902, 'class902'), (94, 'class94'), (768, 'class768'), (405, 'class405'), (490, 'class490'), (14, 'class14'), (997, 'class997'), (834, 'class834'), (881, 'class881'), (139, 'class139'), (473, 'class473'), (579, 'class579'), (947, 'class947'), (36, 'class36'), (235, 'class235'), (334, 'class334'), (914, 'class914'), (395, 'class395'), (270, 'class270'), (49, 'class49'), (386, 'class386'), (35, 'class35'), (243, 'class243'), (323, 'class323'), (724, 'class724'), (187, 'class187'), (482, 'class482'), (349, 'class349'), (689, 'class689'), (717, 'class717'), (269, 'class269'), (629, 'class629'), (266, 'class266'), (539, 'class539'), (347, 'class347'), (258, 'class258'), (273, 'class273'), (784, 'class784'), (491, 'class491'), (435, 'class435'), (581, 'class581'), (558, 'class558'), (707, 'class707'), (585, 'class585'), (629, 'class629'), (250, 'class250'), (844, 'class844'), (7, 'class7'), (474, 'class474'), (863, 'class863'), (437, 'class437'), (183, 'class183')]
```

Process finished with exit code 0

4.5.3.2.4 With Invalid data:

N/A, any errors should be addressed before calling function by verifying user id in advance

4.5.3.2.5 Evaluation

Function correctly returns complete list of classes which contain user with given user id as an array of tuples of the format (class id, class name)

4.5.4 Function to get a list of all homework for a given class id

4.5.4.1 Final Code

```
def get_homework_of_class(class_id) -> list:
    # Takes a class id, finds all homework belonging to the class, and returns the
    # homework id and name for each
    # relevant homework from the homework table by joining all relevant tables ap-
    # propriately
    sql: str = 'SELECT homework.id, homework.name ' \
              'FROM homework INNER JOIN class_homework ON homework.id = '
    sql += class_homework.homework_id ' \
              'WHERE class_homework.class_id = ?;'
    c.execute(sql, (class_id,))
    return c.fetchall()
```

4.5.4.2 Testing

4.5.4.2.1 SQL

DB Browser for SQLite - C:\Users\Joe\PycharmProjects\Coursework\database.db

File Edit View Help

New Database Open Database Write Changes Revert Changes

Database Structure Browse Data Edit Pragmas Execute SQL

SQL 1

```
1 SELECT homework.id, homework.name FROM homework INNER JOIN class_homework ON homework.id = class_homework.homework_id WHERE class_homework.class_id = 56;
```

	id	name
1	3229	name3229
2	4338	name4338
3	7396	name7396
4	7494	name7494
5	8013	name8013
6	8464	name8464
7	8684	name8684
8	8997	name8997
9	9554	name9554
10	9930	name9930

10 rows returned in 1ms from: SELECT homework.id, homework.name FROM homework INNER JOIN class_homework ON homework.id = class_homework.homework_id WHERE class_homework.class_id = 56;

DB Browser for SQLite - C:\Users\Joe\PycharmProjects\Coursework\database.db

File Edit View Help

New Database Open Database Write Changes Revert Changes

Database Structure Browse Data Edit Pragmas Execute SQL

Table: class_homework

	id	class_id	homework_id	due_date
1	3229	56	3229	2011-3-10
2	4338	56	4338	2017-12-23
3	7396	56	7396	2013-9-25
4	7494	56	7494	2020-4-9
5	8013	56	8013	2010-12-9
6	8464	56	8464	2016-4-5
7	8684	56	8684	2013-9-22
8	8997	56	8997	2016-2-10
9	9554	56	9554	2019-3-3
10	9930	56	9930	2019-2-22

1 - 10 of 10

Correctly finds all 10 homework set for class 56 and outputs the required data (homework id and homework name)

4.5.4.2.2 Test code

```
if __name__ == '__main__':
    cid = input("Enter class id: ")
    print(get_homework_of_class(cid))
```

4.5.4.2.3 With valid data:

Enter class id: 123

```
[(682, 'name682'), (1547, 'name1547'), (1995, 'name1995'), (2196, 'name2196'), (2455, 'name2455'), (2613, 'name2613'), (4096, 'name4096'), (4882, 'name4882'), (6172, 'name6172'), (7341, 'name7341'), (7424, 'name7424'), (8301, 'name8301'), (8638, 'name8638'), (8769, 'name8769'), (9516, 'name9516')]
```

Process finished with exit code 0

4.5.4.2.4 With Invalid data:

N/A, any errors should be addressed before calling function by verifying class id in advance

4.5.4.2.5 Evaluation

Function correctly returns complete list of homework set by class with given class id as an array of tuples of the format (homework id, homework name)

4.5.5 Function to get a list of all classes for a given teacher id

4.5.5.1 Final Code

```
def get_classes_of_teacher(user_id: int) -> list:  
    # Takes a user id (assumed to be that of a teacher and to be verified before  
    # calling function) and returns the  
    # class id and name for each relevant class from the classes table  
    sql: str = 'SELECT id, name FROM classes WHERE teacher = ?;'  
    c.execute(sql, (user_id,))  
    return c.fetchall()
```

4.5.5.2 Testing

4.5.5.2.1 SQL

The screenshot shows the DB Browser for SQLite interface. On the left, there is a SQL editor window containing the query: `SELECT id, name FROM classes WHERE teacher = 145;`. Below the query, the results are displayed in a table:

	id	name
1	42	class42
2	58	class58
3	124	class124
4	141	class141
5	291	class291

Below the table, it says "12 rows returned in 1ms from: SELECT id, name FROM classes WHERE teacher = 145;".

The screenshot shows the DB Browser for SQLite interface. On the left, there is a table viewer for the "classes" table. A filter is applied to the "teacher" column with the value "145". The results are displayed in a table:

	id	name	teacher
1	42	class42	145
2	58	class58	145
3	124	class124	145
4	141	class141	145
5	291	class291	145
6	523	class523	145
7	618	class618	145
8	669	class669	145
9	685	class685	145
10	699	class699	145
11	821	class821	145
12	907	class907	145

Correctly finds all 12 classes under teacher id 145 and outputs the required data (class id and class name)

4.5.5.2.2 Test code

```
if __name__ == '__main__':
    tid = input("Enter teacher id: ")
    print(get_classes_of_teacher(tid))
```

4.5.5.2.3 With valid data:

Enter teacher id: 156

```
[(4, 'class4'), (43, 'class43'), (79, 'class79'), (93, 'class93'), (160, 'class160'), (233, 'class233'), (393, 'class393'), (597, 'class597'), (753, 'class753'), (801, 'class801'), (808, 'class808'), (906, 'class906'), (910, 'class910')]
```

Process finished with exit code 0

4.5.5.2.4 With Invalid data:

N/A, any errors should be addressed before calling function by verifying teacher id in advance

4.5.5.2.5 Evaluation

Function correctly returns complete list of classes for teacher with given teacher id as an array of tuples of the format (class id, class name)

4.5.6 Function to get a list of all students for a given class id

4.5.6.1 Final Code

```
def get_students_of_class(class_id: int) -> list:
    # Takes a class id, finds all students belonging to the class, and returns the
    student_id and name for each relevant
    # student from the users table by joining all relevant tables appropriately
    sql: str = 'SELECT users.id, users.username ' \
               'FROM class_user INNER JOIN users ON class_user.student_id = us-
    ers.id ' \
               'WHERE class_user.class_id = ?;'
    c.execute(sql, (class_id,))
    return c.fetchall()
```

4.5.6.2 Testing

4.5.6.2.1 SQL

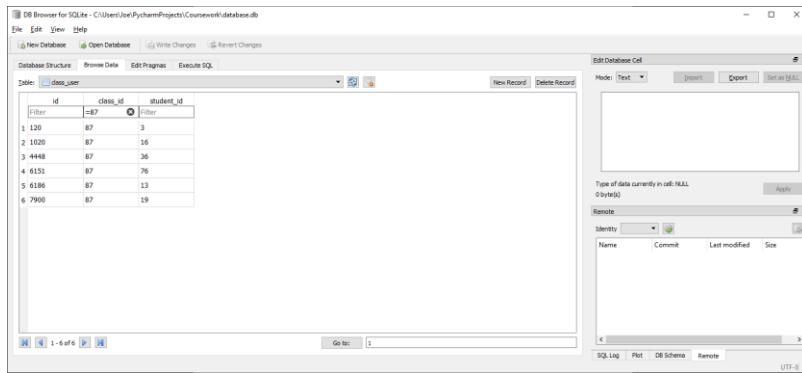
The screenshot shows the DB Browser for SQLite interface. In the top-left window, there is an SQL editor with the following query:

```
SELECT users.id, users.username FROM class_user INNER JOIN users ON class_user.student_id = users.id WHERE class_user.class_id = 42;
```

In the bottom-left window, the results of the query are displayed in a table:

	id	username
1	1	student3
2	16	student16
3	36	student36
4	76	student76
5	113	student113
6	115	student115

The bottom-right window shows the database schema with a single table named 'Remote' containing columns Name, Content, Last modified, and Size.



Correctly finds all 6 students under class id 87 and outputs the required data (user id and username)

4.5.6.2.2 Test code

```
if __name__ == '__main__':
    cid = input("Enter class id: ")
    print(get_students_of_class(cid))
```

4.5.6.2.3 With valid data:

Enter class id: 162

```
[(60, 'student60'), (73, 'student73'), (92, 'student92'), (83, 'student83'), (63, 'student63'), (43, 'student43'), (53, 'student53'), (10, 'student10'), (79, 'student79'), (41, 'student41'), (37, 'student37'), (93, 'student93')]
```

Process finished with exit code 0

4.5.6.2.4 With Invalid data:

N/A, any errors should be addressed before calling function by verifying class id in advance

4.5.6.2.5 Evaluation

Function correctly returns complete list of students for class with given class id as an array of tuples of the format (user id, username)

4.5.7 Function to get a list of all questions for a given homework id

4.5.7.1 Final Code

```
def get_questions_of_homework(homework_id: int) -> list:
    # Takes a homework id, finds all questions belonging to the homework, and returns the question id and name for each
    # relevant question from the questions table by joining all relevant tables appropriately
    sql: str = 'SELECT questions.id, questions.name FROM ' \
               'questions INNER JOIN homework_questions ON questions.id = homework_questions.question_id ' \
               'WHERE homework_questions.homework_id = ?; '
    c.execute(sql, (homework_id,))
    return c.fetchall()
```

4.5.7.2 Testing

4.5.7.2.1 SQL

The screenshot displays two windows of the DB Browser for SQLite application. The top window shows the results of a SQL query:

```
1 SELECT questions.id, questions.name FROM questions INNER JOIN homework_questions ON questions.id = homework_questions.question_id WHERE homework_questions.homework_id = 567;
```

The results table has columns 'id' and 'name', containing 10 rows:

	id	name
1	1984	question1984
2	4688	question4688
3	25303	question25303
4	33704	question33704
5	49598	question49598
6	57507	question57507
7	59427	question59427
8	64798	question64798
9	71598	question71598
10	88124	question88124

10 rows returned in 6ms from: SELECT questions.id, questions.name FROM questions INNER JOIN homework_questions ON questions.id = homework_questions.question_id WHERE homework_questions.homework_id = 567;

The bottom window shows the 'homework_questions' table with columns 'id', 'homework_id', and 'question_id', containing 10 rows:

	id	homework_id	question_id
1	1984	567	1984
2	4688	567	4688
3	25303	567	25303
4	33704	567	33704
5	49598	567	49598
6	57507	567	57507
7	59427	567	59427
8	64798	567	64798
9	71598	567	71598
10	88124	567	88124

Correctly finds all 10 questions under homework id 567 and outputs the required data (question id and question name)

4.5.7.2.2 Test code

```
if name == 'main':
    hid = input("Enter homework id: ")
    print(get_questions_of_homework(hid))
```

4.5.7.2.3 With valid data:

Enter homework id: 872

```
[(3796, 'question3796'), (4329, 'question4329'), (10583, 'question10583'), (13404, 'question13404'), (25395, 'question25395'), (26223, 'question26223'), (31890, 'question31890'), (56620, 'question56620'), (60745, 'question60745'), (62948, 'question62948'), (66821, 'question66821'), (67335, 'question67335'), (77670, 'question77670'), (82953, 'question82953'), (87138, 'question87138'), (99381, 'question99381')]
```

Process finished with exit code 0

4.5.7.2.4 With Invalid data:

N/A, any errors should be addressed before calling function by verifying homework id in advance

4.5.7.2.5 Evaluation

Function correctly returns complete list of questions for homework with given homework id as an array of tuples of the format (question id, question name)

4.5.8 Function to get the score for a given homework for a given user

4.5.8.1 Issues during development

Initially the below code was implemented:

```
def get_homework_score(user_id: int, homework_id: int, class_id: int) -> tuple:
    # Takes a user, class and homework id, finds the results for all questions in
    # the given homework for the given user,
    # and for each question returns a boolean value for if the question has been
    # correctly answered and the number of
    # times that question has been attempted from the question_results table by
    # joining all relevant tables
    # appropriately
    sql: str = 'SELECT question_results.attempts, question_results.correct ' \
               'FROM ((homework INNER JOIN homework_questions ON homework.id = '
    sql += 'homework_questions.homework_id)' \
               '\n                  'INNER JOIN questions ON homework_questions.question_id = ques-
    sql += 'tions.id)' \
               '\n                  'INNER JOIN question_results ON questions.id = question_re-
    sql += 'sults.question_id' \
               '\n                  'WHERE question_results.user_id = ? AND homework.id = ?;'
    c.execute(sql, (user_id, homework_id))
    score_list: tuple = c.fetchall()
    # Iterates through each question result stored in the score list and increments
    # the score for the homework by 1
    # divided by the number of attempts if the question was correctly answered (has
    # 'True' in the first position of the
    # tuple). This value is rounded to the nearest integer
    # Question count is incremented each loop to get a value for the number of
    # questions in the homework
    question_count: int = 0
    score: int = 0
    for each_score in score_list:
        question_count += 1
        if each_score[1] == 'T':
            score += (1 / each_score[0])
    # Calculates the percentage score for the homework by dividing the score by the
    # number of questions and multiplying
    # by 100
    result: int = round(score / question_count * 100)
    # Function to be returned to in UI also requires homework name and due date so
    # this data is found and returned with
    # the homework result
    name_and_due_date: tuple = get_homework_name_and_due_date(homework_id,
    class_id)
    return name_and_due_date[0], result, name_and_due_date[1]
```

This would result in an error occurring if for any reason the question count took a value of 0 (if a given homework has no questions or the given homework was never set for the given user and no results have therefore been stored).

Enter user id: 87

Enter homework id: 49

Enter class id: 778

Traceback (most recent call last):

```
File "C:/Users/Joe/PycharmProjects/Coursework/scripts/ui_scripts.py", line 214, in <module>
```

```
    print(get_homework_score(uid, hid))
```

```
File "C:/Users/Joe/PycharmProjects/Coursework/scripts/ui_scripts.py", line 56, in
get_homework_score
```

```
    result: int = round(score / question_count * 100)
```

ZeroDivisionError: division by zero

Process finished with exit code 1

This was fixed by placing the calculation of the percentage within a try statement and defining that if a division by zero error occurs then the score to be returned is “N/A” (i.e. the score being requested is invalid)

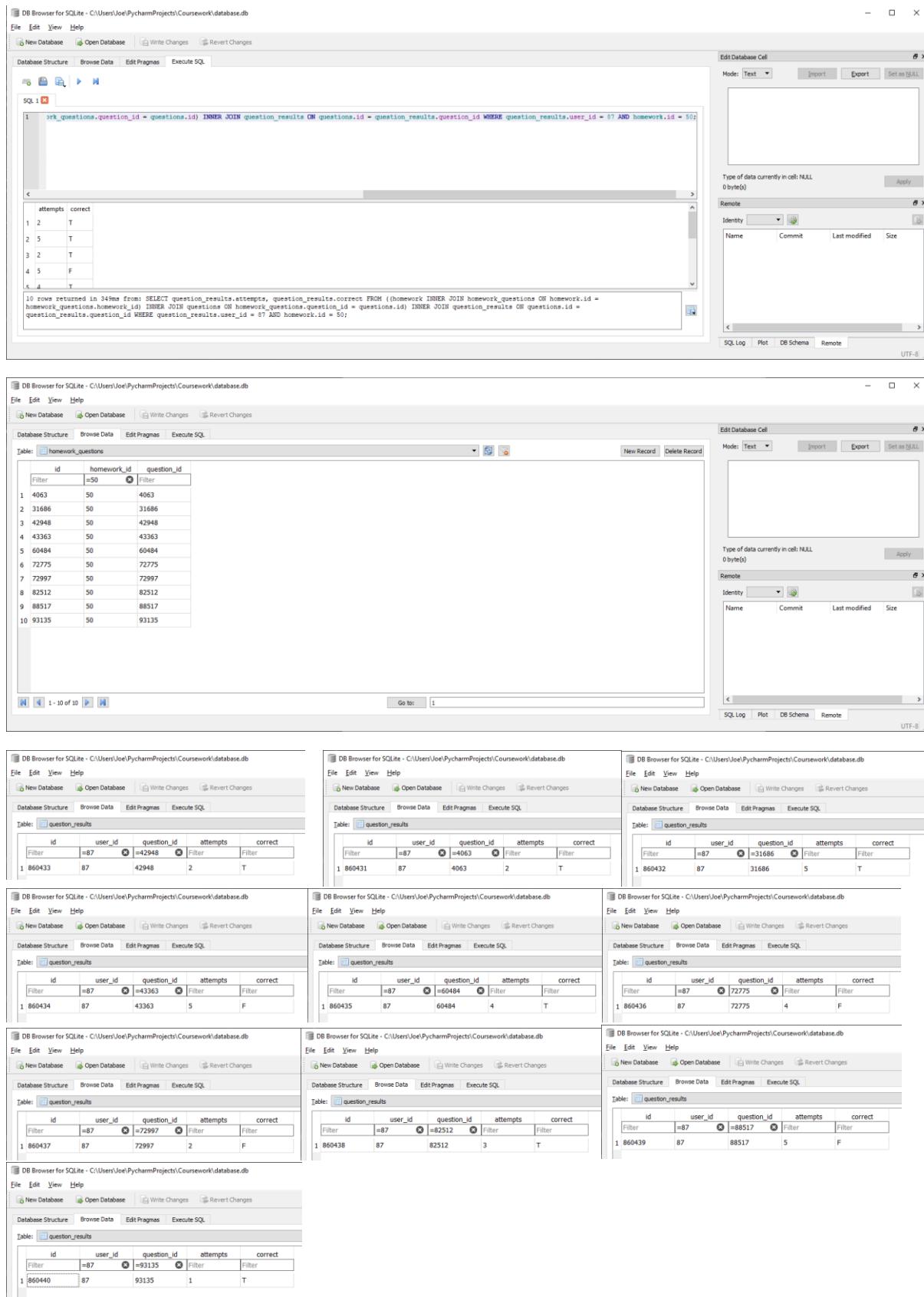
```
# Calculates the percentage score for the homework by dividing the score by the
# number of questions and multiplying
# by 100. If the homework has no questions (e.g. if the given student was never set
# the given homework), a division
# by zero will occur. This is handled by simply setting a score of "N/A"
try:
    result: int = round(score / question_count * 100)
except ZeroDivisionError:
    result = "N/A"
```

4.5.8.2 Final Code

```
def get_homework_score(user_id: int, homework_id: int, class_id: int) -> tuple:
    # Takes a user, class and homework id, finds the results for all questions in
    # the given homework for the given user,
    # and for each question returns a boolean value for if the question has been
    # correctly answered and the number of
    # times that question has been attempted from the question_results table by
    # joining all relevant tables
    # appropriately
    sql: str = 'SELECT question_results.attempts, question_results.correct ' \
               'FROM ((homework INNER JOIN homework_questions ON homework.id = '
    sql += f'homework_questions.homework_id) ' \
           '\n              'INNER JOIN questions ON homework_questions.question_id = ques-
    sql += f'tions.id) ' \
           '\n              'INNER JOIN question_results ON questions.id = question_re-
    sql += f'sults.question_id ' \
           '\n              'WHERE question_results.user_id = ? AND homework.id = ?;'
    c.execute(sql, (user_id, homework_id))
    score_list: tuple = c.fetchall()
    # Iterates through each question result stored in the score list and increments
    # the score for the homework by 1
    # divided by the number of attempts if the question was correctly answered (has
    # 'True' in the first position of the
    # tuple). This value is rounded to the nearest integer
    # Question count is incremented each loop to get a value for the number of
    # questions in the homework
    question_count: int = 0
    score: int = 0
    for each_score in score_list:
        question_count += 1
        if each_score[1] == 'T':
            score += (1 / each_score[0])
    # Calculates the percentage score for the homework by dividing the score by the
    # number of questions and multiplying
    # by 100. If the homework has no questions (e.g. if the given student was never
    # set the given homework), a division
    # by zero will occur. This is handled by simply setting a score of "N/A"
    try:
        result: int = round(score / question_count * 100)
    except ZeroDivisionError:
        result = "N/A"
    # Function to be returned to in UI also requires homework name and due date so
    # this data is found and returned with
    # the homework result
    name_and_due_date: tuple = get_homework_name_and_due_date(homework_id,
    class_id)
    return name_and_due_date[0], result, name_and_due_date[1]
```

4.5.8.3 Testing

4.5.8.3.1 SQL



Correctly finds results for the given user (87) for all questions within the homework with the given id (50) and for each question outputs the number of attempts and whether the correct answer has been achieved

4.5.8.3.2 Test code

```
if __name__ == '__main__':
    uid: int = input("Enter user id: ")
    hid: int = input("Enter homework id: ")
    cid: int = input("Enter class id: ")
    print(get_homework_score(uid, hid, cid))
```

4.5.8.3.3 With valid data:

Enter user id: 87

Enter homework id: 50

Enter class id: 778

('name50', 28, '2017-9-4')

Process finished with exit code 0

For this user and homework id the correct answer should be

$$(((1/2) + (1/2) + (1/5) + (1/4) + (1/3) + (1/1)) / 10) * 100 = 27.8333333333$$

This rounds to 28% to the nearest whole %

4.5.8.3.4 With Invalid data:

No question results for given user for given homework (e.g. if a given homework has no questions or the given homework was never set for the given user and no results have therefore been stored)

Enter user id: 87

Enter homework id: 49

('name49', 'N/A', '2018-5-21')

Process finished with exit code 0

Errors relating to whether the given homework is within the given class should be addressed before calling the function

4.5.8.3.5 Evaluation

Function correctly returns the score for a given user in a given homework as a tuple of the format (homework name, score, due date) where the score is either a percentage or, if the requested score is for any reason invalid, "N/A"

4.5.9 Function to get the homework name and due date for a given homework id (used in `get_homework_score`)

4.5.9.1 Final Code

```
def get_homework_name_and_due_date(homework_id: int, class_id: int) -> tuple:
    # Takes a homework and class id and returns the relevant name and due date from
    # the homework and class_homework
    # tables respectively by joining all relevant tables appropriately
    sql: str = 'SELECT homework.name, class_homework.due_date ' \
               'FROM homework INNER JOIN class_homework ON homework.id = ' \
               'class_homework.homework_id ' \
               'WHERE homework.id = ? AND class_homework.class_id = ?;'
    c.execute(sql, (homework_id, class_id))
    return c.fetchall()[0]
```

4.5.9.2 Testing

4.5.9.2.1 SQL



The screenshot shows the DB Browser for SQLite interface. On the left, there's a SQL editor window with the following query:

```
1 SELECT homework.name, class_homework.due_date FROM homework INNER JOIN class_homework ON homework.id = class_homework.homework_id WHERE homework.id = 1864 AND class_homework.class_id = 135;
```

On the right, there's a results grid showing one row of data:

name	due_date
name1864	2014-3-9

Below the results, a message indicates 1 row was returned.

The screenshot shows two windows of DB Browser for SQLite. The top window displays the 'homework' table with the following data:

id	name	description
1864	name1864	description1864

The bottom window displays the 'class_homework' table with the following data:

id	class_id	homework_id	due_date
1864	135	1864	2014-3-9

Correctly finds and returns the name and due date stored in the database for homework id 1864 within class 135

4.5.9.2.2 Test code

```
if __name__ == '__main__':
    hid: int = input("Enter homework id: ")
    cid: int = input("Enter class id: ")
    print(get_homework_name_and_due_date(hid, cid))
```

4.5.9.2.3 With valid data:

Enter homework id: 7555

Enter class id: 67

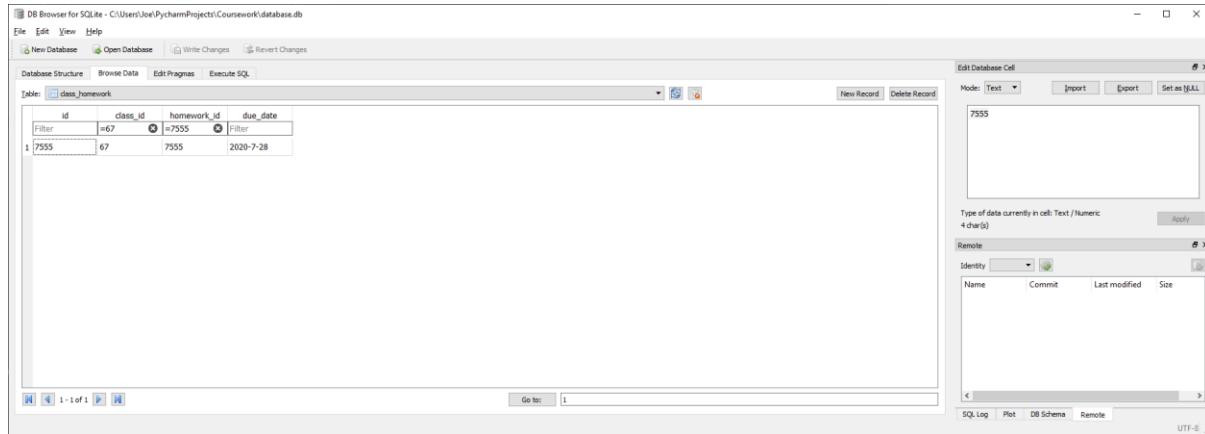
('name7555', '2020-7-28')

Process finished with exit code 0

4.5.9.2.4 With Invalid data:

N/A, any errors should be addressed before calling function by verifying given homework is assigned to given class

4.5.9.2.5 Evaluation



The screenshot shows the DB Browser for SQLite interface. The main window displays a table named 'class_homework' with four columns: id, class_id, homework_id, and due_date. A single row is selected, showing values: id=7555, class_id=67, homework_id=7555, and due_date=2020-7-28. The 'Edit Database Cell' panel on the right shows the value '7555' in the 'Text / Numeric' field. The status bar at the bottom indicates 'UTF-8' encoding.

Function correctly returns list of the format (homework name, homework due date) for the given homework and class ids

4.5.10 Function to insert a student into a class

4.5.10.1 Final Code

```
def add_student_to_class(student_id: int, class_id: int):
    # Inserts required data for the new student into the class_user table (id of
    # class added to and id of student being
    # added)
    sql: str = 'INSERT INTO class_user(class_id, student_id) VALUES(?, ?);'
    c.execute(sql, (class_id, student_id))
    # Inserts required data for the new student into the question_results table by
    # finding all questions within each
    # homework already assigned to the class and creating an entry in the table for
    # each with (student_id, id of
    # question, 0 as the number of attempts, False as the question completion sta-
    # tus)
    sql: str = 'INSERT INTO question_results(user_id, question_id, attempts, cor-
    rect) VALUES (?, ?, ?, ?)'
    for each_homework in get_homework_of_class(class_id):
        for each_question in get_questions_of_homework(each_homework[0]):
            c.execute(sql, (student_id, each_question[0], 0, 'F'))
    conn.commit()
```

4.5.10.2 Testing

4.5.10.2.1 SQL

The screenshots illustrate the process of inserting a new record into the `class_user` table using the DB Browser for SQLite interface.

Screenshot 1: The application window shows the `class_user` table with one existing record: `id = 78, student_id = 12`. The "Edit Database Cell" dialog is open over the second row of the table, with the "Mode: Text" dropdown set to "Text". The "Import" and "Export" buttons are visible at the top of the dialog.

id	class_id	student_id
=78		=12

Screenshot 2: A SQL query window titled "SQL 1" contains the command `INSERT INTO class_user(class_id, student_id) VALUES(78, 12);`. The query has been executed successfully, as indicated by the message "Query executed successfully: INSERT INTO class_user(class_id, student_id) VALUES(78, 12); (took 3ms, 1 rows affected)".

name	due_date
1 name1864	2014-3-9

Screenshot 3: The application window now displays two records in the `class_user` table: `id = 78, student_id = 12` and the newly inserted record `id = 10013, class_id = 78, student_id = 12`.

id	class_id	student_id
=78		=12
10013	78	12

Correctly inserts an entry into class_user linking class 78 to student 12

The screenshots show the following sequence of operations:

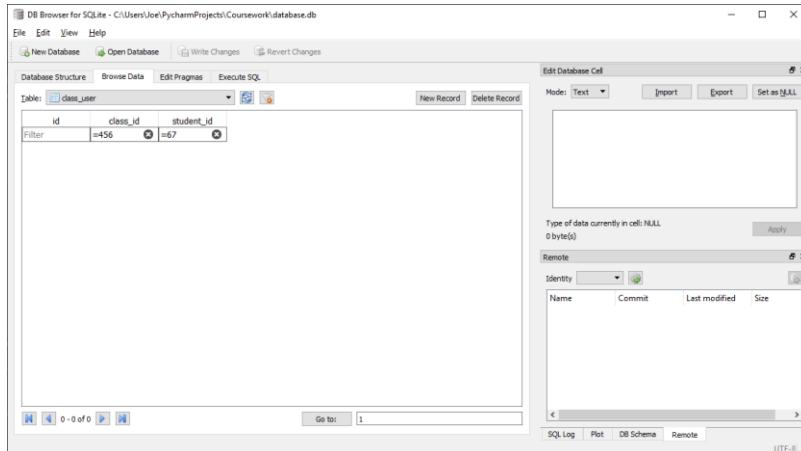
- Screenshot 1:** Shows the 'question_results' table in the database structure. A new record is being inserted with id=87, user_id=12, question_id=45, attempts=0, and correct=F.
- Screenshot 2:** Shows the SQL log window with the executed query: `INSERT INTO question_results(user_id, question_id, attempts, correct) VALUES (87, 45, 0, 'F')`. The message "Query executed successfully; INSERT INTO question_results(user_id, question_id, attempts, correct) VALUES (87, 45, 0, 'F') (took 0ms, 1 rows affected)" is displayed.
- Screenshot 3:** Shows the updated 'question_results' table with the new record (id=1004350, user_id=87, question_id=45, attempts=0, correct=F).

Correctly inserts an entry into question_results linking student 87 to question 45 with attempts and correct set to desired values

4.5.10.2.2 Test code

```
if __name__ == '__main__':
    uid: int = input("Enter user id: ")
    cid: int = input("Enter class id: ")
    add_student_to_class(uid, cid)
```

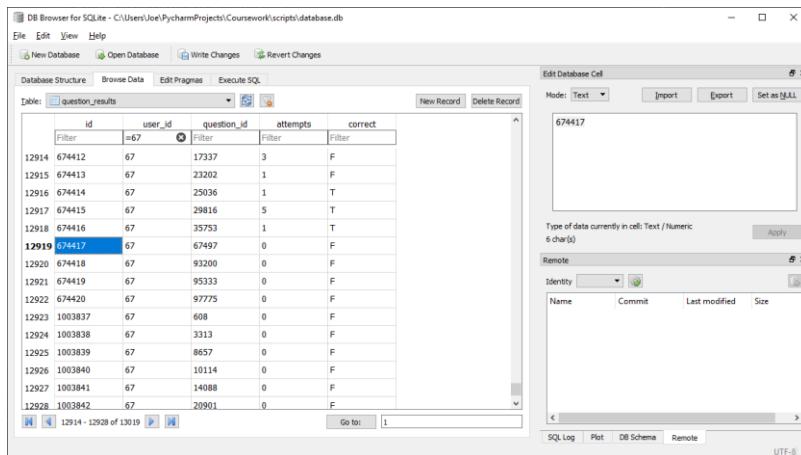
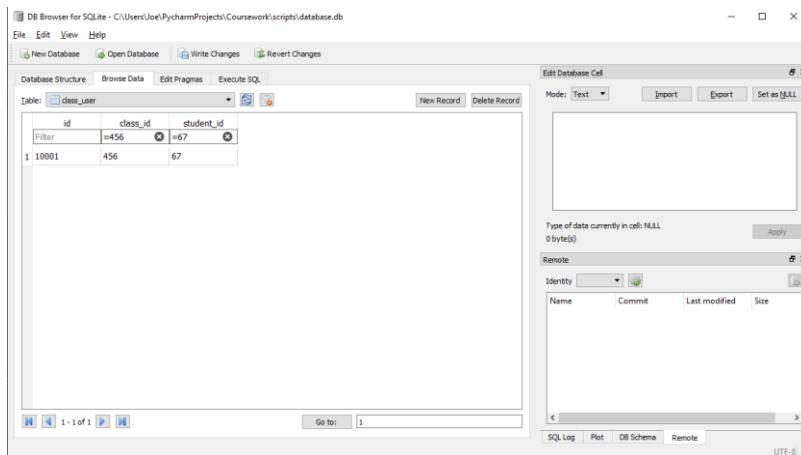
4.5.10.2.3 With valid data:



Enter user id: 67

Enter class id: 456

Process finished with exit code 0



4.5.10.2.4 With Invalid data:

N/A, any errors should be addressed before calling function by verifying student is not already in class before calling function

4.5.10.2.5 Evaluation

Function correctly inserts user-class relationship into class_user table and inserts all required results into question_results table

4.5.11 Function to check if a given student is in a class

4.5.11.1 Final Code

```
def check_student_in_class(student_id: int, class_id: int) -> bool:  
    # Finds the number of occurrences in the class_user table of the inputted stu-  
    # dent id in the same record as the  
    # given class_id. If 0 returned, relationship is not in DB (student not in  
    # class)  
    sql: str = 'SELECT COUNT(1) FROM class_user WHERE class_id = ? AND student_id = ?;  
    c.execute(sql, (class_id, student_id))  
    if c.fetchall()[0][0] > 0:  
        return True  
    else:  
        return False
```

4.5.11.2 Testing

4.5.11.2.1 SQL

4.5.11.2.1.1 User in class

The screenshot shows the DB Browser for SQLite interface. On the left, there is a table view for the 'class_user' table with three columns: 'id', 'class_id', and 'student_id'. A single row is selected with values 1, 325, and 4 respectively. On the right, there is an 'Edit Database Cell' dialog box and a 'Remote' connection status window.

The screenshot shows the DB Browser for SQLite interface. On the left, there is a SQL editor window containing the query: 'SELECT COUNT(1) FROM class_user WHERE class_id = 325 AND student_id = 4;'. The result set shows a single row with a value of 1. On the right, there is an 'Edit Database Cell' dialog box and a 'Remote' connection status window.

Correctly identifies that user 4 is in class 325

4.5.11.2.1.2 User not in class

The screenshot shows the DB Browser for SQLite interface. The main window displays a table named 'class_user' with three columns: 'id', 'class_id', and 'student_id'. A filter is applied with 'Filter' set to 'class_id = 325' and 'student_id = 67'. The result set contains one row with values 1, 325, and 67 respectively. The bottom status bar indicates '0 - 0 of 0' and 'Go to: 1'.

The screenshot shows the DB Browser for SQLite interface. The main window has an 'SQL' tab active with the following query: 'SELECT COUNT(1) FROM class_user WHERE class_id = 325 AND student_id = 67;'. The result pane shows the output: 'COUNT(1)' followed by '0'. Below the result, a message states '1 rows returned in 2ms from: SELECT COUNT(1) FROM class_user WHERE class_id = 325 AND student_id = 67;'.

Correctly identifies that user 67 is not in class 325

4.5.11.2.2 Test code

```
if __name__ == '__main__':
    uid: int = input("Enter user id: ")
    cid: int = input("Enter class id: ")
    add_student_to_class(uid, cid)
```

4.5.11.2.3 With valid data:

4.5.11.2.3.1 User in class

The screenshot shows the DB Browser for SQLite interface. The main window displays a table named 'class_user' with three columns: 'id', 'class_id', and 'student_id'. A single row is selected, showing values 1, 987, and 20 respectively. To the right of the table, an 'Edit Database Cell' dialog is open, specifically for the 'student_id' column of the selected row. The dialog indicates that the current value is NULL. Below the table, a status bar shows '1 - 1 of 1' and 'Go to: 1'. At the bottom of the interface, there are tabs for 'SQL Log', 'Plot', 'DB Schema', 'Remote', and 'UTF-8'.

Enter user id: 20

Enter class id: 987

True

Process finished with exit code 0

4.5.11.2.3.2 User not in class

The screenshot shows the DB Browser for SQLite interface. The main window displays a table named 'class_user' with three columns: 'id', 'class_id', and 'student_id'. A single row is selected, showing values 1, 987, and 45 respectively. To the right of the table, an 'Edit Database Cell' dialog is open, specifically for the 'student_id' column of the selected row. The dialog indicates that the current value is NULL. Below the table, a status bar shows '1 - 1 of 1' and 'Go to: 1'. At the bottom of the interface, there are tabs for 'SQL Log', 'Plot', 'DB Schema', 'Remote', and 'UTF-8'.

Enter user id: 45

Enter class id: 987

False

Process finished with exit code 0

4.5.11.2.4 With Invalid data:

N/A, class_id should be verified to exist and student_id verified to be an integer before calling function

4.5.11.2.5 Evaluation

Function correctly returns True if the given user is in the given class and False otherwise

4.5.12 Function to remove a student from a class

4.5.12.1 Final Code

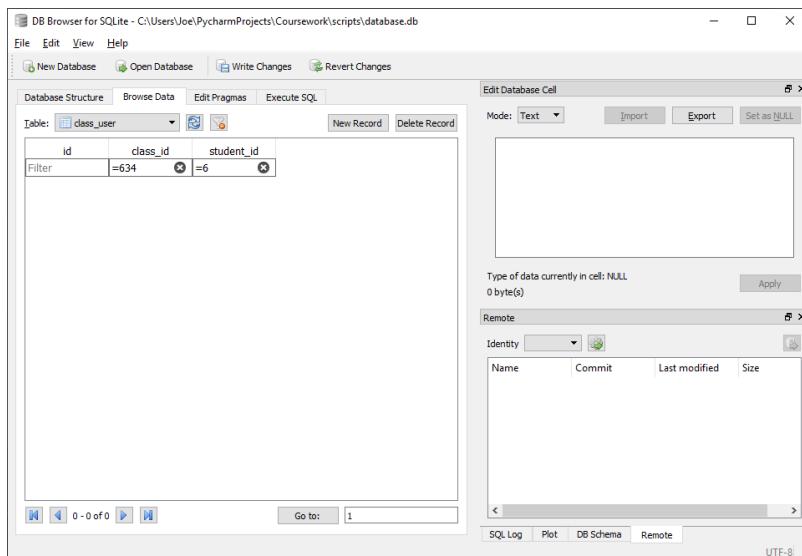
```
def remove_student_from_class(student_id: int, class_id: int):
    # Deletes any entry in class_user which links the given user to the given class
    sql: str = 'DELETE FROM class_user WHERE student_id = ? AND class_id = ?;'
    c.execute(sql, (student_id, class_id))
    conn.commit()
```

4.5.12.2 Testing

4.5.12.2.1 SQL

The screenshot shows the 'DB Browser for SQLite' interface. On the left, the 'Database Structure' tab is selected, displaying the 'class_user' table with three columns: Id, class_id, and student_id. A single row is visible with values 7164, 634, and 6 respectively. A filter is applied to both the class_id and student_id columns. On the right, the 'Edit Database Cell' dialog is open, focusing on the student_id cell. The mode is set to 'Text'. Below the input field, it says 'Type of data currently in cell: NULL 0 byte(s)'. There are 'Import', 'Export', and 'Set as NULL' buttons. At the bottom of the dialog, there's an 'Apply' button. The status bar at the bottom right shows 'UTF-8'.

The screenshot shows the 'DB Browser for SQLite' interface again. On the left, the 'Execute SQL' tab is selected, containing the SQL command: 'DELETE FROM class_user WHERE student_id = 6 AND class_id = 634;'. Below the command, the results of a COUNT(*) query are shown: 'COUNT(1)' followed by '1 0'. At the bottom of the results pane, a message states 'Query executed successfully: DELETE FROM class_user WHERE student_id = 6 AND class_id = 634; (took 2ms, 1 rows affected)'. The status bar at the bottom right shows 'UTF-8'.

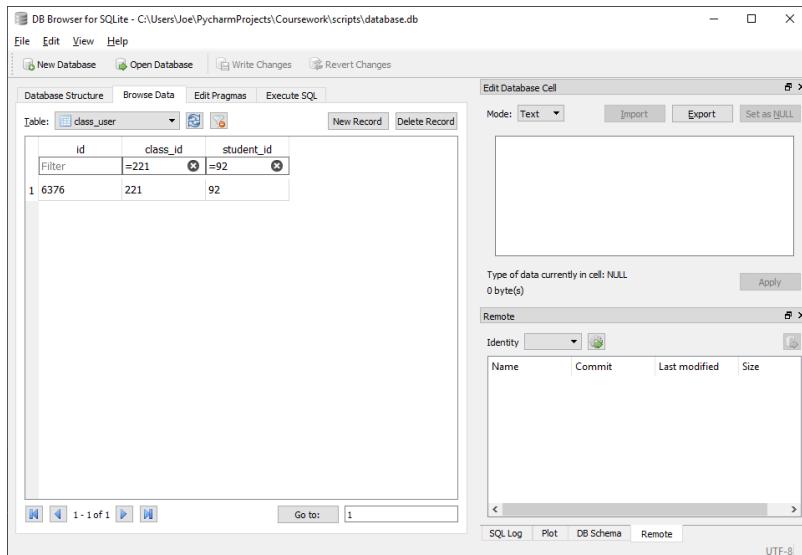


Correctly removes student 6 from class 643

4.5.12.2.2 Test code

```
if __name__ == '__main__':
    uid: int = input("Enter user id: ")
    cid: int = input("Enter class id: ")
    add_student_to_class(uid, cid)
```

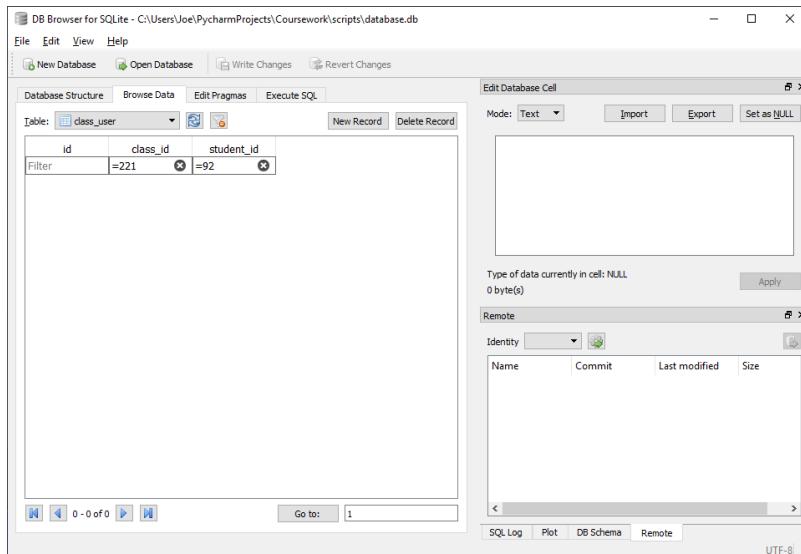
4.5.12.2.3 With valid data:



Enter user id: 92

Enter class id: 221

Process finished with exit code 0



4.5.12.2.4 With Invalid data:

N/A, class_id should be verified to exist and student_id verified to be in class before calling function

4.5.12.2.5 Evaluation

Function correctly removes entry in class_user linking given student to given class

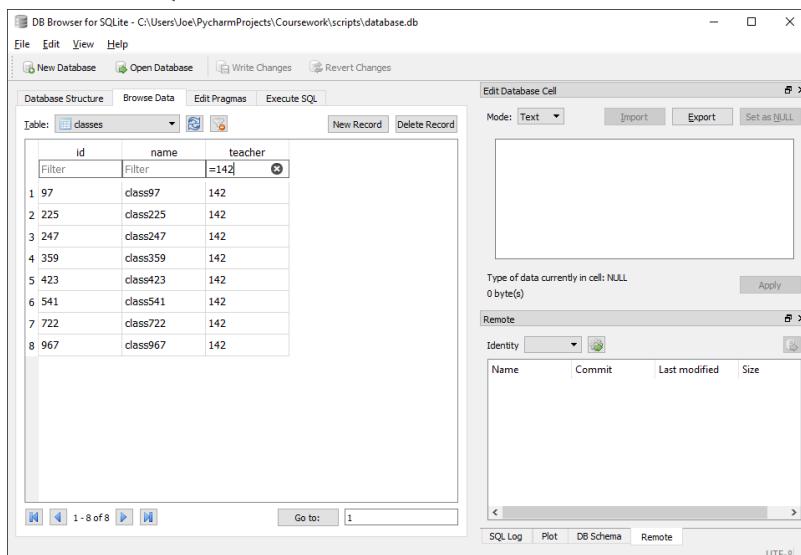
4.5.13 Function to create a new class

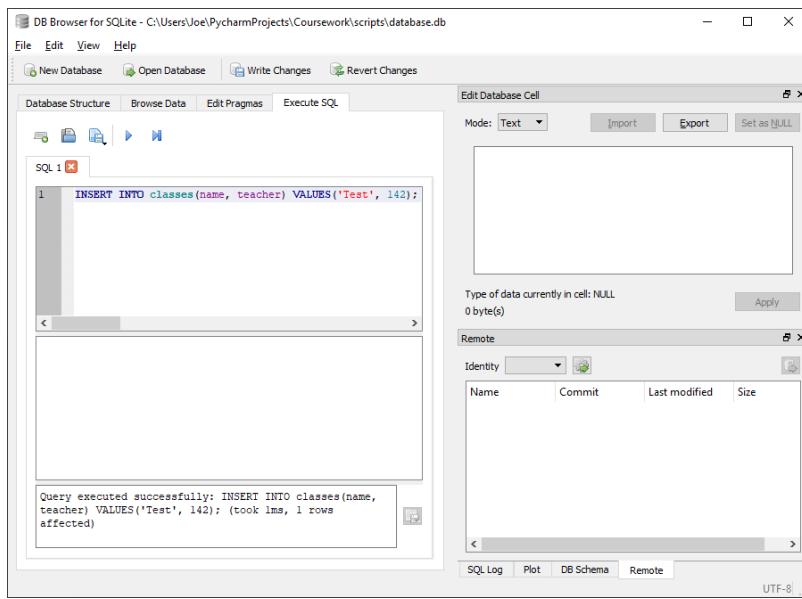
4.5.13.1 Final Code

```
def create_class(teacher_id: int, class_name: str):
    # Inserts into the classes tables a new record with the new class' name and the
    # id of the teacher who created it
    sql: str = 'INSERT INTO classes(name, teacher) VALUES (?, ?);'
    c.execute(sql, (class_name, teacher_id))
    conn.commit()
```

4.5.13.2 Testing

4.5.13.2.1 SQL





The screenshot shows the DB Browser for SQLite interface with the "classes" table selected. The table has three columns: id, name, and teacher. The data is as follows:

	id	name	teacher
1	97	class97	142
2	225	class225	142
3	247	class247	142
4	359	class359	142
5	423	class423	142
6	541	class541	142
7	722	class722	142
8	967	class967	142
9	1001	Test	142

Entry in classes for new class with appropriate name (Test) and teacher id (142) successfully added

4.5.13.2.2 Test code

```
if __name__ == '__main__':
    name = input("Enter class name: ")
    uid: int = input("Enter teacher id: ")
    create_class(uid, name)
```

4.5.13.2.3 With valid data:

The screenshot shows the DB Browser for SQLite interface. On the left, a table named 'classes' is displayed with columns 'id', 'name', and 'teacher'. A filter is applied for 'teacher' = 187, showing 12 records. On the right, an 'Edit Database Cell' dialog is open over the last row of the table, which has been selected. The dialog has a 'Mode: Text' dropdown set to 'Text'. Below it is a large empty text area. At the bottom of the dialog, there is a message: 'Type of data currently in cell: NULL 0 byte(s)' with an 'Apply' button. A 'Remote' tab is also visible at the bottom of the dialog. The main window footer includes tabs for 'SQL Log', 'Plot', 'DB Schema', and 'Remote', along with a 'UTF-8' encoding indicator.

Enter class name: New Class

Enter teacher id: 187

Process finished with exit code 0

This screenshot is identical to the one above, showing the DB Browser for SQLite interface. The 'classes' table now has 13 rows, with the new record 'New Class' having an 'id' of 1001 and a 'teacher' id of 187. The 'Edit Database Cell' dialog is still open over the last row, and the process has completed successfully as indicated by the 'Process finished with exit code 0' message.

4.5.13.2.4 With Invalid data:

N/A, teacher_id should be verified to exist before calling function

4.5.13.2.5 Evaluation

Function correctly inserts new class into database with the appropriate class name and teacher id

4.5.14 Function to remove a class

4.5.14.1 Final Code

```
def remove_class(class_id: int):
    # Deletes all entries in each table which references the existence of the given
    class
    # Deletes any entry in class_user relating to the given class_id
    sql: str = 'DELETE FROM class_user WHERE class_id = ?'
    c.execute(sql, (class_id,))
    # Deletes any entry in class_homework relating to the given class_id
    sql: str = 'DELETE FROM class_homework WHERE class_id = ?'
    c.execute(sql, (class_id,))
    # Deletes any entry in classes relating to the given class_id
    sql: str = 'DELETE FROM classes WHERE id = ?; '
    c.execute(sql, (class_id,))
    conn.commit()
```

4.5.14.2 Testing

4.5.14.2.1 SQL

4.5.14.2.1.1 Remove class from class_user

The screenshot shows the DB Browser for SQLite interface. On the left, there is a table named 'class_user' with columns 'id', 'class_id', and 'student_id'. The data grid contains four rows with values: (1, 678, 78), (2, 678, 12), (3, 678, 1), and (4, 678, 66). A filter bar at the top of the table view shows 'Filter' and '=678'. On the right, there is an 'Edit Database Cell' dialog box. It has a 'Mode: Text' dropdown set to 'Text', and a large text area below it. Below the text area, it says 'Type of data currently in cell: NULL' and '0 byte(s)'. There is an 'Apply' button. Below the dialog, there is a 'Remote' panel with a table header 'Name Commit Last modified Size'. At the bottom of the interface, there are tabs for 'SQL Log', 'Plot', 'DB Schema', and 'Remote', with 'Remote' being the active tab. The status bar at the bottom right shows 'UTF-8'.

The screenshot shows the DB Browser for SQLite interface. In the main window, there is a SQL editor tab titled "SQL 1" containing the following SQL command:

```
1  DELETE FROM class_user WHERE class_id = 678
```

Below the SQL editor, a message box displays the result of the query: "Query executed successfully: DELETE FROM class_user WHERE class_id = 678 (took 2ms, 4 rows affected)".

The screenshot shows the DB Browser for SQLite interface. In the main window, there is a table viewer for the "class_user" table. The table has three columns: "id", "class_id", and "student_id". A filter is applied to the "class_id" column with the value "=678". The results show 0 rows.

Successfully removes all references to given class (678) from class_user table

4.5.14.2.1.2 Remove class from class_homework

The image consists of three vertically stacked screenshots of the DB Browser for SQLite application.

Screenshot 1: Shows the 'Database Structure' tab selected, displaying the 'class_homework' table. The table has four columns: id, class_id, homework_id, and due_date. A filter is applied where class_id = 678. The results show six rows of data.

	id	class_id	homework_id	due_date
1	1857	678	1857	2010-8-13
2	6815	678	6815	2019-6-3
3	8043	678	8043	2014-3-11
4	8547	678	8547	2014-7-25
5	9017	678	9017	2016-1-7
6	9191	678	9191	2011-8-25

Screenshot 2: Shows the 'Execute SQL' tab selected. An SQL command is entered: `DELETE FROM class_homework WHERE class_id = 678`. The message 'Query executed successfully: DELETE FROM class_homework WHERE class_id = 678 (took 0ms, 6 rows affected)' is displayed.

```
SQL 1
1   DELETE FROM class_homework WHERE class_id = 678

Query executed successfully: DELETE FROM class_homework
WHERE class_id = 678 (took 0ms, 6 rows affected)
```

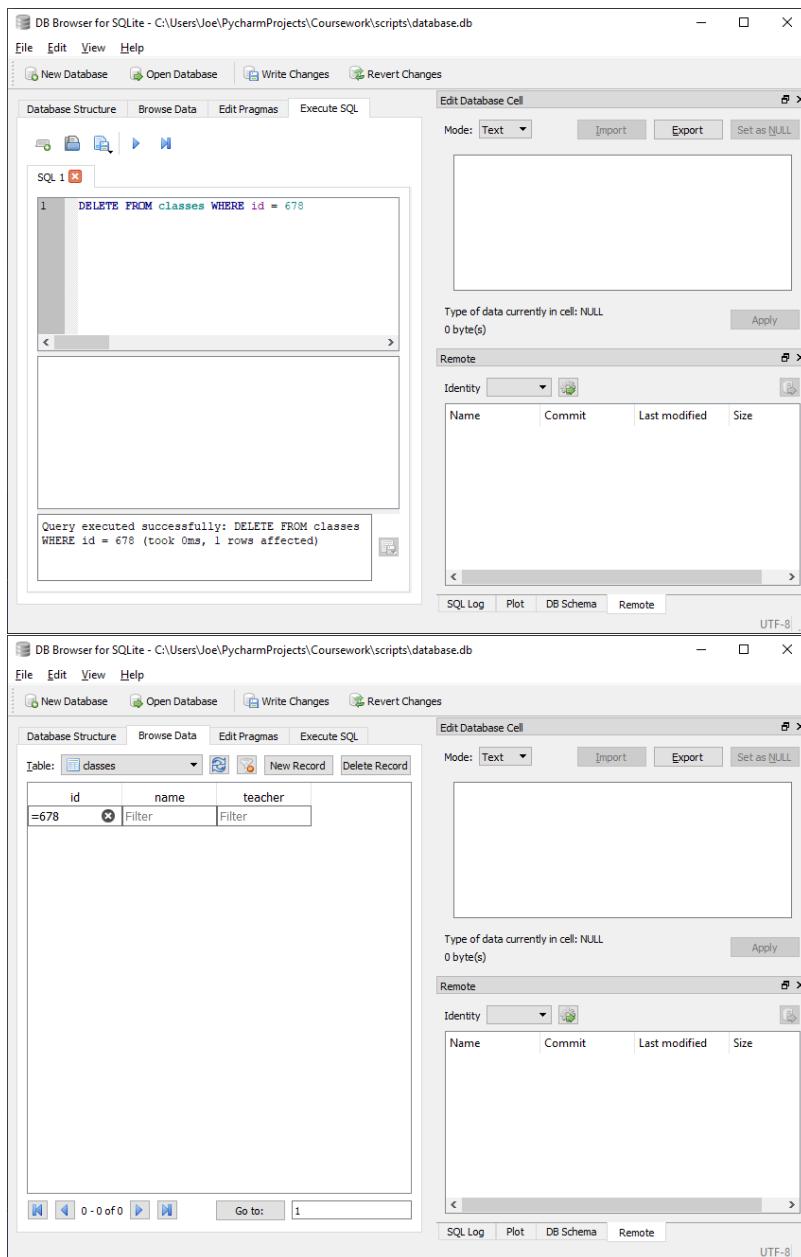
Screenshot 3: Shows the 'Database Structure' tab again. The filter has been removed, resulting in 0 rows of data.

	id	class_id	homework_id	due_date
--	----	----------	-------------	----------

Successfully removes all references to given class (678) from class_homework table

4.5.14.2.1.3 Remove class from classes

The screenshot shows the DB Browser for SQLite interface. The main window displays the 'classes' table with three columns: id, name, and teacher. A single record is selected: id=678, name=class678, and teacher=171. An 'Edit Database Cell' dialog is open over the table, specifically for the 'teacher' column of the selected row. The dialog has a large text input field containing '171'. Below the input field, it says 'Type of data currently in cell: NULL 0 byte(s)' and has an 'Apply' button. At the bottom of the dialog, there's a 'Remote' tab and a status bar indicating 'UTF-8'. The top menu bar includes File, Edit, View, Help, New Database, Open Database, Write Changes, and Revert Changes.



Successfully removes all references to given class (678) from classes table

4.5.14.2.2 Test code

```
if __name__ == '__main__':
    cid: int = input("Enter class id: ")
    remove_class(cid)
```

4.5.14.2.3 With valid data:

DB Browser for SQLite - C:\Users\Joe\PycharmProjects\Coursework\scripts\database.db

Table: class_user

	id	class_id	student_id
1	256	45	26
2	2646	45	78
3	3260	45	56
4	3493	45	100
5	4674	45	14
6	5668	45	47
7	7487	45	81
8	7667	45	100
9	8634	45	59

1 - 9 of 9 Go to: 1

Edit Database Cell

Type of data currently in cell: NULL
0 byte(s)

Remote

Identity

Name Commit Last modified Size

SQL Log Plot DB Schema Remote

UTF-8

DB Browser for SQLite - C:\Users\Joe\PycharmProjects\Coursework\scripts\database.db

Table: class_homework

	id	class_id	homework_id	due_date
1	498	45	498	2014-8-3
2	1281	45	1281	2015-4-24
3	3058	45	3058	2015-1-19
4	3527	45	3527	2014-4-27
5	3764	45	3764	2010-5-3
6	4164	45	4164	2017-11-23
7	5858	45	5858	2010-10-25
8	6278	45	6278	2012-4-13
9	7497	45	7497	2010-11-20
10	7856	45	7856	2014-11-12

1 - 10 of 10 Go to: 1

Edit Database Cell

Type of data currently in cell: NULL
0 byte(s)

Remote

Identity

Name Commit Last modified Size

SQL Log Plot DB Schema Remote

UTF-8

The screenshot shows the DB Browser for SQLite interface. The main window displays the 'classes' table with the following data:

	id	name	teacher
1	45	class45	140

The 'Edit Database Cell' dialog is open over the first row of the table. The 'Mode' dropdown is set to 'Text'. The cell content is currently 'NULL'. Below the cell, it says 'Type of data currently in cell: NULL' and '0 byte(s)'. There is an 'Apply' button. The 'Remote' panel is visible at the bottom right.

Enter class id: 45

Process finished with exit code 0

The screenshot shows the DB Browser for SQLite interface. The main window displays the 'class_user' table with the following data:

	id	class_id	student_id
1		45	

The 'Edit Database Cell' dialog is open over the first row of the table. The 'Mode' dropdown is set to 'Text'. The cell content is currently 'NULL'. Below the cell, it says 'Type of data currently in cell: NULL' and '0 byte(s)'. There is an 'Apply' button. The 'Remote' panel is visible at the bottom right.

The image contains two side-by-side screenshots of the DB Browser for SQLite application.

Screenshot 1 (Top): Shows the 'class_homework' table. The table has columns: id, class_id, homework_id, and due_date. A filter is applied where id = 45. The 'Edit Database Cell' dialog is open over the first row of the table. The 'Mode' dropdown is set to 'Text'. The cell content is empty, and the status message says 'Type of data currently in cell: NULL 0 byte(s)'. An 'Apply' button is visible. Below the table, the status bar shows '0 - 0 of 0' and 'Go to: 1'.

Screenshot 2 (Bottom): Shows the 'classes' table. The table has columns: id, name, and teacher. A filter is applied where id = 45. The 'Edit Database Cell' dialog is open over the first row of the table. The 'Mode' dropdown is set to 'Text'. The cell content is empty, and the status message says 'Type of data currently in cell: NULL 0 byte(s)'. An 'Apply' button is visible. Below the table, the status bar shows '0 - 0 of 0' and 'Go to: 1'.

4.5.14.2.4 With Invalid data:

N/A, class_id should be verified to exist before calling function

4.5.14.2.5 Evaluation

Function correctly removes all references to the given class in all tables where it may occur (class_user, class_homework, classes)

4.5.15 Function to get results for a homework for all students in a class

4.5.15.1 *Issues during development*

Initially the below code was implemented:

```

def get_results_of_homework(class_id: int, homework_id: int) -> list:
    # Gets list of all students in the given class so results for each student in
    # the class can be found later
    students: list = get_students_of_class(class_id)
    scores: list = []
    # For every student in the class, the following data is found and appended as a
    # tuple of the format
    # [first_name, last_name, score, percentage, attempts] to the list 'scores':
    for student in students:
        user_id: int = student[0]
        # Takes the user id for the current student, the id of the current class
        # and the id of the current homework,
        # finds all question results for the given user for the given homework
        # within the given class, and returns the
        # question correctly answered status and number of results for each relevant
        # result from the question_results
        # table by joining all relevant tables appropriately. Data is stored in tuple
        # 'results'
        sql: str = 'SELECT question_results.correct, question_results.attempts ' \
                   'FROM users INNER JOIN ((class_homework ' \
                   'INNER JOIN homework_questions ON class_homework.homework_id = ' \
                   'homework_questions.homework_id) ' \
                   'INNER JOIN question_results ON homework_questions.question_id = ' \
                   'question_results.question_id) ' \
                   'ON question_results.user_id = users.id ' \
                   'WHERE class_homework.class_id = ? AND class_homework.home- \
                   work_id = ? AND users.id = ?'
        c.execute(sql, (class_id, homework_id, user_id))
        results: tuple = c.fetchall()
        # Gets the first and last names for the given user
        first_name: str = db_scripts.get_first_name(user_id)
        last_name: str = db_scripts.get_last_name(user_id)
        # Calculates the score as a percentage for the current user by taking the
        # sum of the number of questions with
        # True as the correct status and dividing by the question count then multiplying
        # by 100. The number of attempts
        # is found by iterating through each question score and setting the number
        # of attempts variable to the number of
        # attempts for that question if it is greater than the currently stored
        # value (finds the maximum number of
        # attempts taken across all questions for that user)
        score: int = 0
        attempts: int = 0
        question_count: int = 0
        for question in results:
            question_count += 1
            if question[0] == 'T':
                score += 1
            if question[1] > attempts:
                attempts = question[1]
        percentage: str = round(score / question_count * 100)
        # Appends entry for given student to scores list as a tuple of the appropriate
        # format
        scores.append([first_name, last_name, score, percentage, attempts])
    # Returns array of score data
    return scores

```

This would result in an error occurring if for any reason the question count took a value of 0 (if a given homework has no questions or the given homework was never set for the given user and no results have therefore been stored).

Enter class id: 56

Enter homework id: 43

Traceback (most recent call last):

```
File "C:/Users/Joe/PycharmProjects/Coursework/scripts/ui_scripts.py", line 273, in <module>
```

```
    print(get_results_of_homework(cid, hid))
```

```
File "C:/Users/Joe/PycharmProjects/Coursework/scripts/ui_scripts.py", line 197, in
get_results_of_homework
```

```
    percentage: str = round(score / question_count * 100)
```

```
ZeroDivisionError: division by zero
```

Process finished with exit code 1

This was fixed by placing the calculation of the percentage within a try statement and defining that if a division by zero error occurs then the score to be returned is “N/A” (i.e. the score being requested is invalid).

```
# Calculates the score as a percentage for the current user by taking the sum of
# the number of questions with
# True as the correct status and dividing by the question count then multiplying by
# 100. The number of attempts
# is found by iterating through each question score and setting the number of at-
# tempts variable to the number of
# attempts for that question if it is greater than the currently stored value
# (finds the maximum number of
# attempts taken across all questions for that user). If for any reason the number
# of questions is 0 (i.e.
# homework has no questions), a division by zero error occurs when calculating
# score. As a score out of 0 cannot
# exist anyway, this is fixed by catching a division by zero error and setting the
# score to 'N/A'
score: int = 0
attempts: int = 0
question_count: int = 0
for question in results:
    question_count += 1
    if question[0] == 'T':
        score += 1
    if question[1] > attempts:
        attempts = question[1]
try:
    percentage: str = round(score / question_count * 100)
except ZeroDivisionError:
    percentage: str = "N/A"
```

4.5.15.2 Final Code

```

def get_results_of_homework(class_id: int, homework_id: int) -> list:
    # Gets list of all students in the given class so results for each student in
    the class can be found later
    students: list = get_students_of_class(class_id)
    scores: list = []
    # For every student in the class, the following data is found and appended as a
    tuple of the format
    # [first name, last name, score, percentage, attempts] to the list 'scores':
    for student in students:
        user_id: int = student[0]
        # Takes the user id for the current student, the id of the current class
        and the id of the current homework,
        # finds all question results for the given user for the given homework
        within the given class, and returns the
        # question correctly answered status and number of results for each relevant
        result from the question_results
        # table by joining all relevant tables appropriately. Data is stored in tuple
        'results'
        sql: str = 'SELECT question_results.correct, question_results.attempts ' \
                   'FROM users INNER JOIN ((class_homework ' \
                   'INNER JOIN homework_questions ON class_homework.homework_id = ' \
                   'homework_questions.homework_id) ' \
                   'INNER JOIN question_results ON homework_questions.question_id = ' \
                   'question_results.question_id) ' \
                   'ON question_results.user_id = users.id ' \
                   'WHERE class_homework.class_id = ? AND class_homework.home- ' \
                   'work_id = ? AND users.id = ?'
        c.execute(sql, (class_id, homework_id, user_id))
        results: tuple = c.fetchall()
        # Gets the first and last names for the given user
        first_name: str = db_scripts.get_first_name(user_id)
        last_name: str = db_scripts.get_last_name(user_id)
        # Calculates the score as a percentage for the current user by taking the
        sum of the number of questions with
        # True as the correct status and dividing by the question count then multiplying
        by 100. The number of attempts
        # is found by iterating through each question score and setting the number
        of attempts variable to the number of
        # attempts for that question if it is greater than the currently stored
        value (finds the maximum number of
        # attempts taken across all questions for that user). If for any reason the
        number of questions is 0 (i.e.
        # homework has no questions), a division by zero error occurs when calculating
        score. As a score out of 0 cannot
        # exist anyway, this is fixed by catching a division by zero error and setting
        the score to 'N/A'
        score: int = 0
        attempts: int = 0
        question_count: int = 0
        for question in results:
            question_count += 1
            if question[0] == 'T':
                score += 1
            if question[1] > attempts:
                attempts = question[1]
        try:
            percentage: str = round(score / question_count * 100)
        except ZeroDivisionError:
            percentage: str = "N/A"
        # Appends entry for given student to scores list as a tuple of the appropriate
        format
        scores.append([first_name, last_name, score, percentage, attempts])
    # Returns array of score data
    return scores

```

4.5.15.3 Testing

4.5.15.3.1 SQL

The screenshot shows two instances of DB Browser for SQLite. Both instances have the following interface elements:

- Top menu bar: File, Edit, View, Help.
- Toolbar: New Database, Open Database, Write Changes, Revert Changes.
- Database structure tab.
- Browse Data tab (selected).
- Edit Pragmas tab.
- Execute SQL tab.
- Table list: class_user and class_homework.
- Table view area:

 - class_user table:**

id	class_id	student_id
1	127	59
 - class_homework table:**

id	class_id	homework_id	due_date
1	127	7280	2014-5-18

- Edit Database Cell dialog box (open on the right):
 - Mode: Text.
 - Text input field: (empty).
 - Buttons: Import, Export, Set as NULL.
 - Status message: Type of data currently in cell: NULL.
 - Identity dropdown: (empty).
 - Buttons: Name, Commit, Last modified, Size.
 - Buttons: Apply, Cancel.
- Bottom status bar: Go to, 1-1 of 1, SQL Log, Plot, DB Schema, Remote, UTF-8.

DB Browser for SQLite - C:\Users\Joe\PycharmProjects\Coursework\scripts\database.db

Table: homework_questions

	id	homework_id	question_id
1	4200	7280	4200
2	14081	7280	14081
3	26908	7280	26908
4	32505	7280	32505
5	45909	7280	45909
6	48808	7280	48808
7	53067	7280	53067
8	63482	7280	63482
9	67799	7280	67799
10	68854	7280	68854
11	69297	7280	69297
12	72756	7280	72756
13	73138	7280	73138

DB Browser for SQLite - C:\Users\Joe\PycharmProjects\Coursework\scripts\database.db

SQL: 1

```

1 SELECT question_results.correct, question_results.attempts FROM users INNER JOIN ((class_homework INNER JOIN homework_questions ON class_homework.homework_id = homework_questions.homework_id)
2 INNER JOIN question_results ON homework_questions.question_id = question_results.question_id) ON question_results.user_id = users.id
3 WHERE class_homework.class_id = 127 AND class_homework.homework_id = 7280 AND users.id = 59

```

Table: question_results

	correct	attempts
1	T	3
2	F	5
3	F	3
4	F	5
5	F	0
6	T	1
7	F	0
8	T	5
9	F	5
10	T	2
11	T	4

13 rows returned in 35ms from: SELECT question_results.correct, question_results.attempts FROM users INNER JOIN ((class_homework INNER JOIN homework_questions ON class_homework.homework_id = homework_questions.homework_id)
INNER JOIN question_results ON homework_questions.question_id = question_results.question_id) ON question_results.user_id = users.id
WHERE class_homework.class_id = 127 AND class_homework.homework_id = 7280 AND users.id = 59

Correctly finds the correct status and number of attempts for each question in the given homework (7280) in the given class (127) for the given user (59)

4.5.15.3.2 Test code

```

if __name__ == '__main__':
    cid: int = input("Enter class id: ")
    hid: int = input("Enter homework id: ")
    print(get_results_of_homework(cid, hid))

```

4.5.15.3.3 With valid data:

The screenshot shows two instances of DB Browser for SQLite. The top instance displays the 'class_homework' table with the following data:

	id	class_id	homework_id	due_date
1	7280	127	7280	2014-5-18

The bottom instance displays the 'class_user' table with the following data:

	id	class_id	student_id
1	54	127	61
2	556	127	23
3	1300	127	11
4	2208	127	80
5	2628	127	27
6	3119	127	35
7	4045	127	25
8	4339	127	33
9	5534	127	29
10	5950	127	38
11	7946	127	16
12	8513	127	82
13	9138	127	8
14	9898	127	59

Enter class id: 127

Enter homework id: 7280

```
[['first61', 'last61', 6, 46, 5], ['first23', 'last23', 5, 38, 5], ['first11', 'last11', 7, 54, 4], ['first80', 'last80', 5, 38, 4], ['first27', 'last27', 7, 54, 5], ['first35', 'last35', 2, 15, 5], ['first25', 'last25', 3, 23, 5], ['first33', 'last33', 3, 23, 5], ['first29', 'last29', 5, 38, 5], ['first38', 'last38', 7, 54, 5], ['first16', 'last16', 3, 23, 5], ['first82', 'last82', 4, 31, 5], ['first8', 'last8', 4, 31, 5], ['first59', 'last59', 6, 46, 5]]
```

Process finished with exit code 0

Correctly outputs correct results for the given class and homework as an array of tuples of the format [first name, last name, score for the given homework, percentage for the given homework, number of attempts for the given homework and number of attempts at the given homework]

4.5.15.3.4 With Invalid data:

Homework with no results stored:

The screenshot shows the DB Browser for SQLite interface. On the left, there's a table named 'class_homework' with columns: id, class_id, homework_id, and due_date. A single row is selected with id=56, class_id=43, homework_id=43, and due_date set to null. An 'Edit Database Cell' dialog is open over this row, showing 'Mode: Text' and the value 'NULL'. Below it, a 'Remote' dialog shows connection details. The bottom status bar indicates 'UTF-8' encoding.

Class was not set given homework so no question results should be stored

Enter class id: 876

Enter homework id: 23

```
[['first76', 'last76', 0, 'N/A', 0], ['first63', 'last63', 0, 'N/A', 0], ['first27', 'last27', 0, 'N/A', 0],  
['first56', 'last56', 0, 'N/A', 0], ['first91', 'last91', 0, 'N/A', 0], ['first25', 'last25', 0, 'N/A', 0], ['first4',  
'last4', 0, 'N/A', 0], ['first39', 'last39', 0, 'N/A', 0]]
```

Process finished with exit code 0

Percentages for each student are correctly set to 'N/A'

Any errors relating to the use of invalid class_id and/or homework_id should be addressed before calling function

4.5.15.3.5 Evaluation

Function correctly finds all students in the given class, for each student finds their [first name, last name, score for the given homework, percentage for the given homework, number of attempts for the given homework and number of attempts at the given homework]. This data is then correctly outputted as an array of tuples containing the data for each student as just defined.

In the event of a no questions being present for any reason which would result in a division by zero error usually, the program correctly sets percentage to 'N/A'.

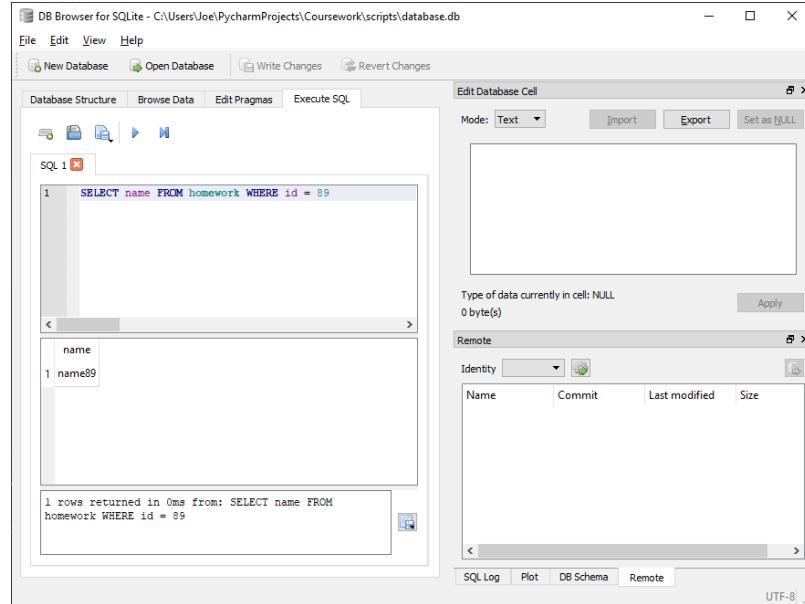
4.5.16 Function to get the name of a homework

4.5.16.1 Final Code

```
def get_name_of_homework(homework_id: int):
    # Returns homework name from database for given homework id
    sql: str = 'SELECT name FROM homework WHERE id = ?;'
    c.execute(sql, (homework_id,))
    return c.fetchall()[0][0]
```

4.5.16.2 Testing

4.5.16.2.1 SQL



Correctly finds matching homework name (name89) for given homework id (89)

4.5.16.2.2 Test code

```
if __name__ == '__main__':
    hid: int = input("Enter homework id: ")
    print(get_name_of_homework(hid))
```

4.5.16.2.3 With valid data:

Enter homework id: 348

name348

Process finished with exit code 0

Correctly finds matching homework name for given homework id

4.5.16.2.4 With Invalid data:

N/A, any errors should be addressed before calling function by verifying homework id in advance

4.5.16.2.5 Evaluation

Function correctly finds matching homework name for given homework id

4.5.17 Function to get the all homework scores for a given user in a given class

4.5.17.1 Final Code

```

def get_scores_of_student_in_class(class_id: int, student_id: int):
    # Gets list of all homework in the given class so results for each homework in
    the class can be found later
    homework: list = get_homework_of_class(class_id)
    scores: list = []
    # For every homework in the class, the following data is found and appended as
    a tuple of the format
    # [homework name, score, percentage, attempts] to the list 'scores':
    for homework in homework:
        homework_id = homework[0]
        # Takes the user id for the current student, the id of the current class
        and the id of the current homework,
        # finds all question results for the given user for the given homework
        within the given class, and returns the
        # question correctly answered status and number of attempts for each relevant
        result from the question_results
        # table by joining all relevant tables appropriately. Data is stored in tuple
        'results'
        sql: str = 'SELECT question_results.correct, question_results.attempts ' \
                   'FROM users INNER JOIN ((class_homework ' \
                   'INNER JOIN homework_questions ON class_homework.homework_id = ' \
                   'homework_questions.homework_id) ' \
                   'INNER JOIN question_results ON homework_questions.question_id = ' \
                   'question_results.question_id) ' \
                   'ON question_results.user_id = users.id ' \
                   'WHERE class_homework.class_id = ? AND class_homework.home- \
                   work_id = ? AND users.id = ?'
        c.execute(sql, (class_id, homework_id, student_id))
        results: tuple = c.fetchall()
        # Gets the name of the current homework being analysed
        homework_name: str = get_name_of_homework(homework_id)
        # Calculates the score as a percentage for the current homework by taking
        the sum of the number of questions
        # with True as the correct status and dividing by the question count then
        multiplying by 100. The number of
        # attempts is found by iterating through each question score and setting
        the number of attempts variable to the
        # number of attempts for that question if it is greater than the currently
        stored value (finds the maximum
        # number of attempts taken across all questions for that user). If for any
        reason the number of questions is 0
        # (i.e. homework has no questions), a division by zero error occurs when
        calculating score. As a score out of 0
        # cannot exist anyway, this is fixed by catching a division by zero error
        and setting the score to 'N/A'
        score: int = 0
        attempts: int = 0
        question_count: int = 0
        for question in results:
            question_count += 1
            if question[0] == 'T':
                score += 1
            if question[1] > attempts:
                attempts = question[1]
    try:
        percentage: str = round(score / question_count * 100)
    except ZeroDivisionError:
        percentage: str = "N/A"
    # Appends entry for given homework to scores list as a tuple of the appropriate
    # format
    scores.append([homework_name, score, percentage, attempts])
    # Returns array of score data
    return scores

```

4.5.17.2 Testing

4.5.17.2.1 SQL

DB Browser for SQLite - C:\Users\Joe\PycharmProjects\Coursework\scripts\database.db

File Edit View Help
New Database Open Database Write Changes Revert Changes

Database Structure Browse Data Edit Pragmas Execute SQL

Table class_user

	id	class_id	student_id
1	8880	824	77

Filter: id = 824 student_id = 77

New Record Delete Record

Go to: 1

Edit Database Cell Mode: Text Import Export Set as NULL

Type of data currently in cell: NULL 0 byte(s)

Remote Identity Name Commit Last modified Size

SQL Log Plot DB Schema Remote UTF-8

DB Browser for SQLite - C:\Users\Joe\PycharmProjects\Coursework\scripts\database.db

File Edit View Help
New Database Open Database Write Changes Revert Changes

Database Structure Browse Data Edit Pragmas Execute SQL

Table class_homework

	id	class_id	homework_id	due_date
1	7316	824	7316	2011-6-24

Filter: id = 7316

New Record Delete Record

Go to: 1

Edit Database Cell Mode: Text Import Export Set as NULL

Type of data currently in cell: NULL 0 byte(s)

Remote Identity Name Commit Last modified Size

SQL Log Plot DB Schema Remote UTF-8

DB Browser for SQLite - C:\Users\Joe\PycharmProjects\Coursework\scripts\database.db

File Edit View Help
New Database Open Database Write Changes Revert Changes

Database Structure Browse Data Edit Pragmas Execute SQL

Table homework_questions

	id	homework_id	question_id
1	17547	7316	17547
2	21637	7316	21637
3	29999	7316	29999
4	42896	7316	42896
5	43471	7316	43471
6	43767	7316	43767
7	44597	7316	44597
8	46059	7316	46059
9	60219	7316	60219
10	62364	7316	62364
11	63844	7316	63844
12	83106	7316	83106

Filter: id = 7316

New Record Delete Record

Go to: 1

Edit Database Cell Mode: Text Import Export Set as NULL

Type of data currently in cell: NULL 0 byte(s)

Remote Identity Name Commit Last modified Size

SQL Log Plot DB Schema Remote UTF-8

DB Browser for SQLite - C:\Users\Joe\PycharmProjects\Coursework\scripts\database.db

File Edit View Help
New Database Open Database Write Changes Revert Changes

Database Structure Browse Data Edit Pragmas Execute SQL

SQL 1

```
1 SELECT question_results.correct, question_results.attempts FROM users INNER JOIN ((class_homework INNER JOIN homework_questions ON class_homework.homework_id = homework_questions.homework_id) INNER JOIN question_results ON homework_questions.question_id = question_results.question_id) ON question_results.user_id = users.id
2 WHERE class_homework.class_id = 824 AND class_homework.homework_id = 7316 AND users.id = 77
```

12 rows returned in 406ms from: SELECT question_results.correct, question_results.attempts FROM users INNER JOIN ((class_homework INNER JOIN homework_questions ON class_homework.homework_id = homework_questions.homework_id) INNER JOIN question_results ON homework_questions.question_id = question_results.question_id) ON question_results.user_id = users.id
WHERE class_homework.class_id = 824 AND class_homework.homework_id = 7316 AND users.id = 77

question_results.correct question_results.attempts

1	F	1
2	F	0
3	F	0
4	F	0

Type of data currently in cell: NULL 0 byte(s)

Remote Identity Name Commit Last modified Size

SQL Log Plot DB Schema Remote UTF-8

Correctly finds question correct status and attempt count for the given user id (77) for each of the 12 questions in the given homework (7316) within the given class (824)

4.5.17.2.2 Test code

```
if __name__ == '__main__':
    cid: int = input("Enter class id: ")
    sid: int = input("Enter student id: ")
    print(get_scores_of_student_in_class(cid, sid))
```

4.5.17.2.3 With valid data:

The screenshot shows two instances of DB Browser for SQLite. The top instance displays the 'class_user' table with one row: id=8980, class_id=134, student_id=19. The bottom instance displays the 'class_homework' table with 8 rows of data:

	id	class_id	homework_id	due_date
1	936	134	936	2017-12-16
2	1836	134	1836	2018-4-23
3	2681	134	2681	2015-4-24
4	3346	134	3346	2016-2-5
5	5424	134	5424	2020-11-12
6	5504	134	5504	2012-11-23
7	7061	134	7061	2017-2-16
8	8256	134	8256	2014-12-13

Enter class id: 134

Enter student id: 19

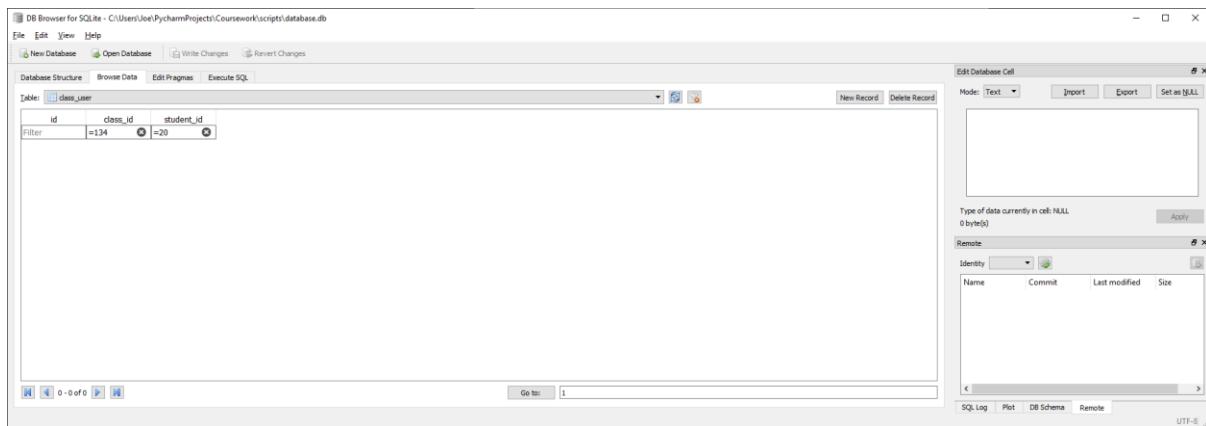
```
[['name936', 8, 67, 4], ['name1836', 7, 64, 5], ['name2681', 2, 29, 5], ['name3346', 3, 38, 5],
 ['name5424', 2, 25, 5], ['name5504', 6, 50, 5], ['name7061', 4, 44, 5], ['name8256', 0, 0, 5]]
```

Process finished with exit code 0

Correctly finds all 8 matching homework for the given class (134) and student (19) and successfully outputs all results as an array of tuples containing, for each homework, [homework name, score, percentage, number of attempts]

4.5.17.2.4 With Invalid data:

Homework with no results stored:



Student is not in the class so should have no results for any of the homework within the class

Enter class id: 134

Enter student id: 20

```
[['name936', 0, 'N/A', 0], ['name1836', 0, 'N/A', 0], ['name2681', 0, 'N/A', 0], ['name3346', 0, 'N/A', 0], ['name5424', 0, 'N/A', 0], ['name5504', 0, 'N/A', 0], ['name7061', 0, 'N/A', 0], ['name8256', 0, 'N/A', 0]]
```

Process finished with exit code 0

Percentages for each homework are correctly set to 'N/A'

Any errors relating to the use of invalid class_id and/or student_id should be addressed before calling function

4.5.17.2.5 Evaluation

Function correctly finds all homework in the given class and, for each homework, for the given student, finds the [homework name, score, percentage, number of attempts]. This data is then correctly outputted as an array of tuples containing the data for each homework as just defined.

In the event of no questions being present within a homework for any reason which would result in a division by zero error usually, the program correctly sets percentage to 'N/A'.

4.5.18 Function to get the question text for a given question

4.5.18.1 Final Code

```
def get_question_text_of_question(question_id: int):
    # Returns question text from database for given question id
    sql: str = 'SELECT question_text FROM questions WHERE id = ?;'
    c.execute(sql, (question_id,))
    return c.fetchall()[0][0]
```

4.5.18.2 Testing

4.5.18.2.1 SQL

The screenshot shows two instances of DB Browser for SQLite. The top instance displays the 'questions' table with the following data:

id	name	type_id	question_text	correct_answer	answer_b	answer_c	answer_d
4567	question4567	1	question text ...	correct answe... b 4567	c 4567	d 4567	

The bottom instance shows the results of the SQL query `SELECT question_text FROM questions WHERE id = 4567;`:

question_text
question text 4567

Both screenshots show the 'Edit Database Cell' and 'Remote' panes.

Correctly returns correct question text (question text 4567) for given question id (4567)

4.5.18.2.2 Test code

```
if __name__ == '__main__':
    qid: int = input("Enter question id: ")
    print(get_question_text_of_question(qid))
```

4.5.18.2.3 With valid data:

```
C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe  
C:/Users/Joe/PycharmProjects/Coursework/scripts/ui_scripts.py
```

Enter question id: 6782

question text 6782

Process finished with exit code 0

Correctly returns matching question text (question 6782) for given question id (6782)

4.5.18.2.4 With Invalid data:

N/A, any errors should be addressed before calling function by verifying question id in advance

4.5.18.2.5 Evaluation

Correctly returns matching question text for a given valid question id

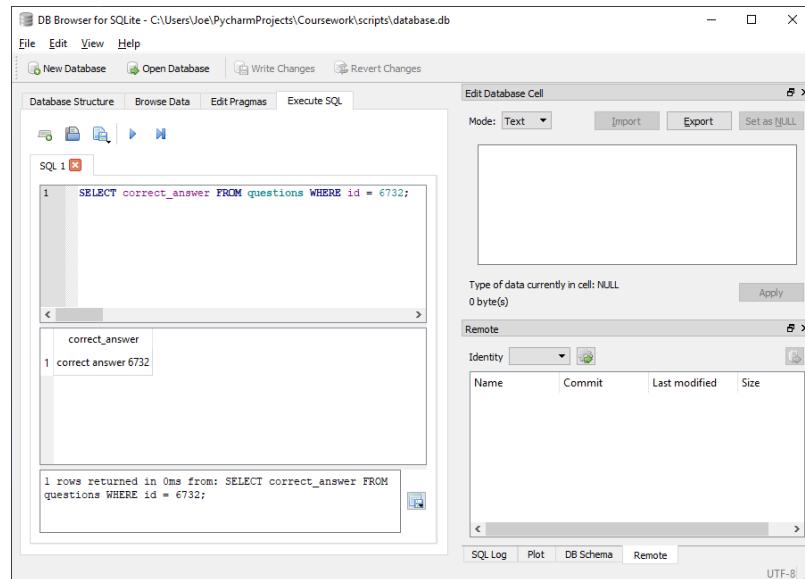
4.5.19 Function to get the correct answer for a given question

4.5.19.1 Final Code

```
def get_correct_answer_of_question(question_id: int):  
    # Returns correct answer from database for given question id  
    sql: str = 'SELECT correct_answer FROM questions WHERE id = ?;'  
    c.execute(sql, (question_id,))  
    return c.fetchall()[0][0]
```

4.5.19.2 Testing

4.5.19.2.1 SQL



Correctly returns matching correct answer (correct answer 6732) for given question id (6732)

4.5.19.2.2 Test code

```
if __name__ == '__main__':  
    qid: int = input("Enter question id: ")  
    print(get_correct_answer_of_question(qid))
```

4.5.19.2.3 With valid data:

C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe
C:/Users/Joe/PycharmProjects/Coursework/scripts/ui_scripts.py

Enter question id: 8245

correct answer 8245

Process finished with exit code 0

Correctly returns matching correct answer (correct answer 8245) for given question id (8245)

4.5.19.2.4 With Invalid data:

N/A, any errors should be addressed before calling function by verifying question id in advance

4.5.19.2.5 Evaluation

Correctly returns matching correct answer for given question id

4.5.20 Function to remove a question from a homework

4.5.20.1 Final Code

```
def remove_question_from_homework(question_id: int, homework_id: int):
    # Deletes any entry in homework_questions which links the given question to the
    given homework
    sql: str = 'DELETE FROM homework_questions WHERE question_id = ? AND home-
    work_id = ?;'
    c.execute(sql, (question_id, homework_id))
    conn.commit()
```

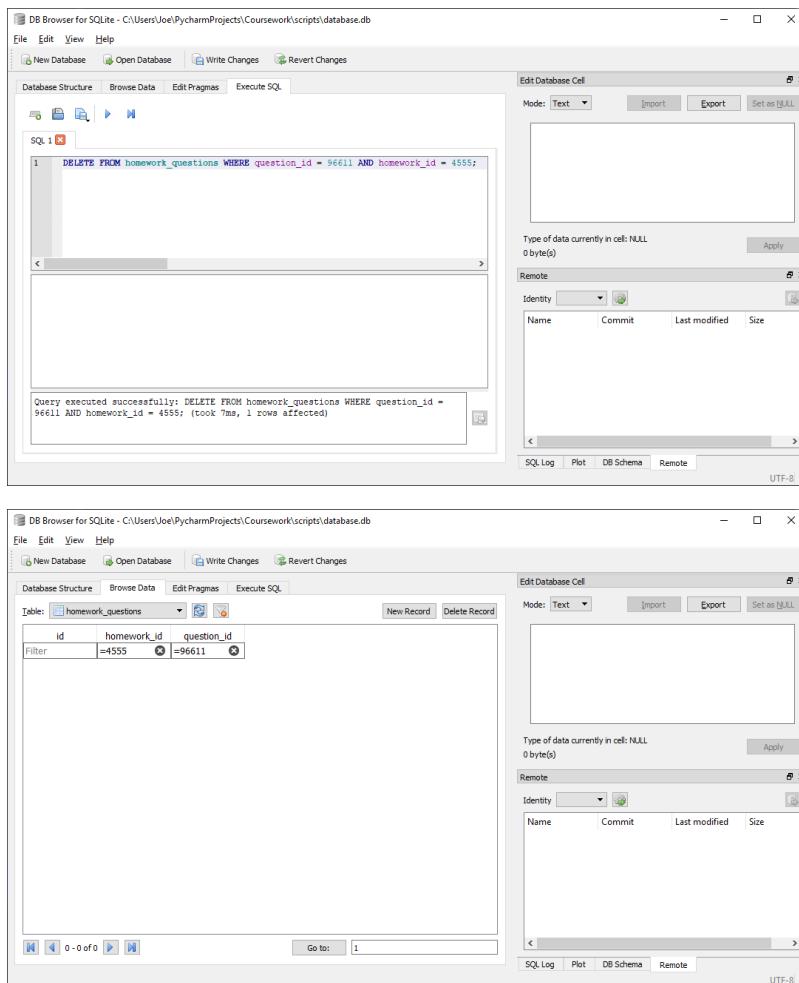
4.5.20.2 Testing

4.5.20.2.1 SQL

The screenshot shows the DB Browser for SQLite interface. The database is C:\Users\Joe\PycharmProjects\Coursework\scripts\database.db. The homework_questions table is selected, displaying the following data:

	id	homework_id	question_id
1	96611	4555	96611

Below the table, there is a filter bar with the values: id = 4555, homework_id = 96611, and question_id = 96611. The status bar at the bottom indicates "1 - 1 of 1".

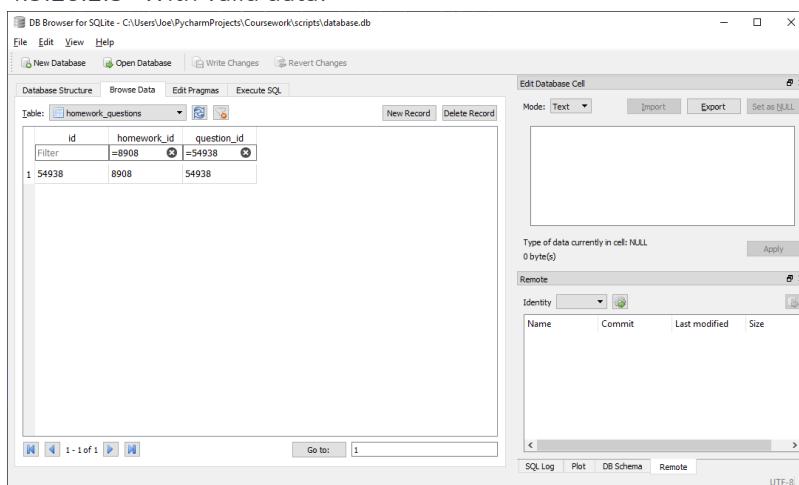


Correctly removes entry in database linking given question (96611) to given homework (4555)

4.5.20.2.2 Test code

```
if __name__ == '__main__':
    qid: int = input("Enter question id: ")
    hid: int = input("Enter homework id: ")
    remove_question_from_homework(qid, hid)
```

4.5.20.2.3 With valid data:

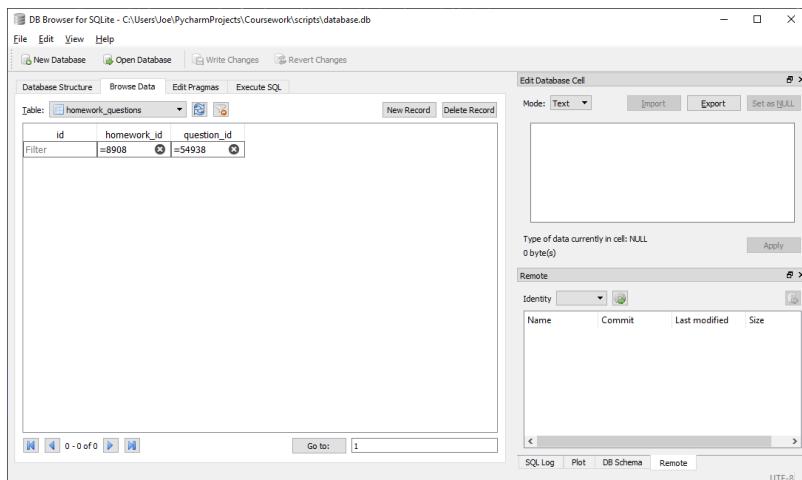


C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe
C:/Users/Joe/PycharmProjects/Coursework/scripts/ui_scripts.py

Enter question id: 54938

Enter homework id: 8908

Process finished with exit code 0



Correctly removes entry in database linking given question (54938) to given homework (8908)

4.5.20.2.4 With Invalid data:

N/A, any errors should be addressed before calling function by verifying question belongs to homework and both are valid in advance

4.5.20.2.5 Evaluation

Correctly removes entry in database linking given question to given homework

4.5.21 Function to insert a question into a homework

4.5.21.1 Final Code

```
def insert_question_into_homework(class_id: int, homework_id: int, question_id: int):
    # Inserts required data to link a homework and question into the homework_questions table (id of question and id of
    # homework to be added to)
    sql: str = 'INSERT INTO homework_questions(homework_id, question_id) VALUES(?,?)'
    c.execute(sql, (homework_id, question_id))
    # Inserts required data for the new question into the question_results table by
    # finding all students within the
    # given class and for each student inserting an entry linking the user id to
    # the question id along with the number
    # of attempts (set initially to 0 as question unattempted) and correct status
    # (set initially to False as question
    # unattempted)
    for user in get_students_of_class(class_id):
        sql: str = 'INSERT INTO question_results(user_id, question_id, attempts, correct) VALUES(?,?,?,?)'
        c.execute(sql, (user[0], question_id, 0, 0))
    conn.commit()
```

4.5.21.2 Testing

4.5.21.2.1 SQL

The figure consists of three vertically stacked screenshots of the DB Browser for SQLite application. Each screenshot shows a database window with a toolbar at the top and a main content area below.

Screenshot 1: Shows the 'homework_questions' table in the Database Structure tab. A record with id=67 and question_id=45 is selected. An 'Edit Database Cell' dialog is open over the selected cell, showing 'Mode: Text' and a large empty text area. A status message at the bottom left says 'Type of data currently in cell: NULL 0 byte(s)'. A 'Remote' panel is visible on the right.

Screenshot 2: Shows the 'Execute SQL' tab with the following SQL command entered:
1 INSERT INTO homework_questions(homework_id, question_id) VALUES(67,45)
A status message at the bottom left says 'Query executed successfully: INSERT INTO homework_questions(homework_id, question_id) VALUES(67,45) (took 3ms, 1 rows affected)'.

Screenshot 3: Shows the 'homework_questions' table again. A new record has been added with id=100001, homework_id=67, and question_id=45. The 'Edit Database Cell' dialog is still open over the newly inserted record.

Correctly inserts new entry into homework_questions linking given homework (67) to given question (45)

The image consists of three vertically stacked screenshots of the DB Browser for SQLite application. Each screenshot shows a database window with the following details:

- File Bar:** File, Edit, View, Help
- Toolbar:** New Database, Open Database, Write Changes, Revert Changes
- Database Structure Tab:** Shows the 'question_results' table with columns: id, user_id, question_id, attempts, correct.
- Table View:** A grid view showing one row where user_id = 4 and question_id = 342. The 'attempts' column has a filter set to 0, and the 'correct' column has a filter set to F.
- SQL Log Tab:** Shows the SQL query being run.
- Edit Database Cell Dialog:** A modal dialog for editing the current cell. It shows the mode is 'Text', the value is NULL, and the type is '0 byte(s)'.
- Identity Dialog:** A small dialog showing the current identity values for Name, Commit, Last modified, and Size.
- Status Bar:** Shows 'UTF-8' encoding.

In the middle screenshot, the SQL log shows the execution of the following SQL query:

```
INSERT INTO question_results(user_id, question_id, attempts, correct) VALUES(4,342,0,'F')
```

The status bar at the bottom of this screenshot indicates "Query executed successfully: INSERT INTO question_results(user_id, question_id, attempts, correct) VALUES(4,342,0,'F') (took 0ms, 1 rows affected)".

In the bottom screenshot, the table view now shows a single row with the following data:

	id	user_id	question_id	attempts	correct
1	1004031	4	342	0	F

Correctly inserts new entry into question_results linking given question (342) to given user (4) along with the given attempts count (0) and correct status (F)

4.5.21.2.2 Test code

```
if __name__ == '__main__':
    qid: int = input("Enter question id: ")
    hid: int = input("Enter homework id: ")
    remove_question_from_homework(qid, hid)
```

4.5.21.2.3 With valid data:

The image contains three separate screenshots of the DB Browser for SQLite application, each displaying a different table with valid data.

Screenshot 1: homework_questions Table

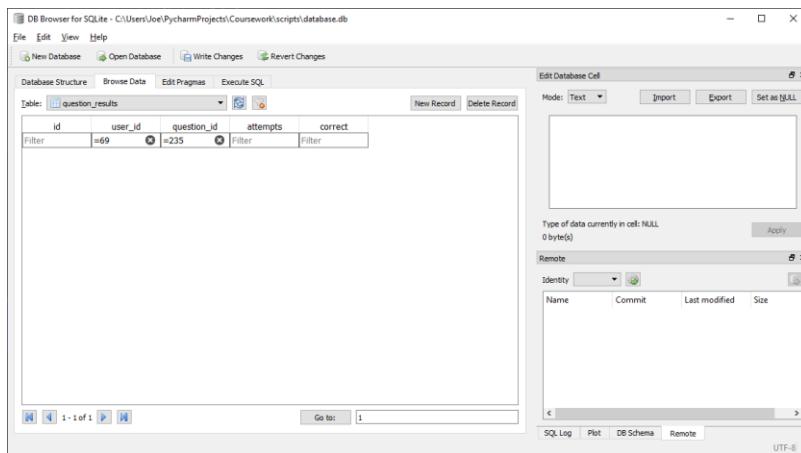
id	homework_id	question_id
=413	=235	

Screenshot 2: class_user Table

	id	class_id	student_id
1	1976	21	98
2	2106	21	14
3	2470	21	23
4	3305	21	73
5	6476	21	16
6	6855	21	36
7	9161	21	90
8	9949	21	21
9	9983	21	69

Screenshot 3: question_results Table

id	user_id	question_id	attempts	correct
=98	=235			



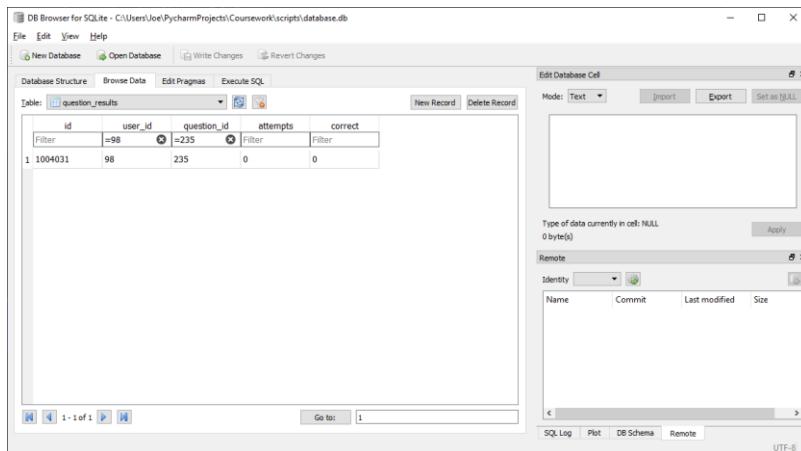
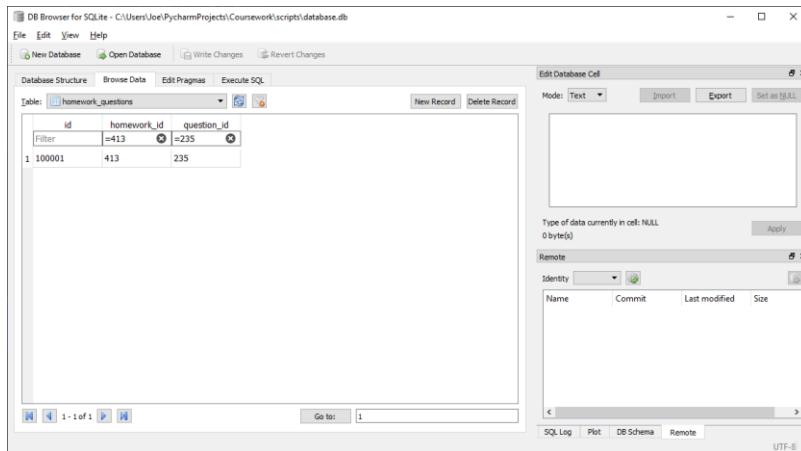
C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe
C:/Users/Joe/PycharmProjects/Coursework/scripts/ui_scripts.py

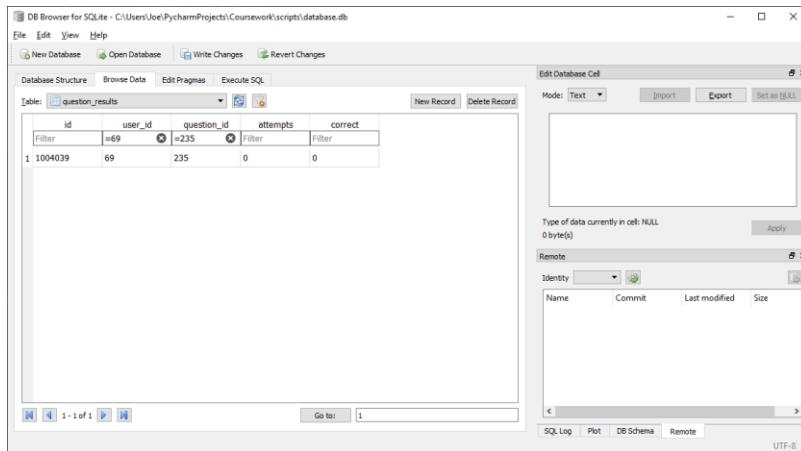
Enter class id: 21

Enter question id: 235

Enter homework id: 413

Process finished with exit code 0





Correctly links given question (235) to given homework (413) in homework_questions and creates a valid entry in question_results for each member of given class

4.5.21.2.4 With Invalid data:

N/A, any errors should be addressed before calling function by verifying class id, homework id and question id are valid and homework is not already set

4.5.21.2.5 Evaluation

Correctly links given question to given homework in homework_questions and creates a valid entry in question_results for each member of given class

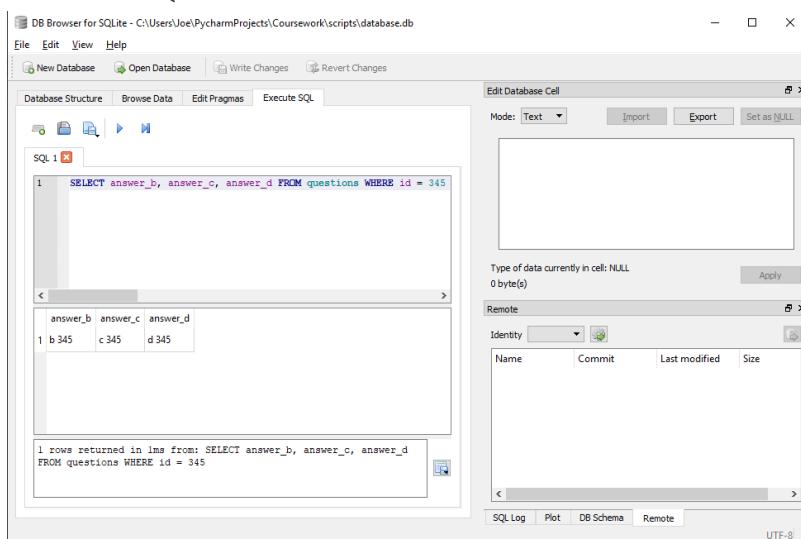
4.5.22 Function to get the incorrect answers for a given question

4.5.22.1 Final Code

```
def get_incorrect_answers_of_question(question_id: int) -> list:
    # Returns three incorrect answers from database questions table for given question id
    sql: str = 'SELECT answer_b, answer_c, answer_d FROM questions WHERE id = ?;'
    c.execute(sql, (question_id,))
    return c.fetchall()[0]
```

4.5.22.2 Testing

4.5.22.2.1 SQL



Correctly returns matching incorrect answer (b 345, c 345, d 345) for given question id (345)

4.5.22.2.2 Test code

```
if __name__ == '__main__':
    qid: int = input("Enter question id: ")
    print(get_correct_answer_of_question(qid))
```

4.5.22.2.3 With valid data:

```
C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe
C:/Users/Joe/PycharmProjects/Coursework/scripts/ui_scripts.py
```

Enter question id: 67845

('b 67845', 'c 67845', 'd 67845')

Process finished with exit code 0

Correctly returns matching correct answer (b 67845, c 67845, d 67845) for given question id (67845)

4.5.22.2.4 With Invalid data:

N/A, any errors should be addressed before calling function by verifying question id in advance

4.5.22.2.5 Evaluation

Correctly returns matching incorrect answers for given question id

4.5.23 Function to get the correct status for a given question and user

4.5.23.1 *Issues during development*

Initially the below code was written:

```
def get_correct_status_of_question(student_id: int, question_id: int) -> bool:
    # Returns correct status from database question_results table for given question id and student id
    sql: str = 'SELECT correct FROM question_results WHERE user_id = ? AND question_id = ?;'
    c.execute(sql, (student_id, question_id))
    return c.fetchall()[0][0]
```

This would result in the following output:

```
C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe
C:/Users/Joe/PycharmProjects/Coursework/scripts/ui_scripts.py
```

Enter student id: 67

Enter question id: 1913

F

Process finished with exit code 0

While the correct status is correctly indicated to be False, the character F is itself not actually a Boolean value, but a string so is unusable without conversion. This was done by removing the final line and replacing it with:

```

# Converts stored string value to a boolean representation
if c.fetchall()[0][0] == 'T':
    return True
else:
    return False

```

The function now returns a correct Boolean value (see final testing of function)

4.5.23.2 Final Code

```

def get_correct_status_of_question(student_id: int, question_id: int) -> bool:
    # Returns correct status from database question_results table for given question id and student id
    sql: str = 'SELECT correct FROM question_results WHERE user_id = ? AND question_id = ?;'
    c.execute(sql, (student_id, question_id))
    # Converts stored string value to a boolean representation
    if c.fetchall()[0][0] == 'T':
        return True
    else:
        return False

```

4.5.23.3 Testing

4.5.23.3.1 SQL

The screenshot shows the DB Browser for SQLite interface. On the left, the 'Browse Data' tab is selected, displaying the 'question_results' table. The table has columns: id, user_id, question_id, attempts, and correct. There is one row with values: 1, 23, 74644, 0, and F. On the right, there is an 'Edit Database Cell' dialog open over the table, showing the current value 'F' in the 'correct' column.

The screenshot shows the DB Browser for SQLite interface. On the left, the 'Execute SQL' tab is selected, containing the SQL query: 'SELECT correct FROM question_results WHERE user_id = 23 AND question_id = 74644'. The results pane on the right shows a single row with the column 'correct' containing the value 'F'. Below the results, a message indicates '1 rows returned in 128ms'.

Correctly returns matching correct status (F) for given question id (74644) and user id (23)

4.5.23.3.2 Test code

```
if __name__ == '__main__':
    uid: int = input("Enter student id: ")
    qid: int = input("Enter question id: ")
    print(get_incorrect_answers_of_question(qid))
```

4.5.23.3.3 With valid data:

4.5.23.3.3.1 Stored value is True

The screenshot shows the DB Browser for SQLite interface. On the left, there's a table named 'question_results' with four columns: id, user_id, question_id, and attempts. There is one row with values: id=661500, user_id=67, question_id=11090, and attempts=2. The 'correct' column contains the value 'T'. On the right, there's an 'Edit Database Cell' panel with a large empty text area, indicating the current cell type is NULL. Below it, there's a 'Remote' section and a 'Identity' section.

C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe
C:/Users/Joe/PycharmProjects/Coursework/scripts/ui_scripts.py

Enter question id: 67845

('b 67845', 'c 67845', 'd 67845')

Process finished with exit code 0

Correctly returns correct Boolean correct status (True) for given question id (11090) and user id (67)

4.5.23.3.3.2 Stored value is False

The screenshot shows the DB Browser for SQLite interface. On the left, there's a table named 'question_results' with four columns: id, user_id, question_id, and attempts. There is one row with values: id=661499, user_id=67, question_id=1913, and attempts=4. The 'correct' column contains the value 'F'. On the right, there's an 'Edit Database Cell' panel with a large empty text area, indicating the current cell type is NULL. Below it, there's a 'Remote' section and a 'Identity' section.

```
C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe  
C:/Users/Joe/PycharmProjects/Coursework/scripts/ui_scripts.py
```

Enter student id: 67

Enter question id: 1913

False

Process finished with exit code 0

Correctly returns correct Boolean correct status (False) for given question id (11090) and user id (1913)

4.5.23.3.4 With Invalid data:

N/A, any errors should be addressed before calling function by verifying question id and user id are valid and question and user are related in advance

4.5.23.3.5 Evaluation

Correctly returns Boolean correct status for a given question id and user id

4.5.24 Function to increment user attempts at question

4.5.24.1 Final Code

```
def increment_user_attempts_at_question(user_id: int, question_id: int):  
    # Finds the entry in the question_results table linking a given user to a given  
    # question and adds 1 to the value  
    # in the attempts field  
    sql: str = 'UPDATE question_results SET attempts = attempts + 1 WHERE user_id =  
    ? AND question_id = ?;  
    c.execute(sql, (user_id, question_id))  
    conn.commit()
```

4.5.24.2 Testing

4.5.24.2.1 SQL

The screenshot shows the DB Browser for SQLite interface. The database is C:/Users/Joe/PycharmProjects/Coursework/scripts/database.db. The 'question_results' table is selected, displaying the following data:

id	user_id	question_id	attempts	correct
1	375992	36057	4	T

The screenshot shows two windows of DB Browser for SQLite. The top window is titled 'DB Browser for SQLite - C:\Users\Joe\PycharmProjects\Coursework\scripts\database.db'. It has a SQL tab containing the following SQL code:

```
1 UPDATE question_results SET attempts = attempts + 1 WHERE user_id = 39 AND question_id = 36057
```

Below the SQL tab, a message says 'Query executed successfully: UPDATE question_results SET attempts = attempts + 1 WHERE user_id = 39 AND question_id = 36057 (took 5ms, 1 rows affected)'. The bottom window is also titled 'DB Browser for SQLite - C:\Users\Joe\PycharmProjects\Coursework\scripts\database.db' and shows a table named 'question_results' with the following data:

id	user_id	question_id	attempts	correct
1	39	36057	5	T

Correctly incremented attempts count (from 4 to 5) for given question id (36057) and user id (39)

4.5.24.2.2 Test code

```
if __name__ == '__main__':
    uid: int = input("Enter student id: ")
    qid: int = input("Enter question id: ")
    increment_user_attempts_at_question(uid, qid)
```

4.5.24.2.3 With valid data:

The screenshot shows two windows of DB Browser for SQLite. The top window is titled 'DB Browser for SQLite - C:\Users\Joe\PycharmProjects\Coursework\scripts\database.db'. It has a SQL tab containing the following SQL code:

```
1 UPDATE question_results SET attempts = attempts + 1 WHERE user_id = 45 AND question_id = 9229
```

The bottom window is also titled 'DB Browser for SQLite - C:\Users\Joe\PycharmProjects\Coursework\scripts\database.db' and shows a table named 'question_results' with the following data:

id	user_id	question_id	attempts	correct
1	45	9229	2	F

C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe
C:/Users/Joe/PycharmProjects/Coursework/scripts/ui_scripts.py

Enter student id: 45

Enter question id: 9229

Process finished with exit code 0

The screenshot shows the DB Browser for SQLite interface. On the left, the 'question_results' table is displayed with the following data:

id	user_id	question_id	attempts	correct
1	45	9229	3	F

A new record is being inserted with user_id=45 and question_id=9229. The 'Edit Database Cell' dialog is open over the 'attempts' field, showing its current value as 'NULL'. The 'Model' dropdown is set to 'Text'. There are 'Import' and 'Export' buttons, and a 'Get as NULL' button.

Correctly incremented attempts count (from 2 to 3) for given question id (9229) and user id (45)

4.5.24.2.4 With Invalid data:

N/A, any errors should be addressed before calling function by verifying question id and user id are valid and question and user are related in advance

4.5.24.2.5 Evaluation

Correctly incremented attempts count in question_results for given question id and user id (45)

4.5.25 Function to mark question as correct

4.5.25.1 Final Code

```
def mark_question_as_correct(user_id: int, question_id: int):
    # Finds the entry in the question_results table linking a given user to a given
    # question and sets the value in the
    # attempts field to 'T'
    sql: str = 'UPDATE question_results SET correct = ? WHERE user_id = ? AND ques-
    tion_id = ?'
    c.execute(sql, ('T', user_id, question_id))
    conn.commit()
```

4.5.25.2 Testing

4.5.25.2.1 SQL

The figure consists of three vertically stacked screenshots of DB Browser for SQLite. Each screenshot shows a database window with a toolbar at the top and a table view below.

- Screenshot 1:** Shows the 'question_results' table with one row. The 'correct' column contains 'F'. The table has columns: id, user_id, question_id, attempts, and correct.
- Screenshot 2:** Shows the 'Execute SQL' tab with the following SQL query:


```
1 UPDATE question_results SET correct = ? WHERE user_id = 6 AND question_id = 50109;
```

 The query is executed successfully, updating the 'correct' value to 'T'.
- Screenshot 3:** Shows the 'question_results' table again, but now the 'correct' column for the row (id=50680) has been updated to 'T'.

Correctly set correct status to 'T' for given question id (50109) and user id (6)

4.5.25.2.2 Test code

```
if name == 'main':
    uid: int = input("Enter student id: ")
    qid: int = input("Enter question id: ")
    mark_question_as_correct(uid, qid)
```

4.5.25.2.3 With valid data:

DB Browser for SQLite - C:\Users\Joe\PycharmProjects\Coursework\scripts\database.db

Table: question_results

	id	user_id	question_id	attempts	correct
1	983537	99	83848	4	F

Edit Database Cell

Type of data currently in cell: NULL
0 byte(s)

Mode: Text Import Export Set as NULL

Remote

Identity Commit Last modified Size

SQL Log Plot DB Schema Remote

C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe
C:/Users/Joe/PycharmProjects/Coursework/scripts/ui_scripts.py

Enter student id: 99

Enter question id: 83848

Process finished with exit code 0

DB Browser for SQLite - C:\Users\Joe\PycharmProjects\Coursework\scripts\database.db

Table: question_results

	id	user_id	question_id	attempts	correct
1	439684	45	9229	3	F

Edit Database Cell

Type of data currently in cell: NULL
0 byte(s)

Mode: Text Import Export Set as NULL

Remote

Identity Commit Last modified Size

SQL Log Plot DB Schema Remote

DB Browser for SQLite - C:\Users\Joe\PycharmProjects\Coursework\scripts\database.db

Table: question_results

	id	user_id	question_id	attempts	correct
1	983537	99	83848	4	T

Edit Database Cell

Type of data currently in cell: NULL
0 byte(s)

Mode: Text Import Export Set as NULL

Remote

Identity Commit Last modified Size

SQL Log Plot DB Schema Remote

Correctly set correct status to 'T' for given question id (83848) and user id (99)

4.5.25.2.4 With Invalid data:

N/A, any errors should be addressed before calling function by verifying question id and user id are valid and question and user are related in advance

4.5.25.2.5 Evaluation

Correctly sets correct status to 'T' for given question id and user id

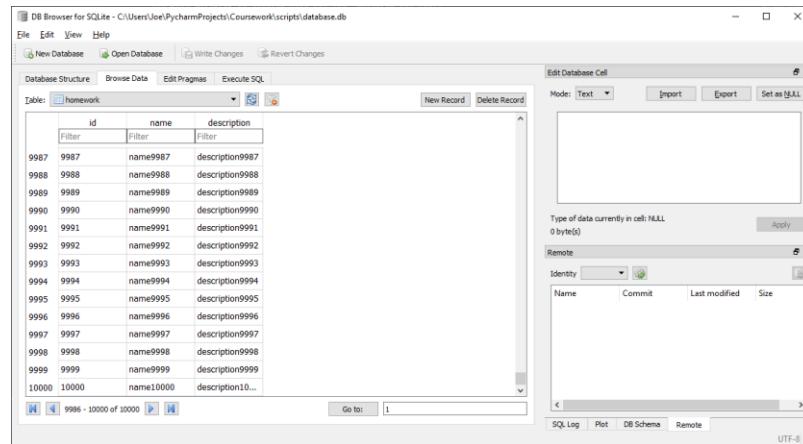
4.5.26 Function to insert a new homework into database

4.5.26.1 Final Code

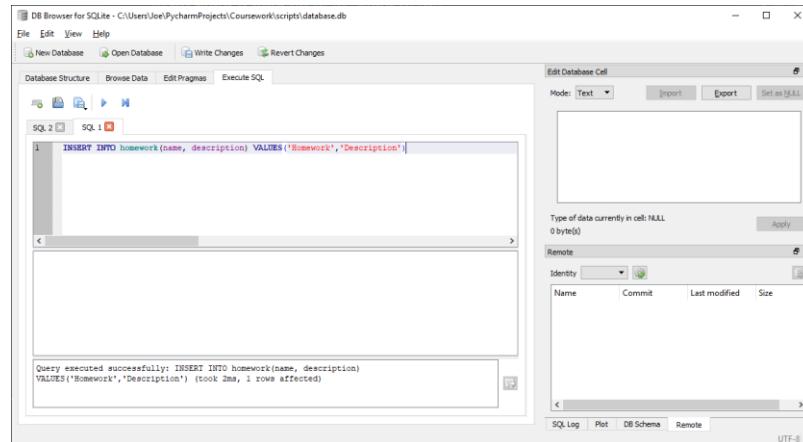
```
def insert_new_homework(name: str, description: str) -> int:
    # Inserts required data (name and description) into homework table to create
    new homework instance
    sql: str = 'INSERT INTO homework(name, description) VALUES(?,?)'
    c.execute(sql, (name, description))
    # Gets the id of the newly created homework, commits database changes and re-
    turns id
    homework_id: int = c.lastrowid
    conn.commit()
    return homework_id
```

4.5.26.2 Testing

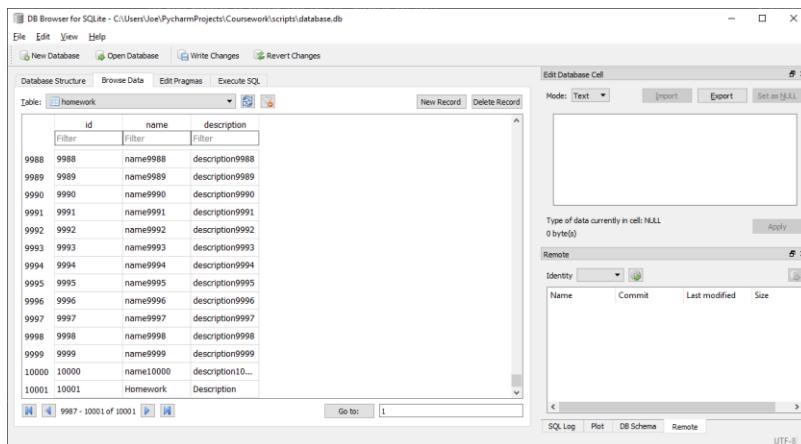
4.5.26.2.1 SQL



The screenshot shows the 'homework' table in DB Browser for SQLite. The table has three columns: 'id', 'name', and 'description'. The 'id' column contains values from 9987 to 10000. The 'name' and 'description' columns contain placeholder text like 'name9987', 'description9987', etc. A modal window titled 'Edit Database Cell' is open over the table, showing the current value 'NULL' in a cell.



The screenshot shows the 'homework' table in DB Browser for SQLite. The table has three columns: 'id', 'name', and 'description'. The 'id' column contains values from 9987 to 10000. The 'name' and 'description' columns contain placeholder text like 'name9987', 'description9987', etc. A modal window titled 'Edit Database Cell' is open over the table, showing the current value 'NULL' in a cell. Below the table, a SQL log window shows the execution of the 'INSERT' query: 'Query executed successfully: INSERT INTO homework(name, description) VALUES('Homework','Description')'.

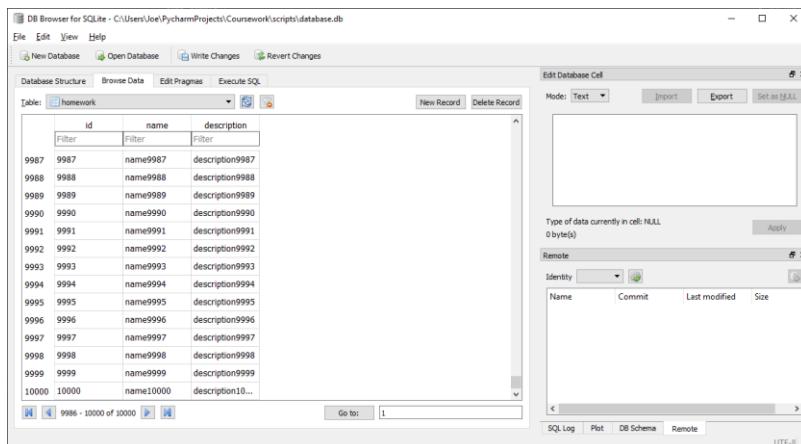


Correctly inserts a new entry into homework table with given name (homework) and description (description)

4.5.26.2.2 Test code

```
if __name__ == '__main__':
    uid: int = input("Enter student id: ")
    qid: int = input("Enter question id: ")
    mark_question_as_correct(uid, qid)
```

4.5.26.2.3 With valid data:



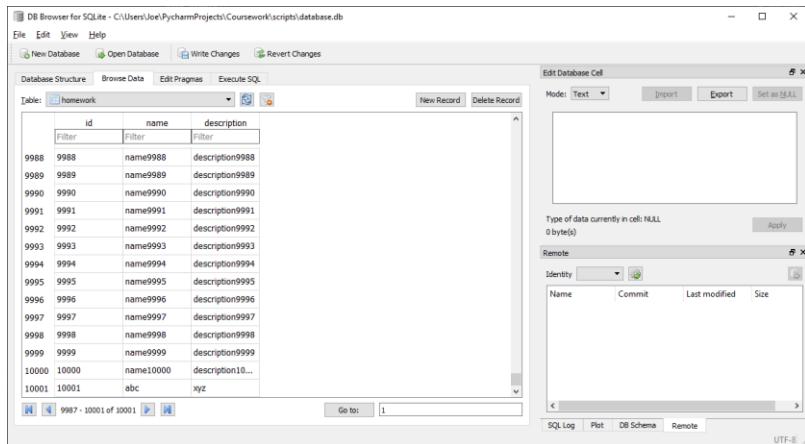
```
C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe
C:/Users/Joe/PycharmProjects/Coursework/scripts/ui_scripts.py
```

Enter homework name: abc

Enter homework description: xyz

10001

Process finished with exit code 0



Correctly inserts a new entry into homework table with given name (abc) and description (xyz) and returns its id in the homework table

4.5.26.2.4 With Invalid data:

N/A, any errors should be addressed before calling function by verifying homework name and description are strings before passing to function

4.5.26.2.5 Evaluation

Correctly inserts a new entry into homework table with given name and description and returns its id in the homework table

4.5.27 Function to add a homework to a class

4.5.27.1 Final Code

```
def add_homework_to_class(class_id: int, homework_id: int, due_date: datetime.date):
    # Converts datetime value passed to an array of the format [year, month, day]
    # by converting to string and splitting
    # at '-'
    due_year_month_day: list = str(due_date).split('-')
    string_due_date: str = "{}-{}-{}".format(int(due_year_month_day[0]),
                                              int(due_year_month_day[1]),
                                              int(due_year_month_day[2]))
    # Inserts required data (class id, homework id, due date) into class homework
    # table to create new class homework
    # relationship
    sql: str = 'INSERT INTO class_homework(class_id, homework_id, due_date) VALUES(?, ?, ?)'
    c.execute(sql, (class_id, homework_id, string_due_date))
    conn.commit()
```

4.5.27.2 Testing

4.5.27.2.1 SQL

The figure consists of three vertically stacked screenshots of the DB Browser for SQLite application.

Screenshot 1: Shows the 'Database Structure' tab with the 'class_homework' table selected. The table has columns: id, class_id, homework_id, and due_date. A new record is being inserted with values: 1, 1, 1, and '2020-1-1'. The 'Edit Database Cell' dialog is open, showing the current value as NULL and the input field containing '2020-1-1'.

id	class_id	homework_id	due_date
9955	9998	786	9998
9956	9999	608	9999
9957	10000	202	10000
9958	10003	4	10005
9959	10004	1008	10006
9960	10005	1008	10007
9961	10006	1008	10008
9962	10007	1008	10009
9963	10008	1008	10010
9964	10009	1008	10011
9965	10010	1010	10012
9966	10011	1008	10013
9967	10012	1008	10014
9968	10013	1008	10015

Screenshot 2: Shows the 'Execute SQL' tab with the SQL command: `INSERT INTO class_homework(class_id, homework_id, due_date) VALUES(1,1,'2020-1-1')`. The message 'Query executed successfully: INSERT INTO class_homework(class_id, homework_id, due_date) VALUES(1,1,'2020-1-1') (took 1ms, 1 rows affected)' is displayed.

Screenshot 3: Shows the 'Database Structure' tab again, with the newly inserted row (id=9969, class_id=1, homework_id=1, due_date='2020-1-1') visible in the table.

Correctly inserts a new entry into class homework table relating given class id (1) and homework id (1) with the given due date (2020-1-1)

4.5.27.2.2 Test code

```
if __name__ == '__main__':
    cid: int = input("Enter class id: ")
    hid: int = input("Enter homework id: ")
    due_date = input("Enter due date: ")
    add_homework_to_class(cid, hid, due_date)
```

4.5.27.2.3 With valid data:

The screenshot shows the 'class_homework' table in DB Browser for SQLite. The table has columns: id, class_id, homework_id, and due_date. The 'Edit Database Cell' dialog is open on the last row, which contains the values: 9966, 1008, 1008, and 2019-6-21 respectively. The dialog also shows the current type of data in the cell is NULL.

C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe
C:/Users/Joe/PycharmProjects/Coursework/scripts/ui_scripts.py

Enter class id: 24

Enter homework id: 67

Enter due date: 2021-7-8

Process finished with exit code 0

The screenshot shows the 'class_homework' table in DB Browser for SQLite. The table has columns: id, class_id, homework_id, and due_date. The 'Edit Database Cell' dialog is open on the last row, which now contains the values: 9967, 24, 67, and 2021-7-8 respectively. The dialog shows the current type of data in the cell is NULL.

Correctly inserts a new entry into class homework table relating given class id (24) and homework id (67) with the given due date (2021-7-8)

4.5.27.2.4 With Invalid data:

N/A, any errors should be addressed before calling function by verifying homework id and class id exist and are valid integers and that due date is in the future before calling function

4.5.27.2.5 Evaluation

Correctly inserts a new entry into class homework table relating given class id and homework id with the given due date

4.5.28 Function to remove a homework from database

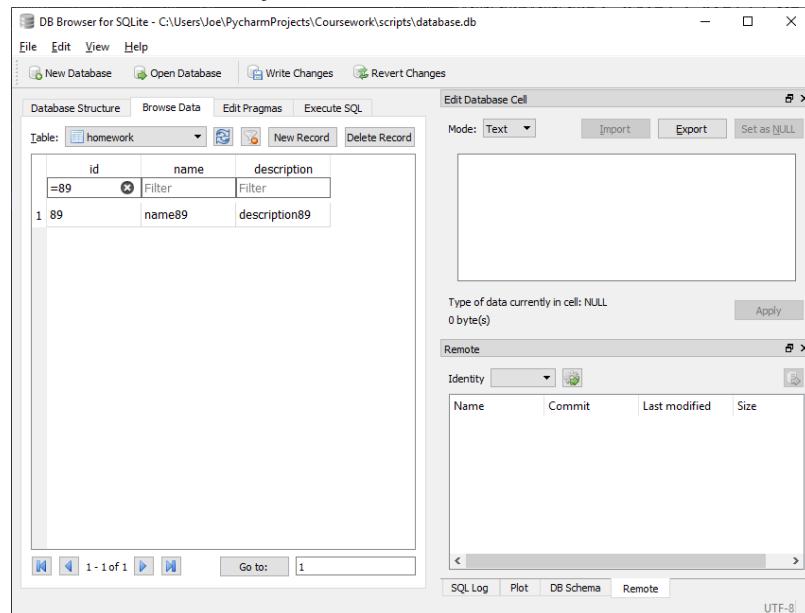
4.5.28.1 Final Code

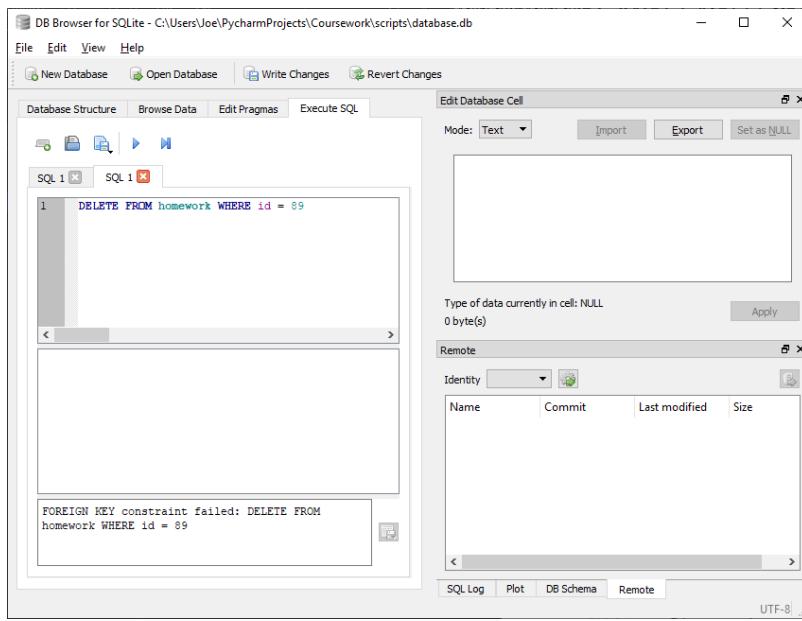
```
def remove_homework(homework_id: int):
    # Deletes all entries in each table which references the existence of the given
    homework
    # Deletes any entry in class homework table relating to the given homework id
    sql: str = 'DELETE FROM class_homework WHERE homework_id = ?'
    c.execute(sql, (homework_id,))
    # Deletes any entry in homework questions table relating to the given homework
    id
    sql: str = 'DELETE FROM homework_questions WHERE homework_id = ?'
    c.execute(sql, (homework_id,))
    # Deletes any entry in homework table relating to the given homework id
    sql: str = 'DELETE FROM homework WHERE id = ?'
    c.execute(sql, (homework_id,))
    conn.commit()
```

4.5.28.2 Testing

4.5.28.2.1 SQL

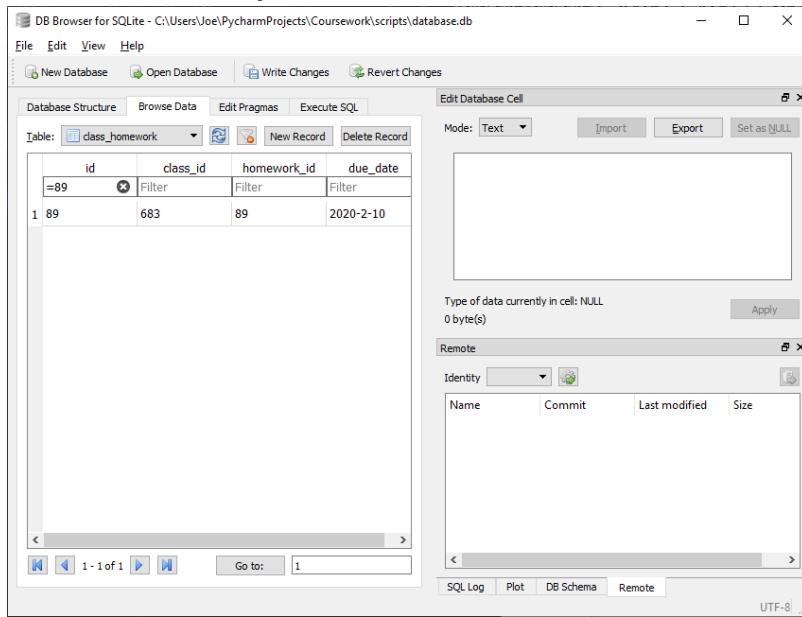
4.5.28.2.1.1 Remove from homework table

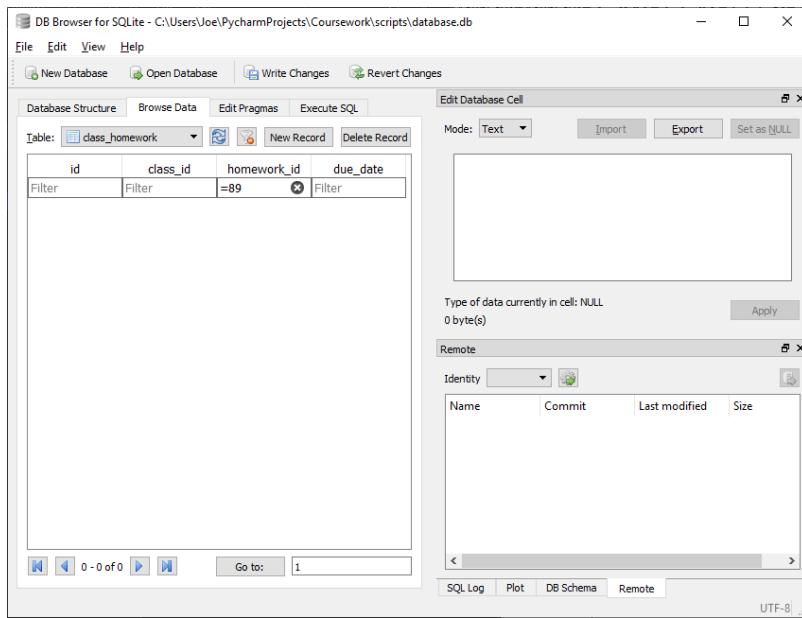
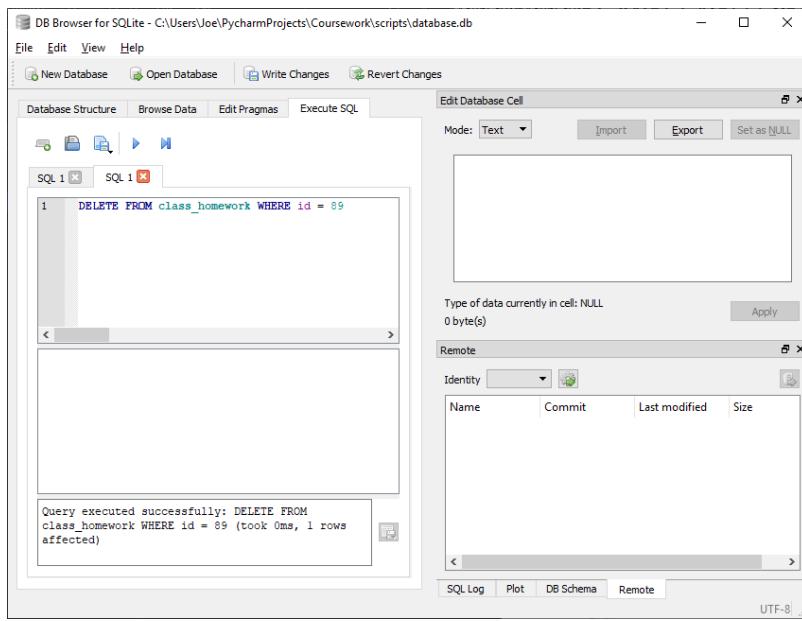




This SQL statement produces an error when run as the current homework id (89) is used as a foreign key elsewhere. This means this statement to remove from the homework table should not be carried out until all references have first been removed in the final python code.

4.5.28.2.1.2 Remove from class homework





Correctly removes entry in class homework table for the entry with homework id given (89)

4.5.28.2.1.3 Remove from homework questions

The screenshot shows the 'homework_questions' table in DB Browser for SQLite. The table has three columns: id, homework_id, and question_id. There are 9 rows of data. A filter is applied where homework_id = 89. The rows are:

	id	homework_id	question_id
1	1855	89	1855
2	33997	89	33997
3	50243	89	50243
4	59628	89	59628
5	61705	89	61705
6	62192	89	62192
7	74807	89	74807
8	78037	89	78037
9	99749	89	99749

The screenshot shows the SQL tab in DB Browser for SQLite. A DELETE query is executed:

```
DELETE FROM homework_questions WHERE homework_id = 89;
```

The message 'Query executed successfully: DELETE FROM homework_questions WHERE homework_id = 89 (took 4ms, 9 rows affected)' is displayed.

The screenshot shows the 'homework_questions' table in DB Browser for SQLite. The table has three columns: id, homework_id, and question_id. The filter is still applied where homework_id = 89. The table is now empty, showing 0 rows.

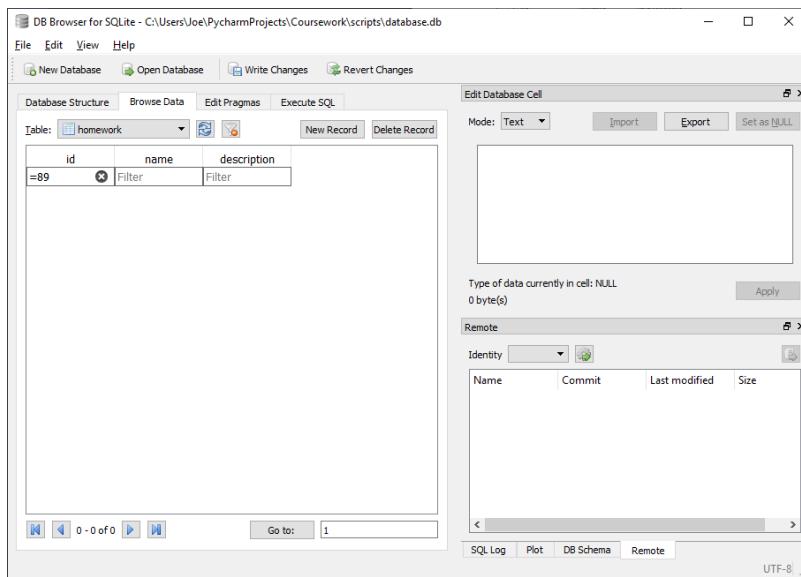
Correctly removes each entry in homework questions table for the entries with homework id given (89)

4.5.28.2.1.4 Remove from homework second attempt

The screenshot shows two instances of DB Browser for SQLite. The top window displays the 'homework' table with one row: id=89, name=name89, and description=description89. The bottom window shows the SQL Log tab with the following query:

```
1 DELETE FROM homework WHERE id = 89
```

The log message indicates: "Query executed successfully: DELETE FROM homework WHERE id = 89 (took 4ms, 1 rows affected)".

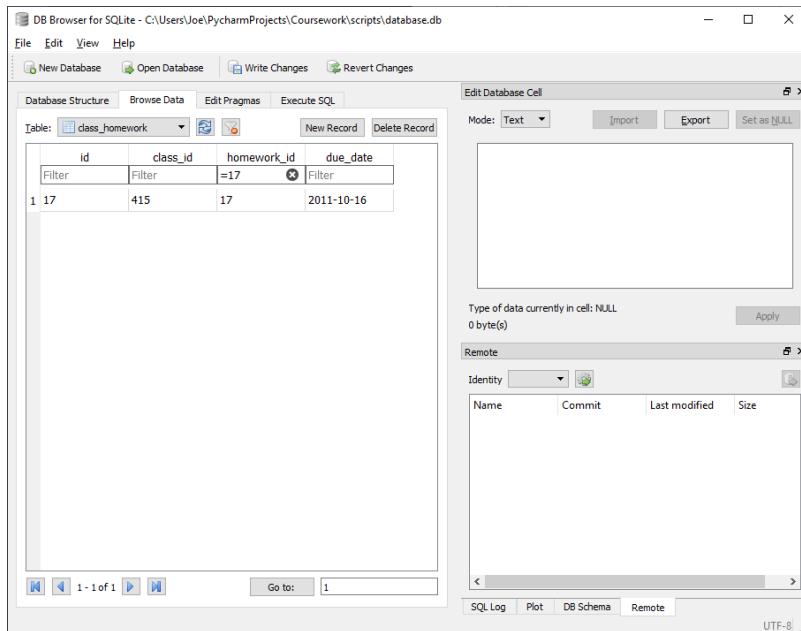


Correctly removes entry in homework table for the entry with homework id given (89)

4.5.28.2.2 Test code

```
if __name__ == '__main__':
    hid: int = input("Enter homework id: ")
    remove_homework(hid)
```

4.5.28.2.3 With valid data:



The screenshot shows the 'homework_questions' table in DB Browser for SQLite. The table has three columns: id, homework_id, and question_id. A filter is applied for homework_id = 17, resulting in 16 rows. The data is as follows:

	id	homework_id	question_id
1	20945	17	20945
2	24275	17	24275
3	25923	17	25923
4	33071	17	33071
5	42209	17	42209
6	42274	17	42274
7	56299	17	56299
8	60060	17	60060
9	62334	17	62334
10	74748	17	74748
11	77127	17	77127
12	86397	17	86397
13	87772	17	87772
14	88073	17	88073
15	95611	17	95611
16	98446	17	98446

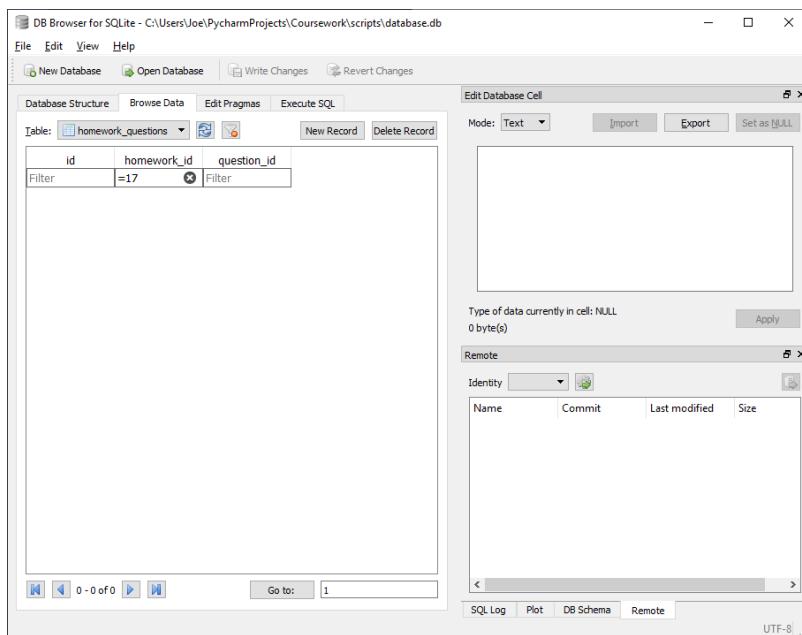
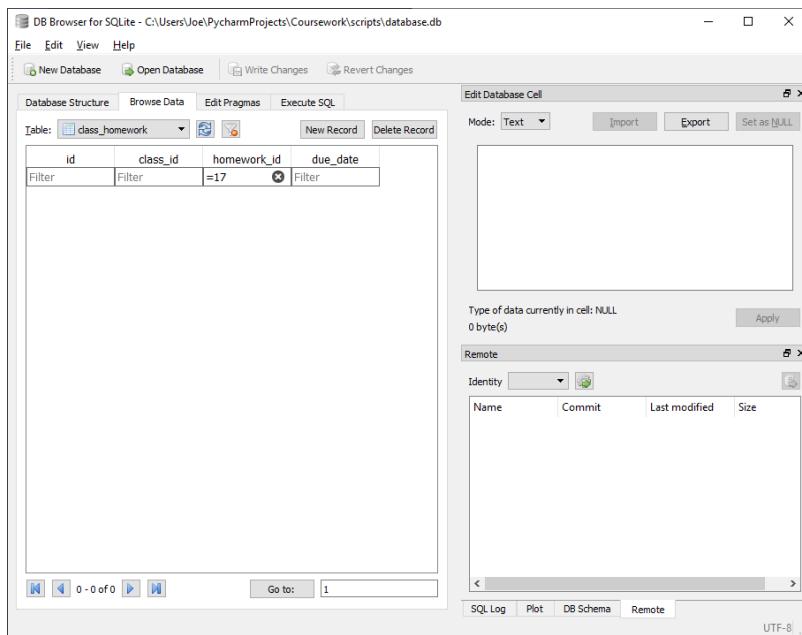
The screenshot shows the 'homework' table in DB Browser for SQLite. The table has three columns: id, name, and description. A filter is applied for id = 17, resulting in 1 row. The data is as follows:

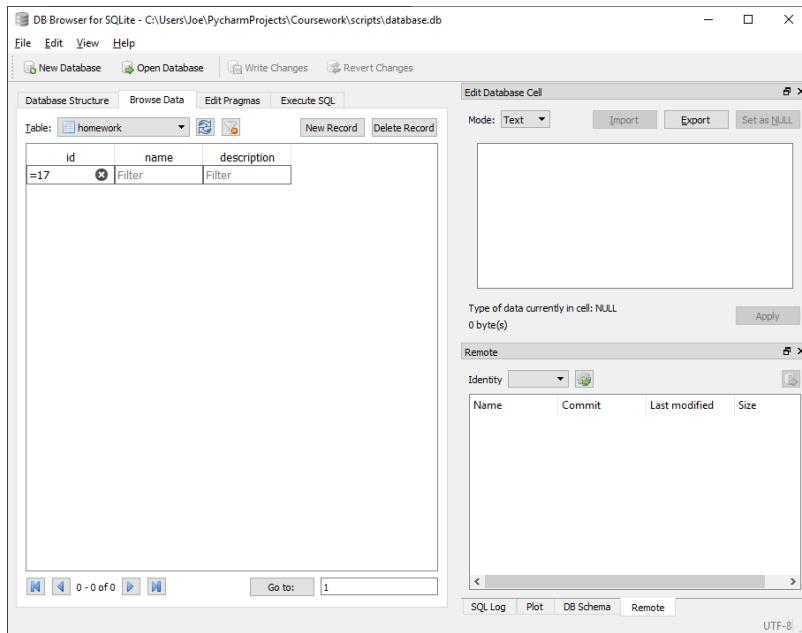
	id	name	description
1	17	name17	description17

```
C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe
C:/Users/Joe/PycharmProjects/Coursework/scripts/ui_scripts.py
```

```
Enter homework id: 17
```

```
Process finished with exit code 0
```





Correctly removes all references to given homework (17) in homework, homework questions and class homework table

4.5.28.2.4 With Invalid data:

N/A, any errors should be addressed before calling function by verifying homework id exists and is a valid integer before calling function

4.5.28.2.5 Evaluation

Correctly removes all references to given homework in homework, homework questions and class homework table

4.5.29 Function to get type of a question

4.5.29.1 Final Code

```
def get_question_type(question_id: int) -> str:
    # Takes a question id and returns the question type from the question types table by joining the questions table
    # appropriately
    sql: str = 'SELECT question_types.type FROM question_types INNER JOIN questions
    \n        ON question_types.id = questions.type_id WHERE questions.id = ?'
    c.execute(sql, (question_id,))
    return c.fetchall()[0][0]
```

4.5.29.2 Testing

4.5.29.2.1 SQL

The figure consists of three vertically stacked screenshots of DB Browser for SQLite, version 5.4.1, running against a database named 'database.db' located at C:\Users\Joel\PycharmProjects\Coursework\script\database.db.

- Screenshot 1:** Shows the 'questions' table. The table has columns: id, name, type_id, question_text, correct_answer, answer_a, answer_b, answer_c, and answer_d. A single row is selected: id=59, name='question59', type_id=16, question_text='question text 59', correct_answer='b 59', answer_a='c 59', answer_b='d 59', answer_c='', and answer_d=''. The 'Edit Database Cell' dialog is open over the 'correct_answer' field, showing its current value as 'NULL' and a 'Type of data currently in cell: NULL' message.
- Screenshot 2:** Shows the 'question_types' table. The table has columns: id and type. A single row is selected: id=16 and type='type16'. The 'Edit Database Cell' dialog is open over the 'type' field, showing its current value as 'NULL' and a 'Type of data currently in cell: NULL' message.
- Screenshot 3:** Shows the results of a SQL query in the 'Execute SQL' tab. The query is: 'SELECT question_types.type FROM question_types INNER JOIN questions ON question_types.id = questions.type_id WHERE questions.id = 59'. The result is a single row: type='type16'. The 'Edit Database Cell' dialog is open over the 'type' field, showing its current value as 'NULL' and a 'Type of data currently in cell: NULL' message.

Correctly returns correct question type (type16) for given question id (16)

4.5.29.2.2 Test code

```
if __name__ == '__main__':
    qid: int = input("Enter question id: ")
    print(get_question_type(78))
```

4.5.29.2.3 With valid data:

The screenshot shows two windows of DB Browser for SQLite. The top window displays the 'questions' table with one record (id: 78, name: 'question78', type_id: 90, question_text: 'question text 78', correct_answer: 'correct answer...', answer_b: 'b 78', answer_c: 'c 78', answer_d: 'd 78'). The bottom window displays the 'question_types' table with one record (id: 90, type: 'type90'). Both windows show an 'Edit Database Cell' dialog box with the message 'Type of data currently in cell: NULL'.

C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe
C:/Users/Joe/PycharmProjects/Coursework/scripts/ui_scripts.py

Enter question id: 78

type90

Process finished with exit code 0

Correctly returns correct question type (type90) for given question id (78)

4.5.29.2.4 With Invalid data:

N/A, any errors should be addressed before calling function by verifying question id exists and is a valid integer before calling function

4.5.29.2.5 Evaluation

Correctly returns correct question type for given question id

4.5.30 Function to get graph equation for a question

4.5.30.1 Final Code

```
def get_question_graph(question_id: int) -> list:
    # Takes a question id and returns the function, minimum x and maximum x values
    from the graphs table
    sql: str = 'SELECT function, minimum_x, maximum_x FROM graphs WHERE question_id = ?'
    c.execute(sql, (question_id,))
    result = c.fetchall()
    # If no graph is stored for the given question, nothing is returned
    if result:
        return result[0]
    else:
        return []
```

4.5.30.2 Testing

4.5.30.2.1 SQL

The screenshot shows the DB Browser for SQLite interface. On the left, there's a tree view labeled 'Database Structure' with a single node 'graphs'. To the right, a table named 'graphs' is displayed with the following data:

id	question_id	function	minimum_x	maximum_x
1	100160	$20x + 3$	4.0	6.0

Below the table, it says '1 row(s) found'.

The screenshot shows the DB Browser for SQLite interface. On the left, there's a tree view labeled 'Database Structure' with a single node 'graphs'. To the right, the 'SQL' tab is active, showing the following query:

```
SELECT function, minimum_x, maximum_x FROM graphs WHERE question_id = 100160
```

Below the query, the results are displayed in a table:

function	minimum_x	maximum_x
$20x + 3$	4.0	6.0

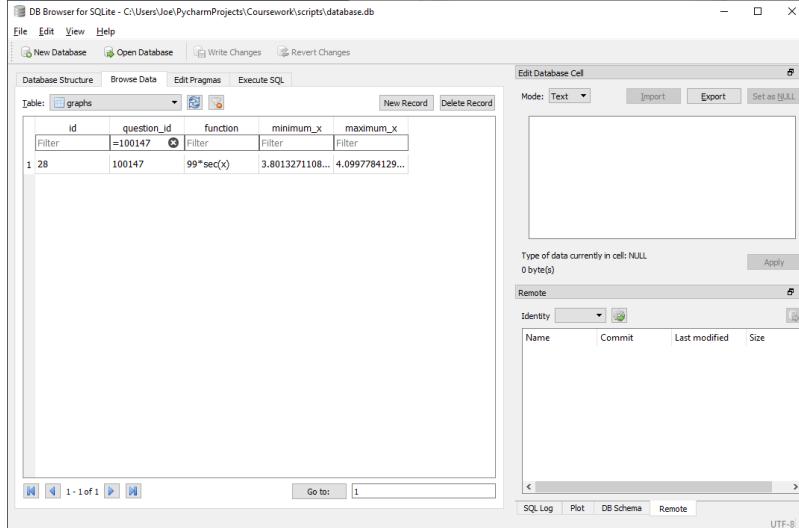
At the bottom, it says '1 row(s) returned in 0ms from: SELECT function, minimum_x, maximum_x FROM graphs WHERE question_id = 100160'.

Correctly returns matching function ($20x + 3$), minimum x (4) and maximum x (6) for given question id (100160)

4.5.30.2.2 Test code

```
if __name__ == '__main__':
    qid: int = input("Enter question id: ")
    print(get_question_type(qid))
```

4.5.30.2.3 With valid data:



C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe
C:/Users/Joe/PycharmProjects/Coursework/scripts/ui_scripts.py

Enter question id: 100147

('99*sec(x)', 3.8013271108436495, 4.09977841293468)

Process finished with exit code 0

Correctly returns matching function (99sec(x)), minimum x (3.801...) and maximum x (4.099...) for given question id (100147)

4.5.30.2.4 With Invalid data:

N/A, any errors should be addressed before calling function by verifying question id exists and is a valid integer before calling function

4.5.30.2.5 Evaluation

Correctly returns matching function, minimum x and maximum x for given question id

4.5.31 Function to set graph equation for a question

4.5.31.1 Final Code

```
def set_question_graph(question_id: int, function: str, minimum_x: float, maximum_x: float):
    # Inserts required data (question id, function, minimum x, maximum x) into
    # graphs table to create new graph
    # instance for the given question id
    sql: str = 'INSERT INTO graphs(question_id, function, minimum_x, maximum_x)
VALUES (?, ?, ?, ?)'
    c.execute(sql, (question_id, function, minimum_x, maximum_x))
    conn.commit()
```

4.5.31.2 Testing

4.5.31.2.1 SQL

The figure consists of three vertically stacked screenshots of DB Browser for SQLite, showing the process of inserting data into the 'graphs' table.

Screenshot 1: Shows the 'Edit Database Cell' dialog open over the 'graphs' table. The table has columns: id, question_id, function, minimum_x, maximum_x. A single row is selected with id=6, question_id=6, function='x', minimum_x=1.0, and maximum_x=2.0. The 'Mode' dropdown is set to 'Text'. The 'Edit Database Cell' dialog shows the current value 'NULL' and a note 'Type of data currently in cell: NULL 0 byte(s)'. The 'Remote' tab is selected in the bottom right.

Screenshot 2: Shows the 'Execute SQL' tab with the following SQL query entered:

```
1 INSERT INTO graphs(question_id, function, minimum_x, maximum_x) VALUES(6, 'x', 1, 2)
```

The results pane shows the inserted row:

	function	minimum_x	maximum_x
1	20*x + 3	4.0	6.0

A message at the bottom indicates the query was executed successfully: "Query executed successfully: INSERT INTO graphs(question_id, function, minimum_x, maximum_x) VALUES(6, 'x', 1, 2) (took 0ms, 1 rows affected)".

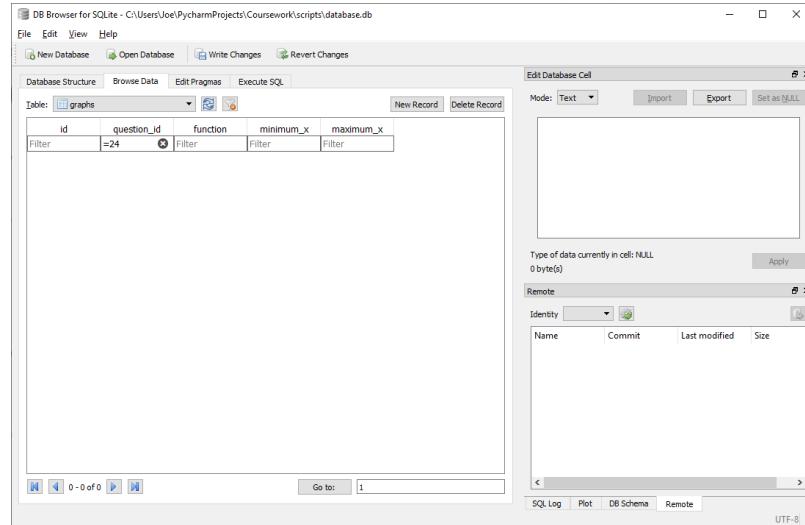
Screenshot 3: Shows the 'Browse Data' tab with the 'graphs' table. The same row is now visible with id=6, question_id=6, function='x', minimum_x=1.0, and maximum_x=2.0.

Correctly inserts given question id (6), function (x), minimum x (1) and maximum x (2) into graphs table

4.5.31.2.2 Test code

```
if __name__ == '__main__':
    qid: int = input("Enter question id: ")
    function = input("Enter function: ")
    minx = input("Enter min x: ")
    maxx = input("Enter max x: ")
    set_question_graph(qid, function, minx, maxx)
```

4.5.31.2.3 With valid data:



C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe
C:/Users/Joe/PycharmProjects/Coursework/scripts/ui_scripts.py

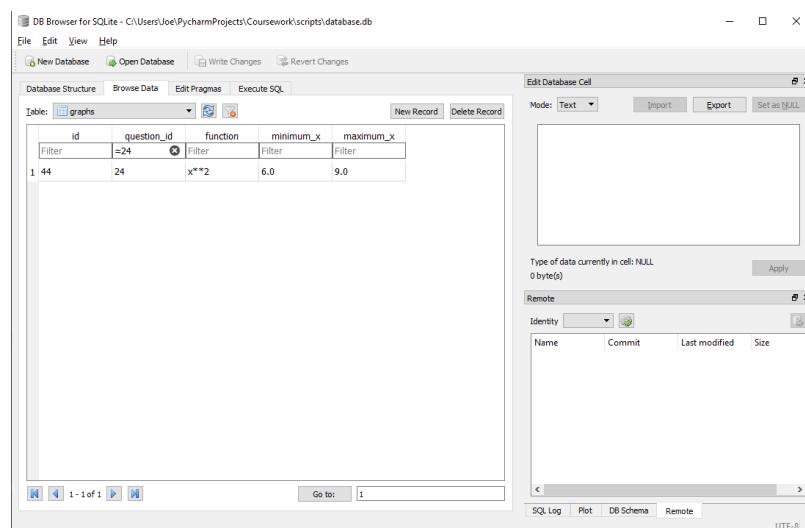
Enter question id: 24

Enter function: x2**

Enter min x: 6

Enter max x: 9

Process finished with exit code 0



Correctly inserts given question id (24), function (x^{**2}), minimum x (6) and maximum x (9) into graphs table

4.5.31.2.4 With Invalid data:

N/A, any errors should be addressed before calling function by verifying question id exists and is a valid integer before calling function, function is of valid from and minimum and maximum x are both floats with minimum x less than maximum x

4.5.31.2.5 Evaluation

Correctly inserts given question id, function, minimum x and maximum x into graphs table

4.6 UI PROGRAMMING (MAIN)

4.6.1 Function to initialise program UI and halt program

4.6.1.1 Final Code

```
# Imports required modules for UI programming and control (no missing module validation required as all modules will be included when .exe created)
import datetime
import random
import sqlite3
import sys

import sympy
from PyQt5 import QtCore, QtChart
from PyQt5 import QtWidgets

import questions.calculus
import questions.mechanics
import questions.question_scripts as question_scripts
import window
from scripts import db_scripts, ui_scripts

if __name__ == '__main__':
    # Sets up the database to avoid errors due to missing or incorrectly structured database
    create_database()
    # Sets up UI and initialises program
    app = QtWidgets.QApplication(sys.argv)
    window = Window()
    window.show()
    # Ends program when close button selected
    sys.exit(app.exec_())
```

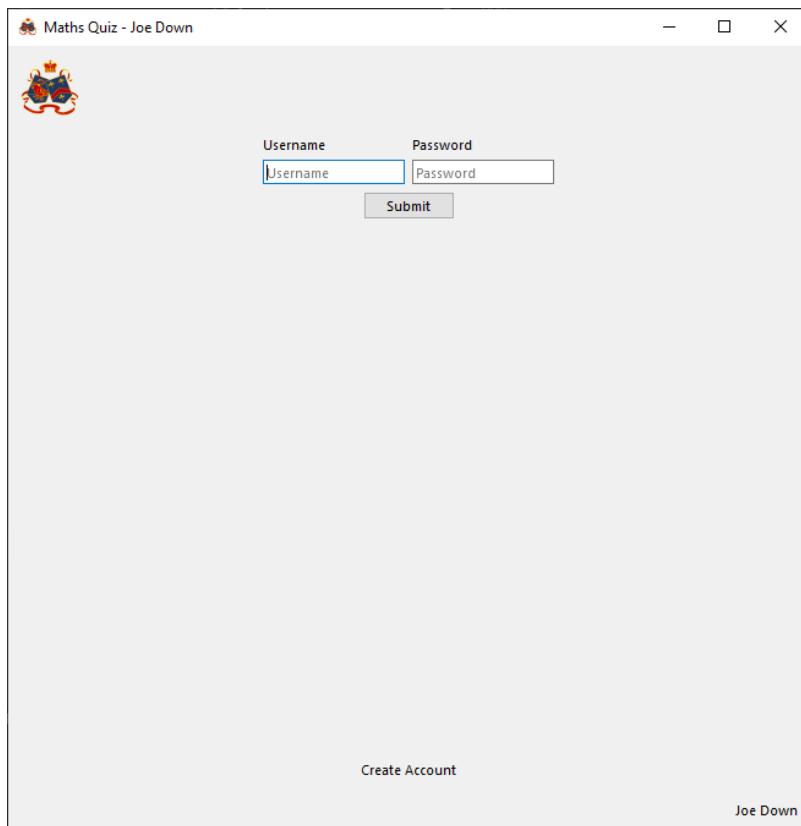
4.6.1.2 Testing

Notes:

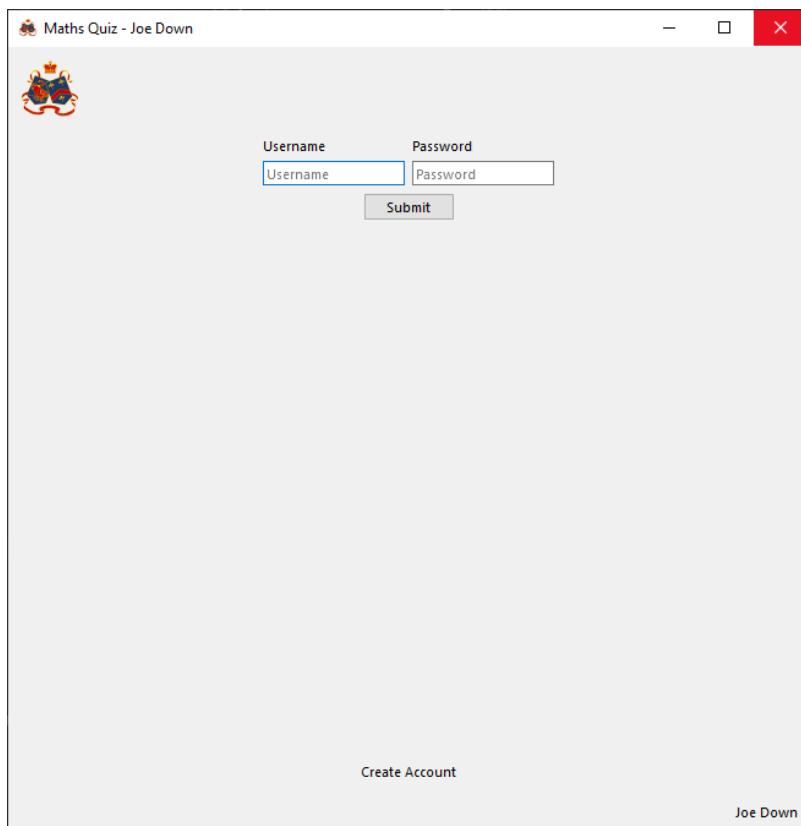
- Create database function and testing will be shown later
- Windows class code setup and testing will be shown later
- Code used is generic to all PyQT programs so can be assumed to be well tested, so it will only be checked that the code has been implemented correctly

4.6.1.2.1 With valid data:

When program is run correctly shows the program window:



When close button pressed, correctly ends program execution:



```
C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe
C:/Users/Joe/PycharmProjects/Coursework/main.py
```

Process finished with exit code 0

4.6.1.2.2 With Invalid data:

N/A, no user input required, and this code can only ever execute in the same way

4.6.1.2.3 Evaluation

Correctly initialises UI and ends program when UI closed. No errors occur during the imports as python environment is correctly set up

4.6.2 Function to setup database or, if already present, make sure database structure is correct

4.6.2.1 *Issues during development*

Initially, the following code was not included:

```
sql: str = 'CREATE TABLE IF NOT EXISTS "graphs" ( `id` INTEGER NOT NULL PRIMARY KEY
AUTOINCREMENT UNIQUE, ' \
        '`question_id` INTEGER NOT NULL UNIQUE, `function` TEXT NOT NULL, `mini-
mum_x` REAL NOT NULL, ' \
        '`maximum_x` REAL NOT NULL, FOREIGN KEY(`question_id`) REFERENCES `ques-
tions`(`id`)''
c.execute(sql, ())
```

This is because this table was not specified to be included as part of the data structure during design. However, during development, it was decided to implement graphs for questions containing a graphable mathematical function of some sort and so this graph equation would need to be stored somewhere, along with the range of x values for which this graph should be drawn. This was done by creating a new graphs table with a question id as a foreign key identifying to what questions graph belongs. This way, the questions table does not need to be changed and so code existing code using this table does not need to be retested. To get a graph, the graphs table can simply be searched for the question id. If there are no occurrences in the table, there is no graph, and if there is a record found, the relevant graph equation and boundaries can be utilised in the graph drawing code.

4.6.2.2 Final Code

```

def create_database():
    # Generic code to open and connect to SQL database. If file is not found a new
    file is created
    database_name: str = 'database.db'
    conn = sqlite3.connect(database_name)
    c = conn.cursor()
    # SQL code below generated by SQLite browser when creating database within it's
    UI. Code will create all tables as
    # defined if found to be missing at program startup and establishes all field
    relationships
    sql: str = 'CREATE TABLE IF NOT EXISTS "class_homework" ' \
        '(`id` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE, '
    `class_id` INTEGER NOT NULL, ' \
        '`homework_id` INTEGER NOT NULL, `due_date` TEXT NOT NULL, ' \
        'FOREIGN KEY(`class_id`) REFERENCES `classes`(`id`), ' \
        'FOREIGN KEY(`homework_id`) REFERENCES `homework`(`id`));'
    c.execute(sql, ())
    sql: str = 'CREATE TABLE IF NOT EXISTS "class_user" (`id` INTEGER NOT NULL
    PRIMARY KEY AUTOINCREMENT UNIQUE, ' \
        '`class_id` INTEGER NOT NULL, `student_id` INTEGER NOT NULL, ' \
        'FOREIGN KEY(`class_id`) REFERENCES `classes`(`id`), ' \
        'FOREIGN KEY(`student_id`) REFERENCES `users`(`id`))'
    c.execute(sql, ())
    sql: str = 'CREATE TABLE IF NOT EXISTS "classes" (`id` INTEGER NOT NULL PRI-
    MARY KEY AUTOINCREMENT UNIQUE, ' \
        '`name` TEXT NOT NULL, `teacher` INTEGER NOT NULL, FOREIGN
    KEY(`teacher`) REFERENCES `users`(`id`))'
    c.execute(sql, ())
    sql: str = 'CREATE TABLE IF NOT EXISTS "graphs" (`id` INTEGER NOT NULL PRIMARY
    KEY AUTOINCREMENT UNIQUE, ' \
        '`question_id` INTEGER NOT NULL UNIQUE, `function` TEXT NOT NULL,
    `minimum_x` REAL NOT NULL, ' \
        '`maximum_x` REAL NOT NULL, FOREIGN KEY(`question_id`) REFERENCES
    `questions`(`id`))'
    c.execute(sql, ())
    sql: str = 'CREATE TABLE IF NOT EXISTS "homework" (`id` INTEGER NOT NULL PRI-
    MARY KEY AUTOINCREMENT UNIQUE, ' \
        '`name` TEXT NOT NULL, `description` TEXT )'
    c.execute(sql, ())
    sql: str = 'CREATE TABLE IF NOT EXISTS "homework_questions" (' \
        '`id` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE, `home-
    work_id` INTEGER NOT NULL, ' \
        '`question_id` INTEGER NOT NULL, FOREIGN KEY(`homework_id`) REFER-
    ENCES `homework`(`id`), ' \
        'FOREIGN KEY(`question_id`) REFERENCES `questions`(`id`))'
    c.execute(sql, ())
    sql: str = 'CREATE TABLE IF NOT EXISTS "question_results" ' \
        '(`id` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE, `user_id` '
    INTEGER NOT NULL, ' \
        '`question_id` INTEGER NOT NULL, `attempts` INTEGER NOT NULL, `cor-
    rect` TEXT NOT NULL, ' \
        'FOREIGN KEY(`question_id`) REFERENCES `questions`(`id`), ' \
        'FOREIGN KEY(`user_id`) REFERENCES `users`(`id`))'
    c.execute(sql, ())
    sql: str = 'CREATE TABLE IF NOT EXISTS "question_types" ' \
        '(`id` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE, `type` '
    TEXT NOT NULL UNIQUE )'
    c.execute(sql, ())
    sql: str = 'CREATE TABLE IF NOT EXISTS "questions" (`id` INTEGER NOT NULL PRI-
    MARY KEY AUTOINCREMENT UNIQUE, ' \
        '`name` TEXT NOT NULL, `type_id` INTEGER NOT NULL, `question_text` '
    TEXT NOT NULL, ' \
        '`correct_answer` TEXT NOT NULL, `answer_b` TEXT NOT NULL, `an-
    swer_c` TEXT NOT NULL, ' \
        '`answer_d` TEXT NOT NULL, FOREIGN KEY(`type_id`) REFERENCES `ques-
    tion_types`(`id`))'
    c.execute(sql, ())

```

```

sql: str = 'CREATE TABLE IF NOT EXISTS "users" ( `id` INTEGER NOT NULL PRIMARY
KEY AUTOINCREMENT UNIQUE, ' \
    ' `username` TEXT NOT NULL UNIQUE, `password_salt` TEXT NOT NULL
UNIQUE, ' \
    ' `password_hash` TEXT NOT NULL, `first_name` TEXT NOT NULL,
`last_name` TEXT NOT NULL, ' \
    ' `type` TEXT NOT NULL )'
c.execute(sql, ())
# Inserts entries into question types table defining the topic names related to
each type id. If there is already
# an entry in the table for the given id, the topic name is made sure to be
correct and changed if needed. To ease
# development, topic ids and names are stored as tuples in a list which is it-
erated through when inserting data,
# meaning new topics can simply be added to the list as they are added to the
program and the database will be
# automatically changed accordingly when the program is next run
sql1: str = 'INSERT INTO question_types(id, type) VALUES(?, ?)'
sql2: str = 'UPDATE question_types SET type = ? WHERE id = ?'
data = [[1, 'Custom'], [2, 'Find resultant of two forces'], [3, "Simpson's
Rule"], [4, 'Trapezium Rule'],
[5, 'Definite Integral'], [6, 'Projectile']]
for topic in data:
    # If topic not already in database, inserts topic
    try:
        c.execute(sql1, (topic[0], topic[1]))
    # If topic already in database, ensures topic name is correct and updates
accordingly
    except sqlite3.IntegrityError:
        c.execute(sql2, (topic[1], topic[0]))
    # Permanently applies all database changes made
conn.commit()

```

4.6.2.3 Testing

4.6.2.3.1 With valid data:

4.6.2.3.1.1 Database does not already exist

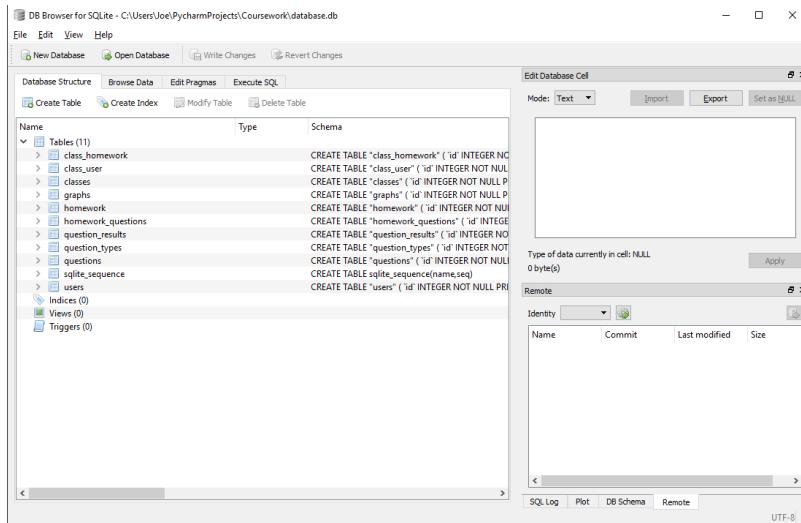
4.6.2.3.1.1.1 Before program is run:

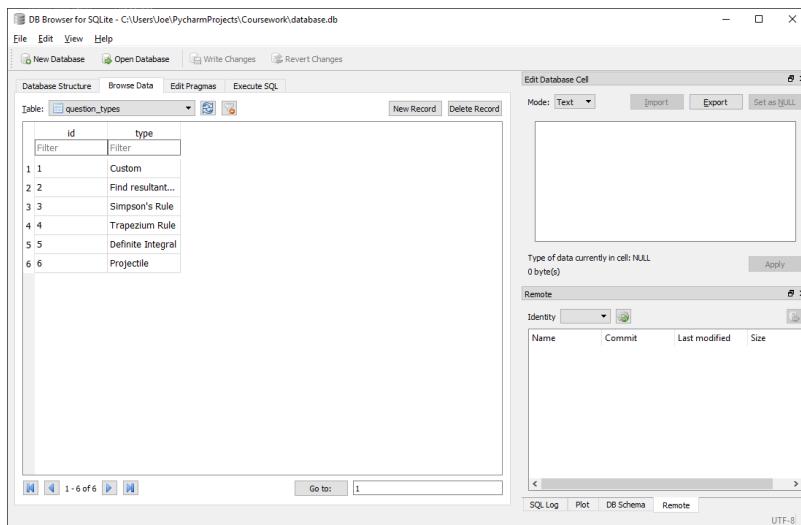
📁 .git	23/03/2019 12:59	File folder
📁 .idea	23/03/2019 13:01	File folder
📁 __pycache__	23/03/2019 12:44	File folder
📁 build	16/03/2019 12:34	File folder
📁 dist	19/03/2019 23:28	File folder
📁 maths_scripts	18/03/2019 18:45	File folder
📁 questions	18/03/2019 17:19	File folder
📁 scripts	20/03/2019 23:46	File folder
📁 testing	16/03/2019 00:34	File folder
📁 venv	12/03/2019 18:51	File folder
📁 window_elements	19/03/2019 23:09	File folder
📄 .gitignore	12/03/2019 16:21	Text Document 2 KB
⚙️ compile.bat	16/03/2019 12:34	Windows Batch File 1 KB
⚙️ databaseold.db	23/03/2019 12:44	Data Base File 44,572 KB
🐍 file_prep.py	16/03/2019 11:48	Python File 1 KB
.ico icon.ico	19/03/2019 23:08	Icon 169 KB
LICENSE	12/03/2019 16:21	File 2 KB
🐍 main.py	23/03/2019 12:59	Python File 62 KB
📄 MathsQuiz.spec	19/03/2019 23:11	SPEC File 1 KB
README.md	12/03/2019 16:21	MD File 1 KB
🐍 window.py	19/03/2019 23:11	Python File 86 KB
window.qrc	16/03/2019 11:57	QRC File 1 KB
window.ui	19/03/2019 23:11	UI File 89 KB
🐍 window_rc.py	19/03/2019 23:11	Python File 3,547 KB

4.6.2.3.1.1.2 After program is run:

.git	23/03/2019 12:59	File folder
.idea	23/03/2019 13:01	File folder
__pycache__	23/03/2019 12:44	File folder
build	16/03/2019 12:34	File folder
dist	19/03/2019 23:28	File folder
maths_scripts	18/03/2019 18:45	File folder
questions	18/03/2019 17:19	File folder
scripts	20/03/2019 23:46	File folder
testing	16/03/2019 00:34	File folder
venv	12/03/2019 18:51	File folder
window_elements	19/03/2019 23:09	File folder
.gitignore	12/03/2019 16:21	Text Document 2 KB
compile.bat	16/03/2019 12:34	Windows Batch File 1 KB
database.db	23/03/2019 13:13	Data Base File 104 KB
databaseold.db	23/03/2019 12:44	Data Base File 44,572 KB
file_prep.py	16/03/2019 11:48	Python File 1 KB
icon.ico	19/03/2019 23:08	Icon 169 KB
LICENSE	12/03/2019 16:21	File 2 KB
main.py	23/03/2019 12:59	Python File 62 KB
MathsQuiz.spec	19/03/2019 23:11	SPEC File 1 KB
README.md	12/03/2019 16:21	MD File 1 KB
window.py	19/03/2019 23:11	Python File 86 KB
window.qrc	16/03/2019 11:57	QRC File 1 KB
window.ui	19/03/2019 23:11	UI File 89 KB
window_rc.py	19/03/2019 23:11	Python File 3,547 KB

Correctly creates database.db file

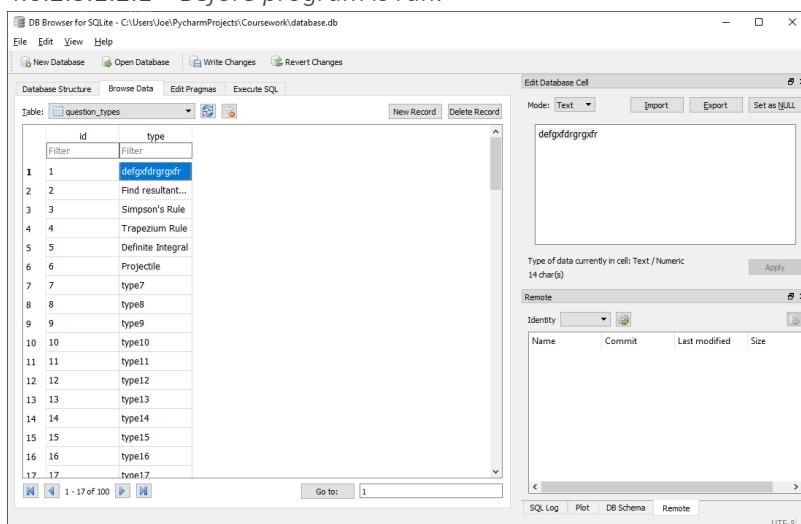




Created database has all required tables and table relations. All predefined data is included as needed

4.6.2.3.1.2 Database already exists

4.6.2.3.1.2.1 Before program is run:



Database contains existing data and has a mistake in the type name for type with id 1 (should be 'Custom')

Database contains existing data

4.6.2.3.1.2.2 After program is run

The screenshot shows the DB Browser for SQLite interface. On the left, a table named 'question_types' is displayed with columns 'id' and 'type'. The data rows are:

id	type
1	Custom
2	Find resultant...
3	Simpson's Rule
4	Trapezium Rule
5	Definite Integral
6	Projectile
7	type7
8	type8
9	type9
10	type10
11	type11
12	type12
13	type13
14	type14
15	type15
16	type16
17	type17

On the right, there are several panes: 'Edit Database Cell' (Mode: Text), 'Import' and 'Export' buttons, a 'Remote' pane (Identity dropdown), and tabs for 'SQL Log', 'Plot', 'DB Schema', and 'Remote'. The status bar at the bottom indicates 'UTF-8' encoding.

All data already in database is

preserved and incorrect data fixed (type with id 1 changed to 'Custom')

4.6.2.3.2 With Invalid data:

N/A, no user input required, and this code can only ever execute in the same way

4.6.2.3.3 Evaluation

Successfully creates the relevant database file if not present with all required table relations and pre-inserted data or, if the database already exists, ensures it fits the expected database structure and changes it as/if needed

4.6.3 Window class

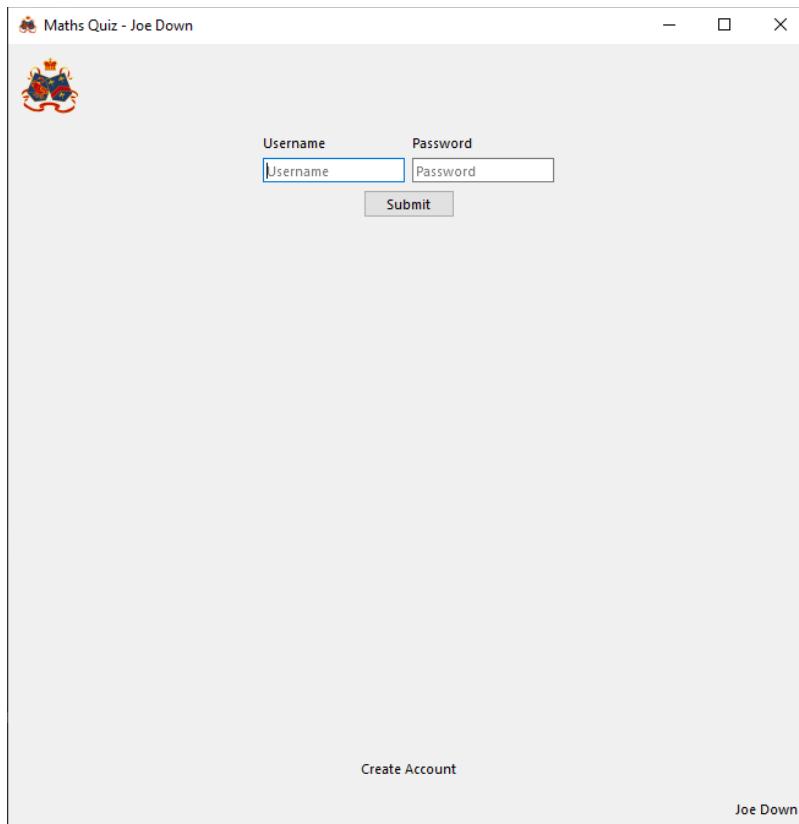
4.6.3.1 Function to initialise class

4.6.3.1.1 Final Code

```
# Window class to control ui, inheriting from the QMainWindow class from PyQt
class Window(QtWidgets.QMainWindow, window.Ui_MainWindow):
    # <editor-fold desc="General">
    # noinspection PyArgumentList
    def __init__(self):
        # Inherits from generic window class from QT
        super().__init__()
        self.setupUi(self)
        # Declare constants related to page indexes of different sections of the
        program as references to the integer
        # page values in a dictionary
        self.page_dictionary: dict = {'login_page': 0, 'create_account_page': 1,
        'student_main_menu_page': 2, 'teacher_main_menu_page': 3, 'question_page': 4,
        'homework_select_page': 5, 'previous_scores_page': 6, 'set_home-
        work_page': 7, 'admin_page': 8, 'account_management_page': 9, 'view_clas-
        ses_page': 10}
        # Sets up all page buttons and layouts and goes to login page
        self.button_setup()
        self.reset_page(self.page_dictionary['login_page'])
        # Sets program to logged out state (and hides logged out message)
        self.logout()
        self.login_success_output.setText("")
        # Holds ID of active user (set to -1 initially as no user logged in)
        self.current_user: int = -1
        # Clears list of user classes
        self.current_classes: list = []
        # Holds list of members of a class when needed to be stored
        self.class_users: list = []
        # Holds homework data when needed to be stored
        self.homework: list = []
        # Holds question data when needed to be stored
        self.questions: list = []
        # Holds homework ids when needed to be stored
        self.homework_ids: list = []
        # Holds the id of the current question number while quiz being done to keep
        track of position within question
        # list
        self.current_question: int = 0
        # Holds the location of the correct answer during a quiz (set to 1 ini-
        tially as it will be changed when quiz
        # page loaded)
        self.correct_answer_location: int = 1
        # Holds classes or students of currently selected class on the view classes
        page
        self.view_classes_students_or_homework: list = []
```

4.6.3.1.2 Testing

4.6.3.1.2.1 With valid data:



Correctly initialises program at login page with logout and main menu buttons hidden (sets program to logged out state)

4.6.3.1.2.2 With Invalid data:

N/A, no user input required, and this code can only ever execute in the same way

4.6.3.1.2.3 Evaluation

Correctly initialises program at login page with logout and main menu buttons hidden (sets program to logged out state)

4.6.3.2 Function to change page to a given index and update logout button, main menu button and name display accordingly

4.6.3.2.1 Issues during development

Initially the following code was implemented:

```

def change_page(self, index: int):
    # Restores target page to default state
    self.reset_page(index)
    # Enables all navigation buttons to then be disabled as needed
    self.show_main_menu_button()
    self.show_logout_button()
    self.show_username_text()
    # If a logged out page or menu page, hides main menu button
    if index in [0, 1, 2, 3]:
        self.hide_main_menu_button()
        # If a logged out page, hide logout button
        if index in [0, 1]:
            self.hide_logout_button()
            self.hide_username_text()
    # Changes the current page index to the value passed
    self.main_widget.setCurrentIndex(index)

```

This resulted in the following error upon running the program:

The screenshot shows the PyCharm IDE's 'Run' tab with a terminal window. The terminal output displays a stack trace starting from the 'main' function, leading down to the 'change_page' method, and finally to the line where the error occurred: 'File "C:/Users/Joe/PycharmProjects/Coursework/main.py", line 140, in change_page'. The error message 'IndexError: list index out of range' is visible at the bottom of the stack trace.

```

Run: main () ×
C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe C:/Users/Joe/PycharmProjects/Coursework/main.py
Traceback (most recent call last):
  File "C:/Users/Joe/PycharmProjects/Coursework/main.py", line 1188, in <module>
    window = Window()
  File "C:/Users/Joe/PycharmProjects/Coursework/main.py", line 109, in __init__
    self.logout()
  File "C:/Users/Joe/PycharmProjects/Coursework/main.py", line 242, in logout
    self.change_page(self.page_dictionary['login_page'])
  File "C:/Users/Joe/PycharmProjects/Coursework/main.py", line 140, in change_page
    self.show_username_text()
  File "C:/Users/Joe/PycharmProjects/Coursework/main.py", line 205, in show_username_text
    self.username_label.setText(db_scripts.get_first_name(self.current_user).title())
  File "C:/Users/Joe/PycharmProjects/Coursework/scripts\db_scripts.py", line 98, in get_first_name
    return c.fetchall()[0][0]
IndexError: list index out of range

Process finished with exit code 1

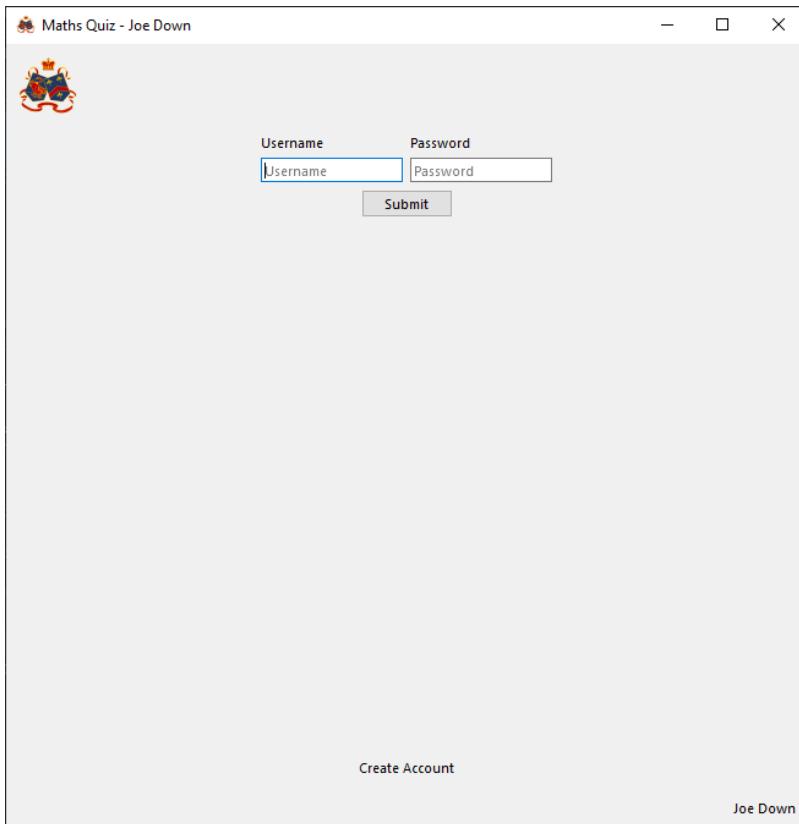
```

This is because when initially loading the login page, an attempt is made to display the current username. As when a user isn't logged in, the current user id is set to -1, there is no corresponding username in the database and so a crash occurs. This will be fixed by checking if the current user id is -1 before attempting to show the username. If it is -1, it is known no user is logged in at the time and so the username display is not needed. The code was changed to include the following:

```

# Only tries to show username if logged in
if self.current_user != -1:
    self.show_username_text()

```



The program now loads correctly, displaying no username until the user logs in

4.6.3.2.2 Final Code

```
def change_page(self, index: int):
    # Restores target page to default state
    self.reset_page(index)
    # Enables all navigation buttons to then be disabled as needed
    self.show_main_menu_button()
    self.show_logout_button()
    # Only tries to show username if logged in
    if self.current_user != -1:
        self.show_username_text()
    # If a logged out page or menu page, hides main menu button
    if index in [0, 1, 2, 3]:
        self.hide_main_menu_button()
        # If a logged out page, hide logout button
        if index in [0, 1]:
            self.hide_logout_button()
            self.hide_username_text()
    # Changes the current page index to the value passed
    self.main_widget.setCurrentIndex(index)
```

4.6.3.2.3 Testing

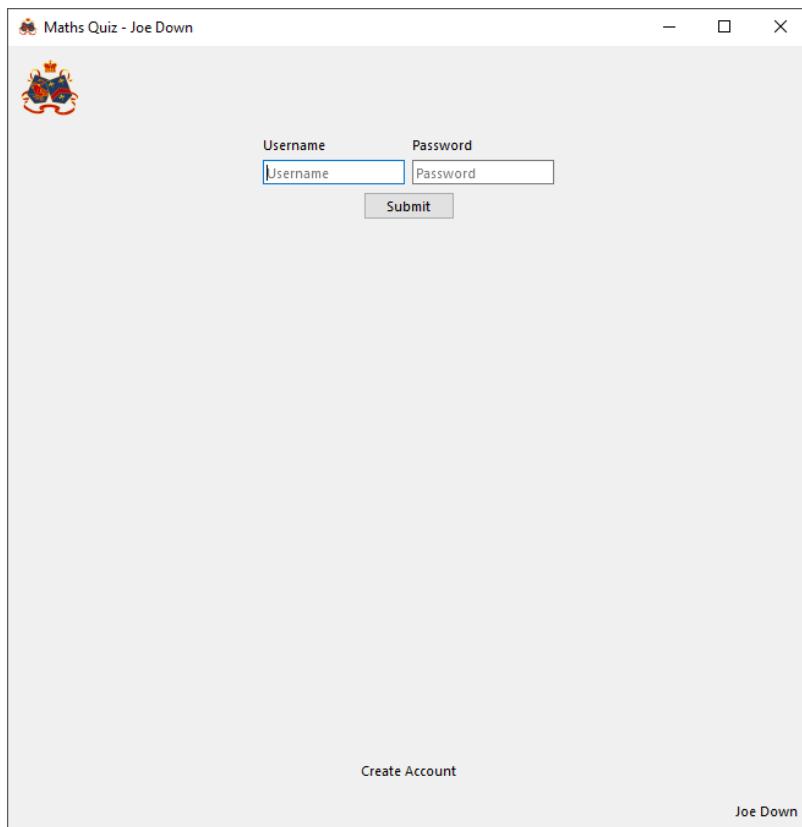
Notes:

- Reset page function testing shown later in development
- Show main menu button function testing shown later in development
- Show logout button function testing shown later in development
- Show username text function testing shown later in development
- Hide main menu button function testing shown later in development
- Hide logout button function testing shown later in development
- Hide username text function testing shown later in development

4.6.3.2.3.1 With valid data:

4.6.3.2.3.1.1 Load page 0

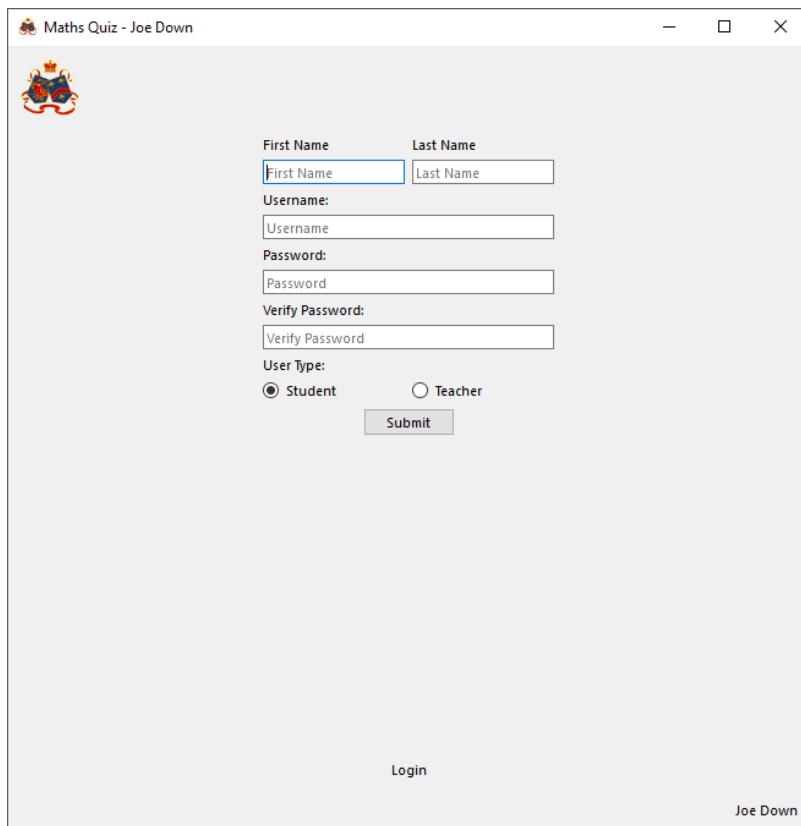
```
window.change_page(0)
```



Loads correct screen and hides username, main menu button and logout button

4.6.3.2.3.1.2 Load page 1

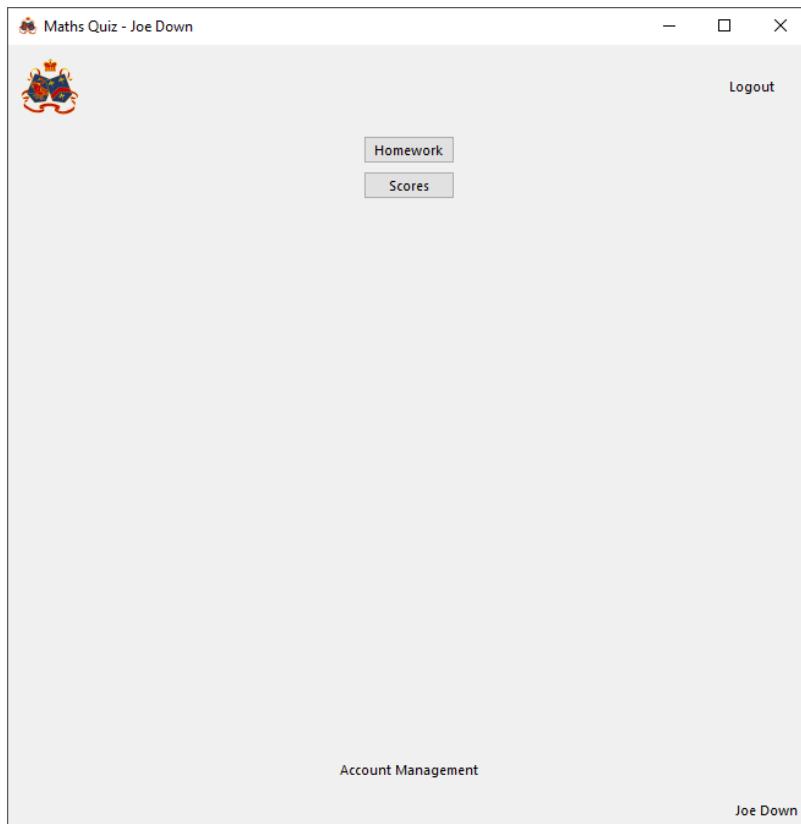
```
window.change_page(1)
```



Loads correct screen and hides username, main menu button and logout button

4.6.3.2.3.1.3 Load page 2

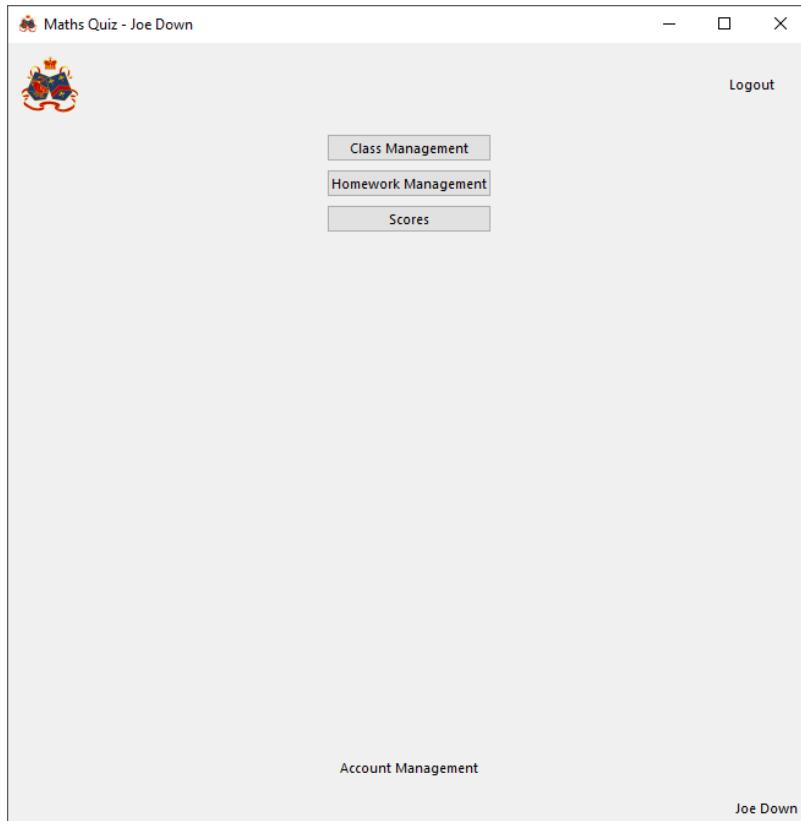
```
window.change_page(2)
```



Loads correct screen and shows username and logout button

4.6.3.2.3.1.4 Load page 3

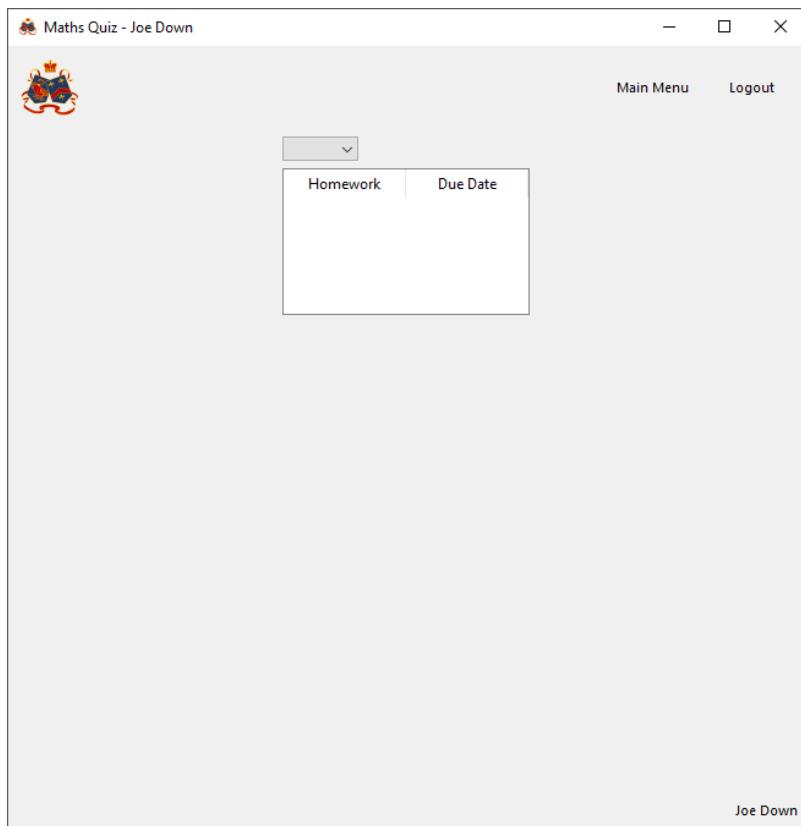
```
window.change_page(3)
```



Loads correct screen and shows username and logout button

4.6.3.2.3.1.5 Load page 5

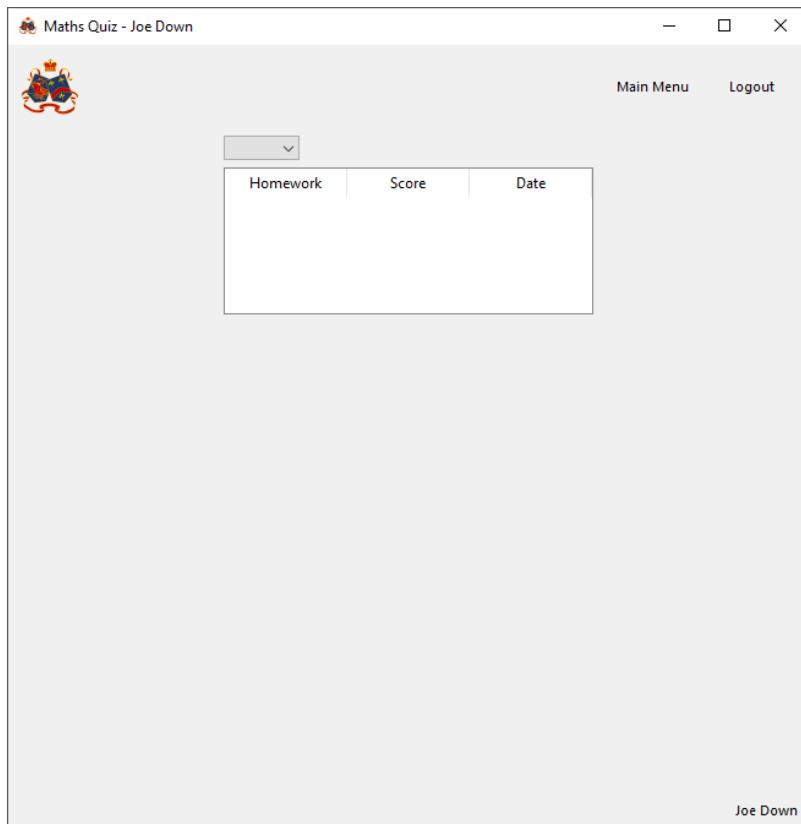
```
window.change_page(5)
```



Loads correct screen and shows username, main menu button and logout button

4.6.3.2.3.1.6 Load page 6

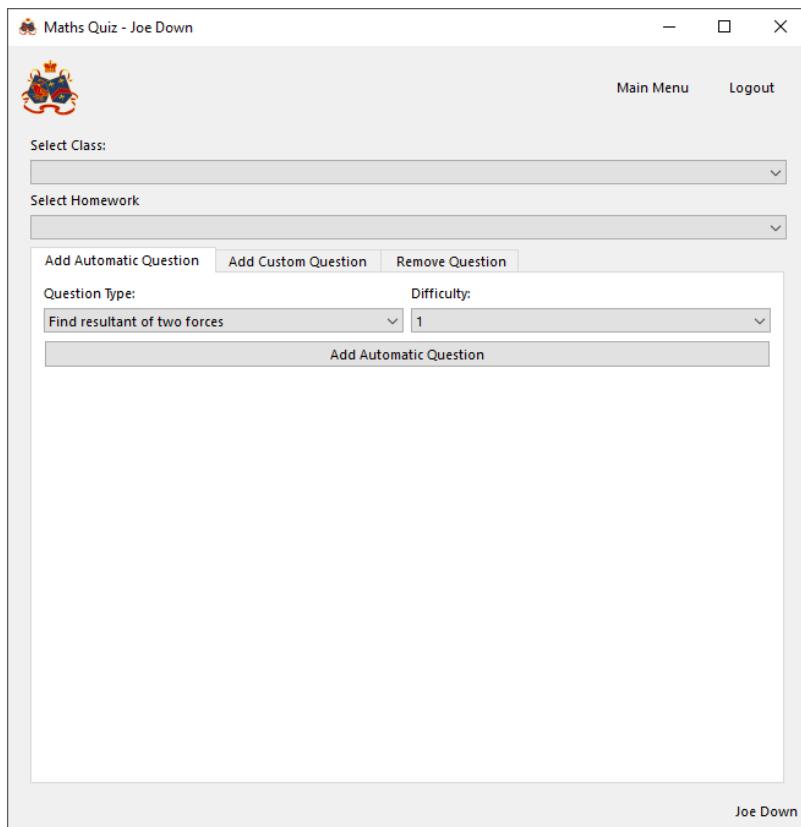
```
window.change_page(6)
```



Loads correct screen and shows username, main menu button and logout button

4.6.3.2.3.1.7 Load page 7

```
window.change_page(7)
```



Loads correct screen and shows username, main menu button and logout button

4.6.3.2.3.1.8 Load page 8

```
window.change_page(8)
```

 Maths Quiz - Joe Down

Main Menu Logout

Create Class:

Class Name Submit

Manage Class:

Add User Remove User Add Homework Remove Homework

Username Submit

Delete Class:

▼ REMOVE

Joe Down

Loads correct screen and shows username, main menu button and logout button

4.6.3.2.3.1.9 Load page 9

```
window.change_page(9)
```

 Maths Quiz - Joe Down

Main Menu Logout

New First Name New Last Name

First Name Last Name

Current Password:

Current Password

New Password

New Password

Verify New Password:

Verify New Password

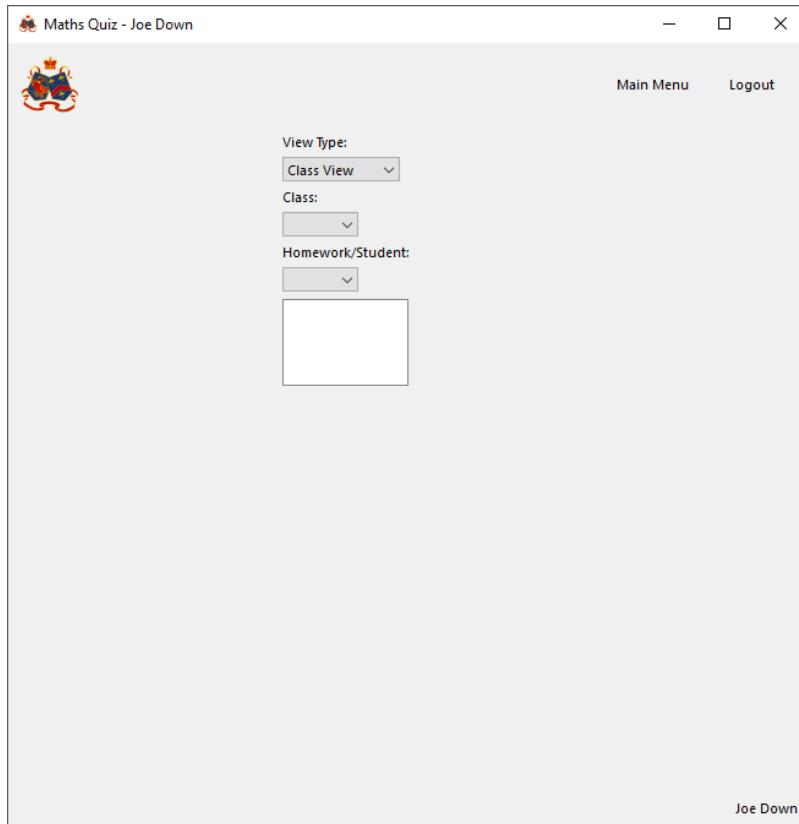
Submit

Joe Down

Loads correct screen and shows username, main menu button and logout button

4.6.3.2.3.1.10 Load page 10

```
window.change_page(10)
```



Loads correct screen and shows username, main menu button and logout button

4.6.3.2.3.2 With Invalid data:

N/A, page index should be validated to exist and be an integer before calling function

4.6.3.2.3.3 Evaluation

Correctly changes to the correct page index and sets it to its initial state. Correctly hides/shows logout and main menu buttons and user's name depending on the page selected

4.6.3.3 Function to set up buttons

4.6.3.3.1 Final Code

```
def button_setup(self):
    # Runs all button setups
    self.navigation_button_setup()
    self.login_page_button_setup()
    self.create_account_page_button_setup()
    self.student_main_menu_page_button_setup()
    self.teacher_main_menu_page_button_setup()
    self.previous_scores_page_button_setup()
    self.admin_page_button_setup()
    self.account_management_page_button_setup()
    self.view_classes_page_button_setup()
    self.set_homework_page_button_setup()
    self.homework_select_page_button_setup()
    self.question_page_button_setup()
```

4.6.3.3.2 Testing

As this function simply calls each button setup function when the program is run, it can be tested later by simply checking buttons work as intended after writing the button setup functions for each page. No inputs are taken and so the script will always run the same way and so testing for incorrect usage does not need to be done

4.6.3.4 Function to reset a page

4.6.3.4.1 Final Code

```
def reset_page(self, target_page):
    # Takes a page index and runs the relevant page reset function for that page
    if target_page == self.page_dictionary['login_page']:
        self.login_reset_page()
    elif target_page == self.page_dictionary['create_account_page']:
        self.create_account_reset_page()
    elif target_page == self.page_dictionary['question_page']:
        self.current_question = 0
        self.question_reset_page()
    elif target_page == self.page_dictionary['previous_scores_page']:
        self.previous_scores_reset_page()
    elif target_page == self.page_dictionary['admin_page']:
        self.admin_reset_page()
    elif target_page == self.page_dictionary['account_management_page']:
        self.account_management_reset_page()
    elif target_page == self.page_dictionary['view_classes_page']:
        self.view_classes_reset_page()
    elif target_page == self.page_dictionary['set_homework_page']:
        self.set_homework_reset_page()
    elif target_page == self.page_dictionary['homework_select_page']:
        self.homework_select_reset_page()
    else:
        pass
```

4.6.3.4.2 Testing

As this function is simply used to take a page index and select the script to setup the page for the given index, it can be tested to be working later as each full individual page reset script is written and will be verified to work on a case by case basis there. Testing of incorrect inputs cannot cause an error as the if statements will simply end up in a pass and the program will not respond. Additionally, the way the program is to be written, and incorrect target page index will not be inputted to the function as it will be verified before this stage.

4.6.3.5 Navigation bar

4.6.3.5.1 Function to hide/show main menu button

4.6.3.5.1.1 Final Code

```
def show_main_menu_button(self):
    # Enables and shows main menu button
    self.main_menu_button.setEnabled(True)
    self.main_menu_button.setVisible(True)

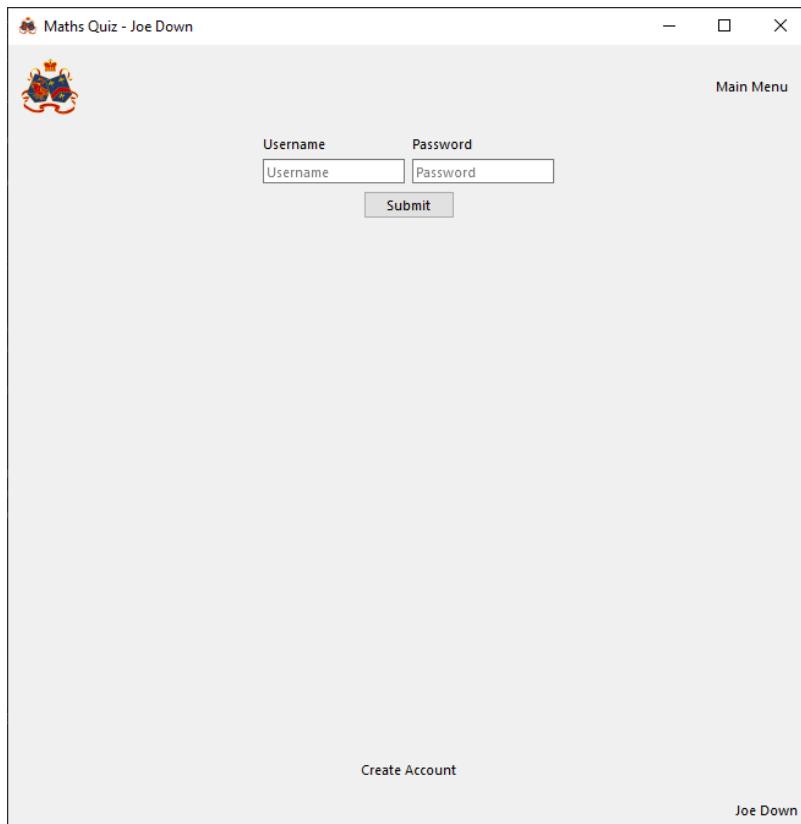
def hide_main_menu_button(self):
    # Disables and hides main menu button
    self.main_menu_button.setEnabled(False)
    self.main_menu_button.setVisible(False)
```

4.6.3.5.1.2 Testing

4.6.3.5.1.2.1 With valid data:

4.6.3.5.1.2.1.1 Show button

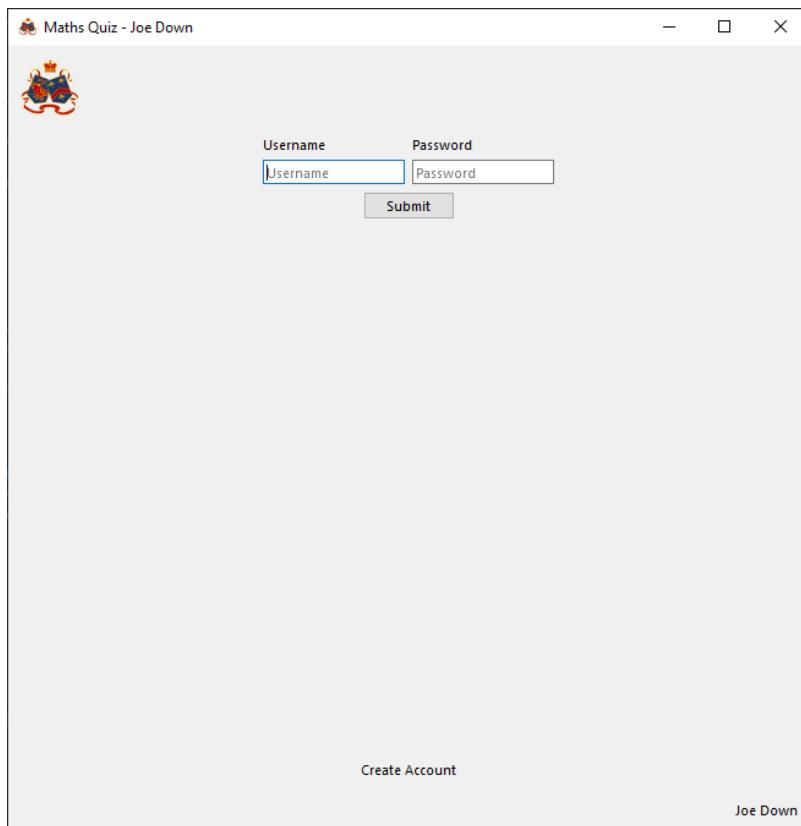
```
window.show_main_menu_button()
```



Button successfully shows

4.6.3.5.1.2.1.2 Hide button

```
window.hide_main_menu_button()
```



Button is successfully hidden

4.6.3.5.1.2.2 With Invalid data:

N/A, functions take no inputs and so button can only ever be enabled or disabled

4.6.3.5.1.3 Evaluation

Functions successfully hide or show button when called

4.6.3.5.2 Function to hide/show username

4.6.3.5.2.1 Final Code

```
def show_username_text(self):
    # Makes the user's first name show in ui
    self.username_label.setText(db_scripts.get_first_name(self.current_user).title())

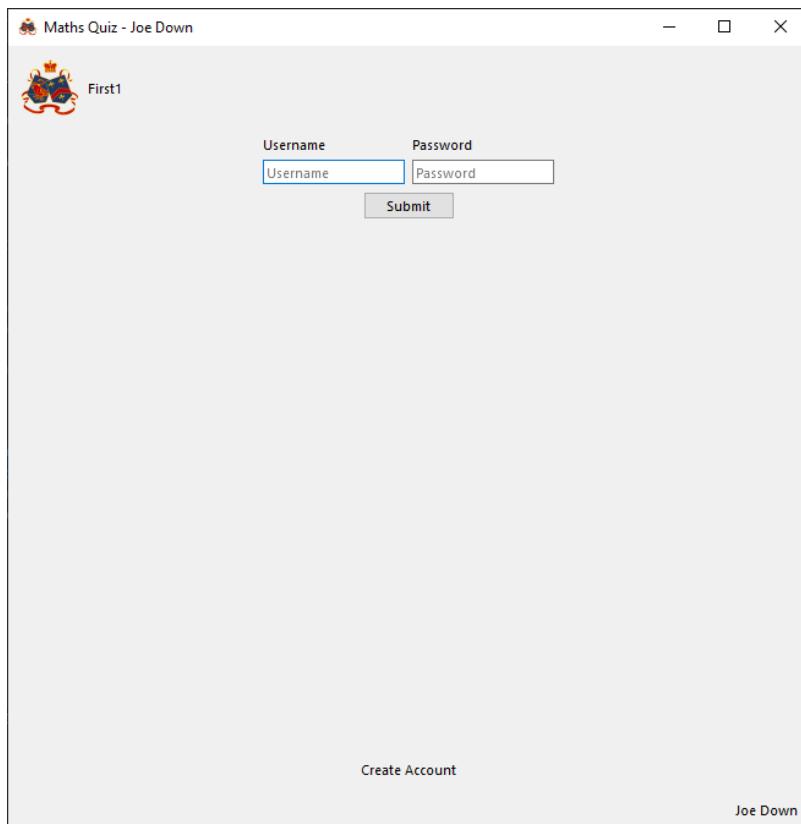
def hide_username_text(self):
    # Stops showing the user's first name in ui
    self.username_label.setText("")
```

4.6.3.5.2.2 Testing

4.6.3.5.2.2.1 With valid data:

4.6.3.5.2.2.1.1 Show

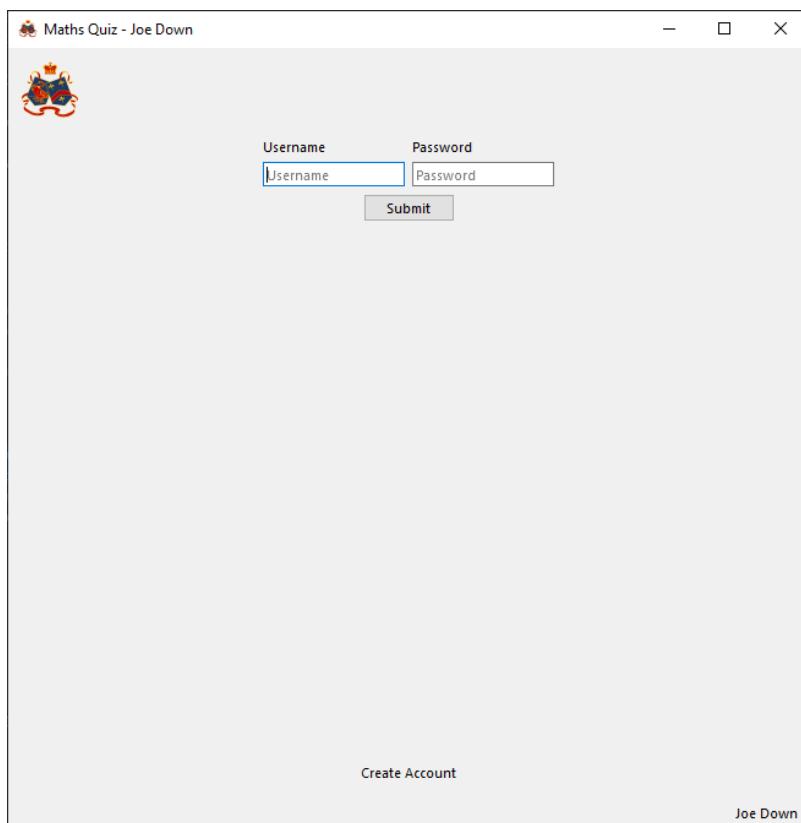
```
window.current_user = 1
window.show_username_text()
```



Label successfully shows

4.6.3.5.2.2.1.2 Hide

```
window.current_user = 1  
window.hide_username_text()
```



Label is successfully hidden

4.6.3.5.2.2 With Invalid data:

N/A, functions take no inputs and so output can only ever be enabled or disabled

4.6.3.5.2.3 Evaluation

Functions successfully hide or show username output when called

4.6.3.5.3 Function to hide/show logout button

4.6.3.5.3.1 Final Code

```
def show_logout_button(self):
    # Enables and shows logout button
    self.logout_button.setEnabled(True)
    self.logout_button.setVisible(True)

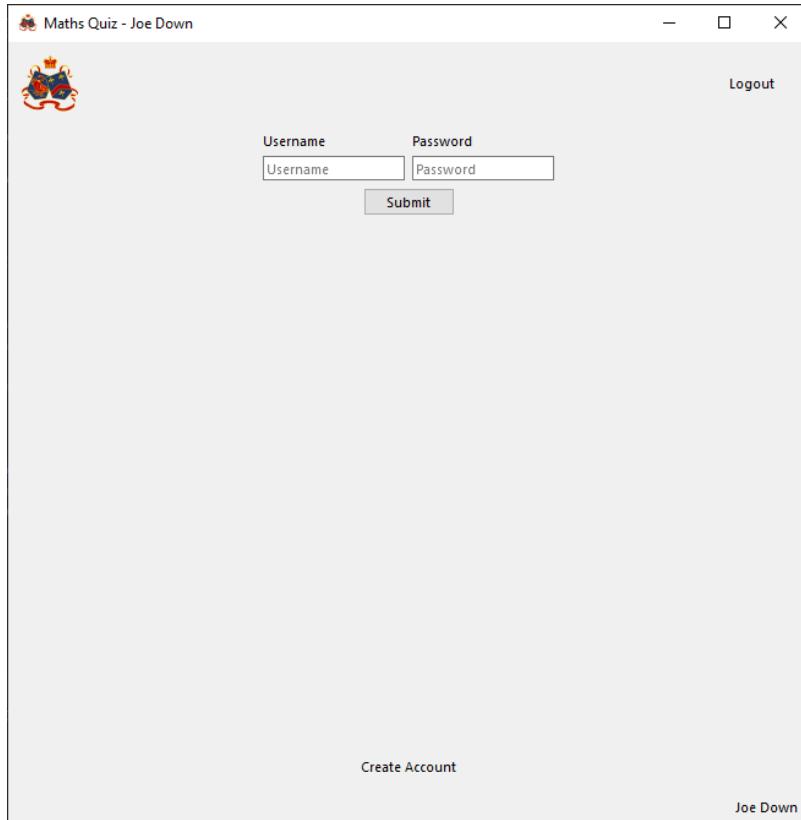
def hide_logout_button(self):
    # Disables and hides the logout button
    self.logout_button.setEnabled(False)
    self.logout_button.setVisible(False)
```

4.6.3.5.3.2 Testing

4.6.3.5.3.2.1 With valid data:

4.6.3.5.3.2.1.1 Show

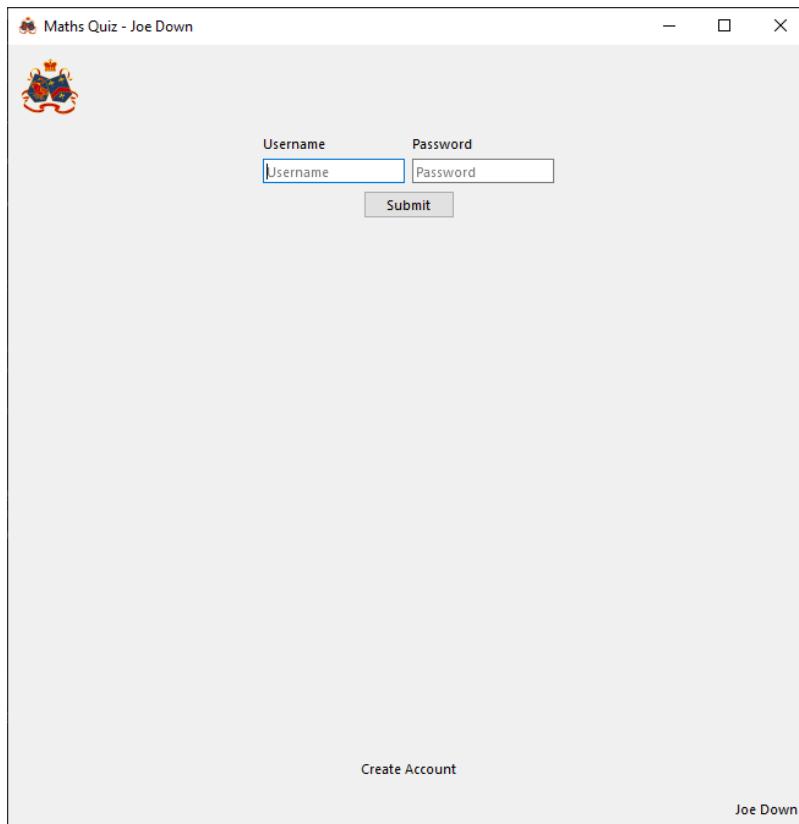
```
window.show_logout_button()
```



Button successfully shows

4.6.3.5.3.2.1.2 Hide

```
window.hide_logout_button()
```



Button is successfully hidden

4.6.3.5.3.2.2 With Invalid data:

N/A, functions take no inputs and so button can only ever be enabled or disabled

4.6.3.5.3.3 Evaluation

Functions successfully hide or show button when called

4.6.3.5.4 Function to set up buttons

4.6.3.5.4.1 Final Code

```
def navigation_button_setup(self):
    # If logout button clicked runs logout scripts
    self.logout_button.clicked.connect(self.logout)
    # If main menu button clicked runs scripts to load main menu
    self.main_menu_button.clicked.connect(self.go_to_main_menu)
```

4.6.3.5.4.2 Testing

4.6.3.5.4.2.1 With valid data:

Functionality of buttons will be verified to work during code for respective functions later in development and this section will be updated if changes are required

4.6.3.5.4.2.2 With Invalid data:

N/A, button is either pressed or not pressed, cannot give invalid inputs to function (as none are taken)

4.6.3.5.5 Function to go to main menu

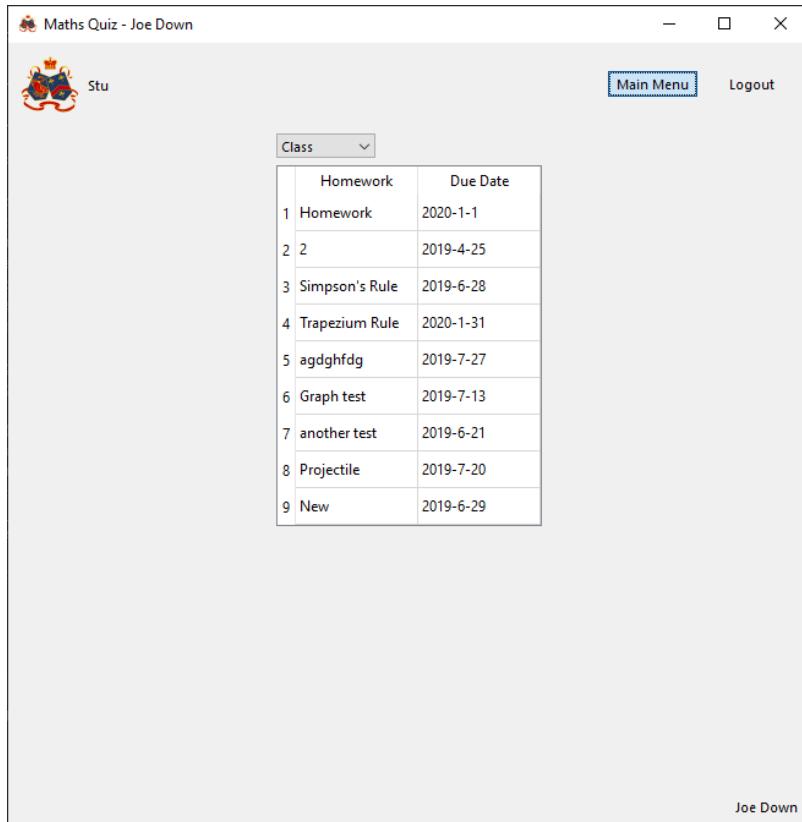
4.6.3.5.5.1 Final Code

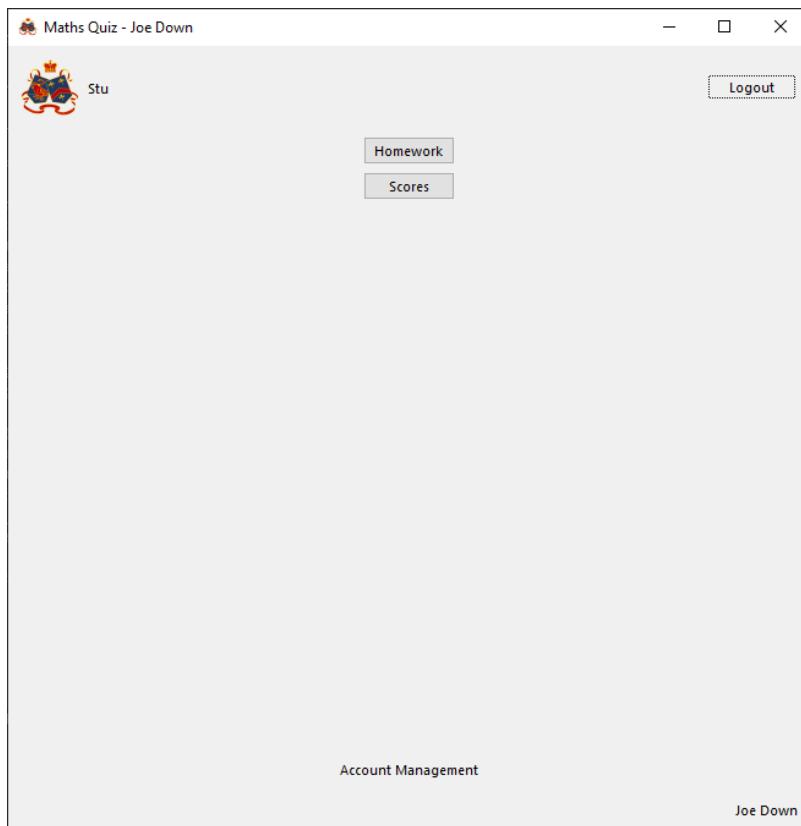
```
def go_to_main_menu(self):
    # Returns to the correct main menu page for the given user
    if db_scripts.get_account_type(self.current_user) == 't':
        self.change_page(self.page_dictionary['teacher_main_menu_page'])
    elif db_scripts.get_account_type(self.current_user) == 's':
        self.change_page(self.page_dictionary['student_main_menu_page'])
    # If data in DB is for some reason invalid, user is logged out and error shown
    else:
        self.logout()
        self.login_success_output.setText("User type error, user has been logged
out")
```

4.6.3.5.5.2 Testing

4.6.3.5.5.2.1 With valid data:

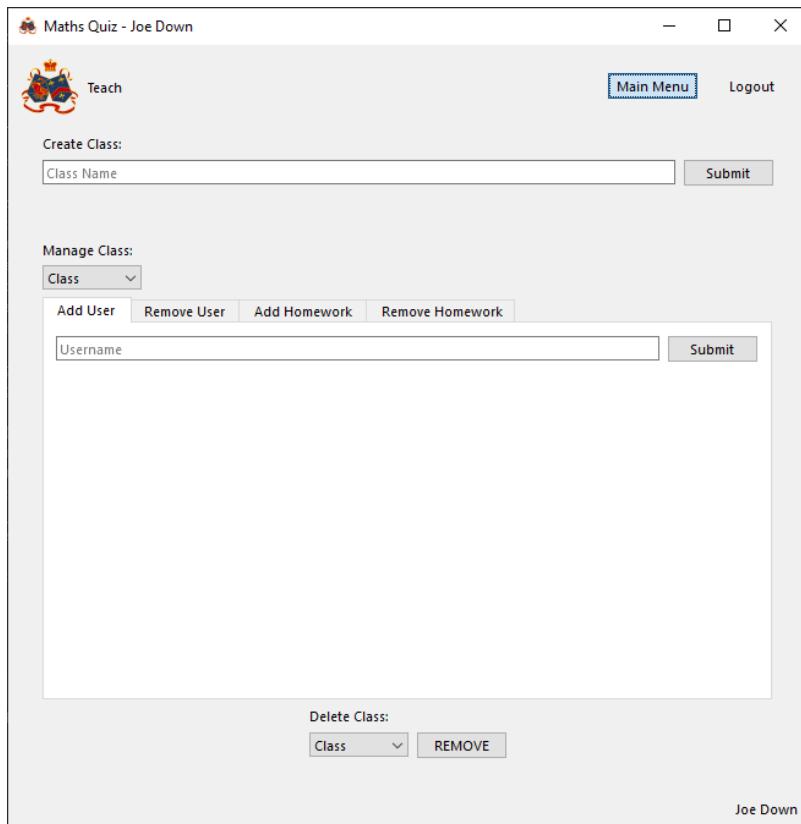
4.6.3.5.5.2.1.1 Student account

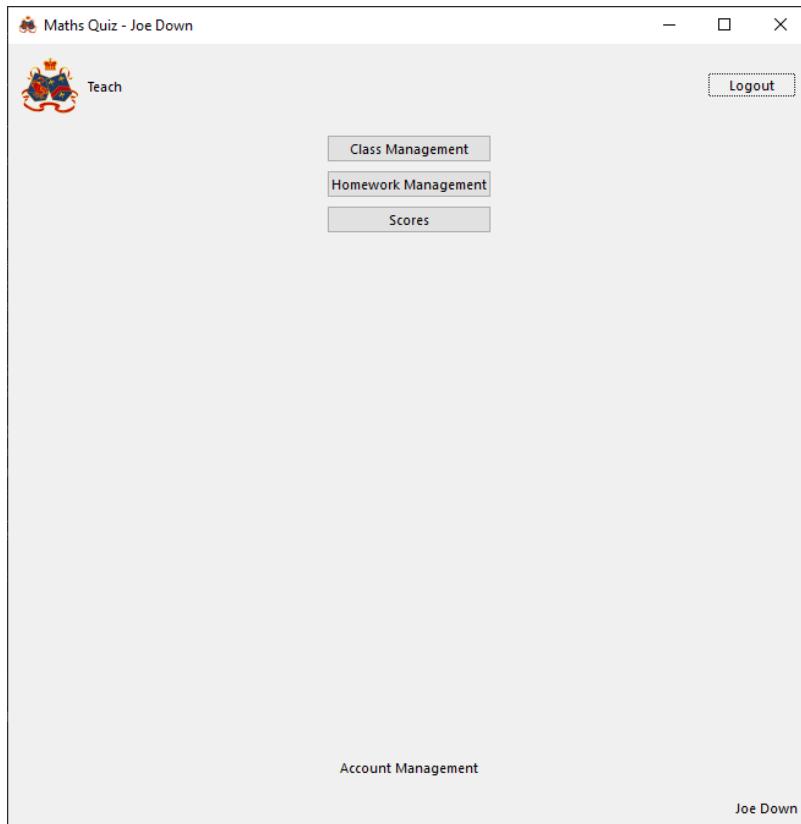




Pressing main menu button successfully navigates to student main menu

4.6.3.5.5.2.1.2 Teacher account





Pressing main menu button successfully navigates to teacher main menu

4.6.3.5.5.2.2 With Invalid data:

N/A, an invalid user type will be verified to be invalid before user can ever log in and at account creation and so no screen with the main menu button will ever be reached before it is validated to be compatible with this function

4.6.3.5.5.3 Evaluation

Main menu button successfully navigates to correct main menu page for given user

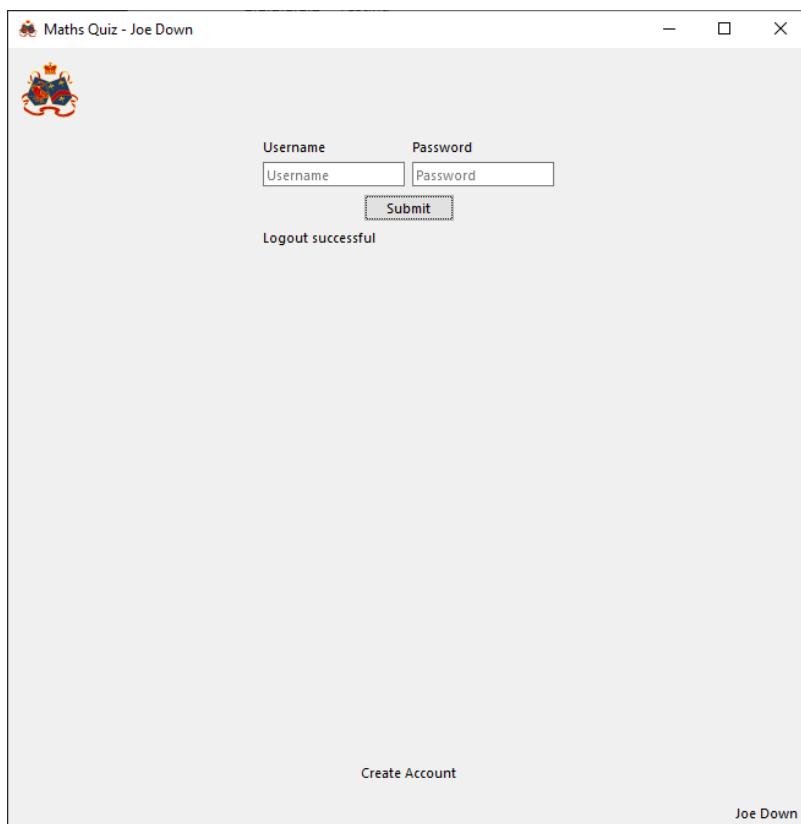
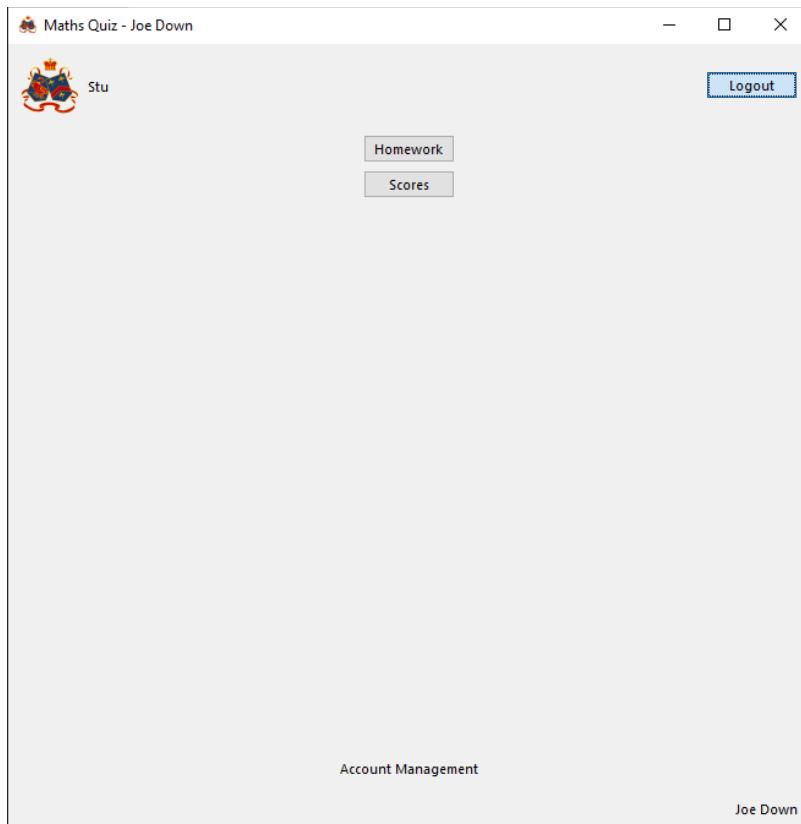
4.6.3.5.6 Function to go to log out

4.6.3.5.6.1 Final Code

```
def logout(self):
    # Resets current user to a default value
    self.current_user: int = -1
    # Returns to login page and sets logout success message
    self.change_page(self.page_dictionary['login_page'])
    self.login_success_output.setText("Logout successful")
```

4.6.3.5.6.2 Testing

4.6.3.5.6.2.1 With valid data:



Successfully returns to login page, informs user they have been logged out and sets current user id back to a placeholder value

4.6.3.5.6.2.2 With Invalid data:

N/A, no data inputted to function and so function will always run the same way when called

4.6.3.5.6.3 Evaluation

Successfully returns to main menu and logs out of account

4.6.3.6 Login page

4.6.3.6.1 Functions to reset page to default state

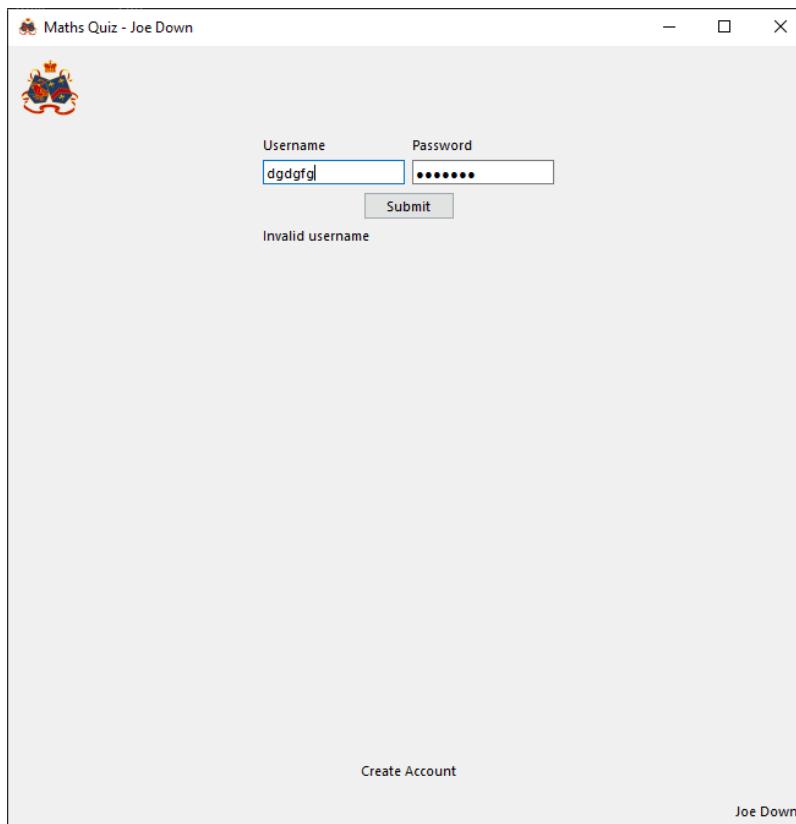
4.6.3.6.1.1 Final Code

```
def login_reset_page(self):
    # Sets input boxes to blank
    self.login_username_input.setText("")
    self.login_password_input.setText("")
    # Sets output labels to blank
    self.login_success_output.setText("")
```

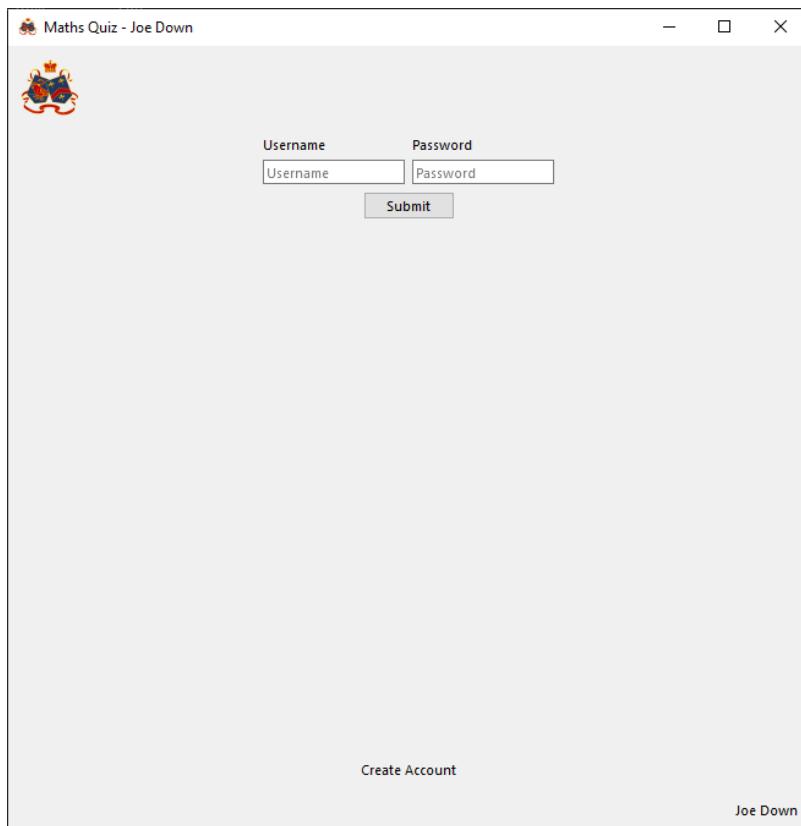
4.6.3.6.1.2 Testing

4.6.3.6.1.2.1 With valid data:

Input boxes have been filled and success output set to an error message:



On leaving page then returning page is successfully reset to default state (empty input boxes and output labels):



4.6.3.6.1.2.2 *With Invalid data:*

N/A, no data input or processing so function will always run the same way each time it is run

4.6.3.6.1.3 *Evaluation*

Correctly clears all labels and inputs as required when page is in its default state

4.6.3.6.2 Functions to verify username and password and login

4.6.3.6.2.1 *Issues during development*

Initially, no `casifold()` was applied to the inputs, meaning it was easy to mistakenly be denied access to an account with any variation of the username but all lowercase (as it is stored this way in the database). A `casifold` was therefore applied when taking input in order to ensure all variations in capitalisation are handled in the same way

4.6.3.6.2.2 Final Code

```
def login(self):
    # Checks if username exists (case fold used to make username case insensitive
    even for characters such as ß)
    if db_scripts.check_user_exists(self.login_username_input.text().lower()):
        # Checks if password is correct
        user_id: int = db_scripts.get_user_id(self.login_username_input.text().lower())
        if db_scripts.check_password(user_id, self.login_password_input.text()):
            # Checks user type to decide which main menu to load
            if db_scripts.get_account_type(user_id) in ['s', 't']:
                self.login_success_output.setText("Success")
                # Sets the current logged in user class attribute to the now logged
                in user
                self.current_user: int = user_id
                # Main menu loaded
                self.go_to_main_menu()
                # If the stored user type is for some reason invalid, login is can-
                celled and error shown
            else:
                self.login_success_output.setText("User type error")
                # If password incorrect, invalid password error is displayed and password
                box is cleared
            else:
                self.login_success_output.setText("Invalid password")
                self.login_password_input.setText("")
                # If username did not exist, invalid username error is displayed and username
                box is cleared
            else:
                self.login_success_output.setText("Invalid username")
                self.login_username_input.setText("")
```

4.6.3.6.2.3 Testing

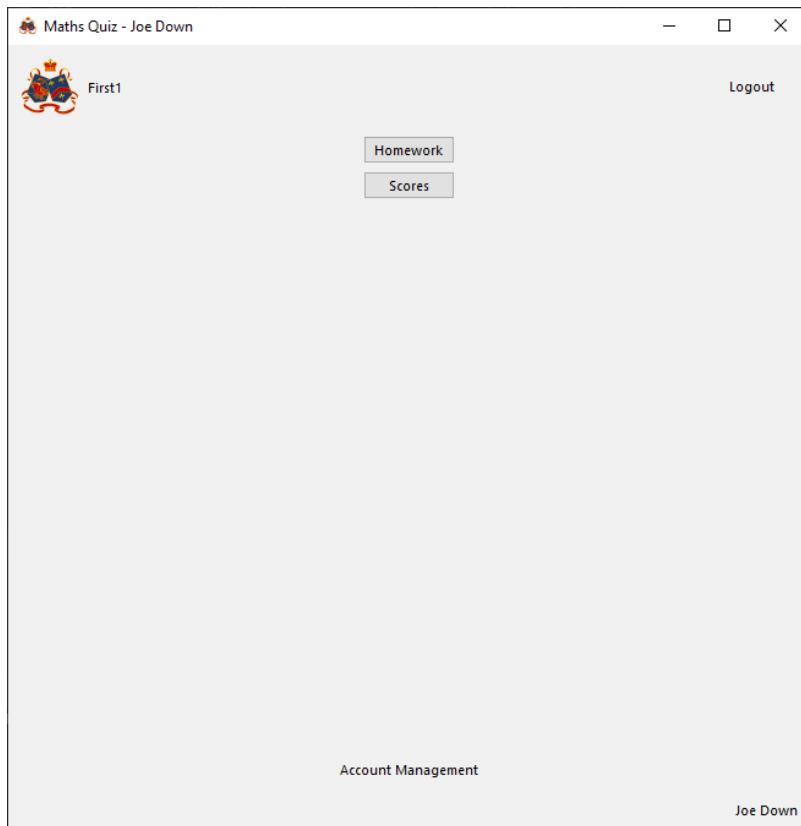
For testing here, valid username entered will be ‘student1’ and valid password will be ‘password’. Invalid inputs will be random long strings of characters of varying types. To login as a teacher the username ‘teacher1’ will instead be used

Notes:

- The nature of an input box means anything entered is always treated as a string, so data type validation is not needed before passing username or password to the database interaction functions used, it can simply be assumed to be of a usable type as they all take in strings
- Code to set up page buttons is given later

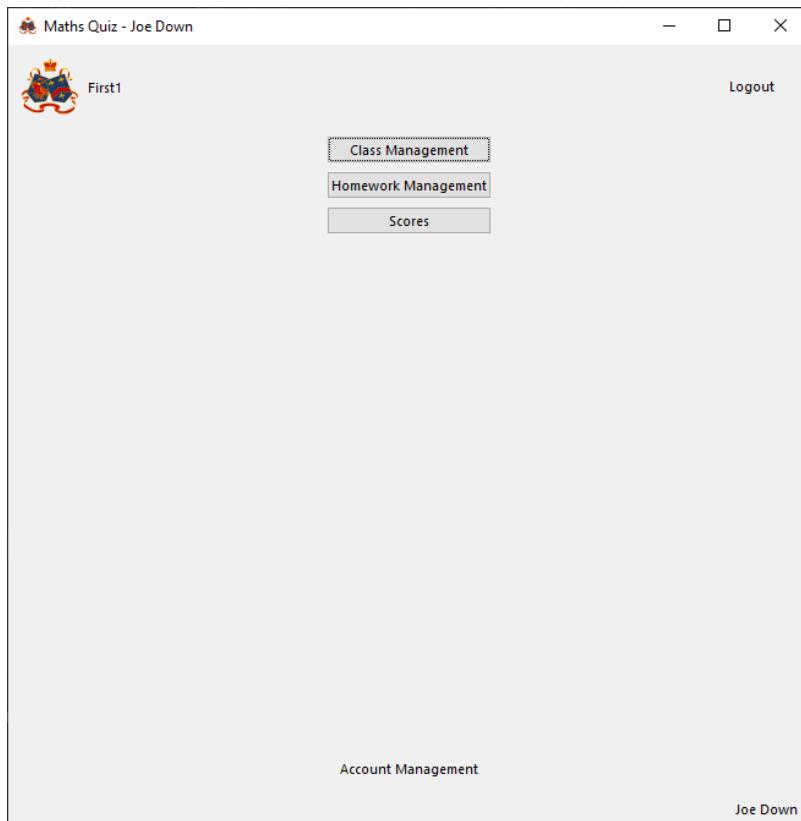
4.6.3.6.2.3.1 With valid data:

4.6.3.6.2.3.1.1 Student



Successfully loads student main menu page as correct user (as indicated by correct first name in top right)

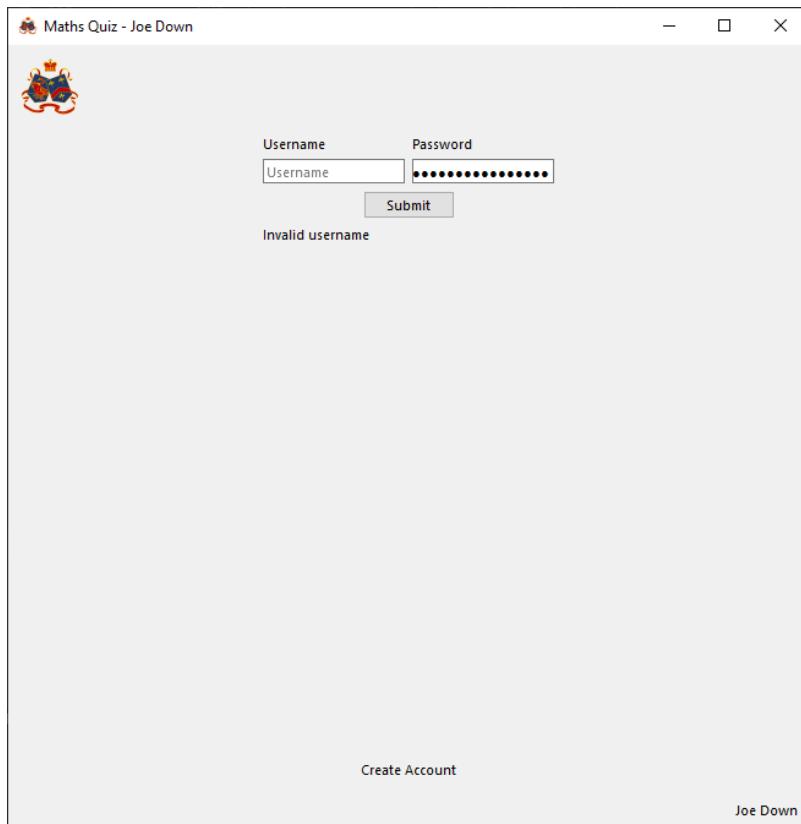
4.6.3.6.2.3.1.2 Teacher



Successfully loads teacher main menu page as correct user (as indicated by correct first name in top right)

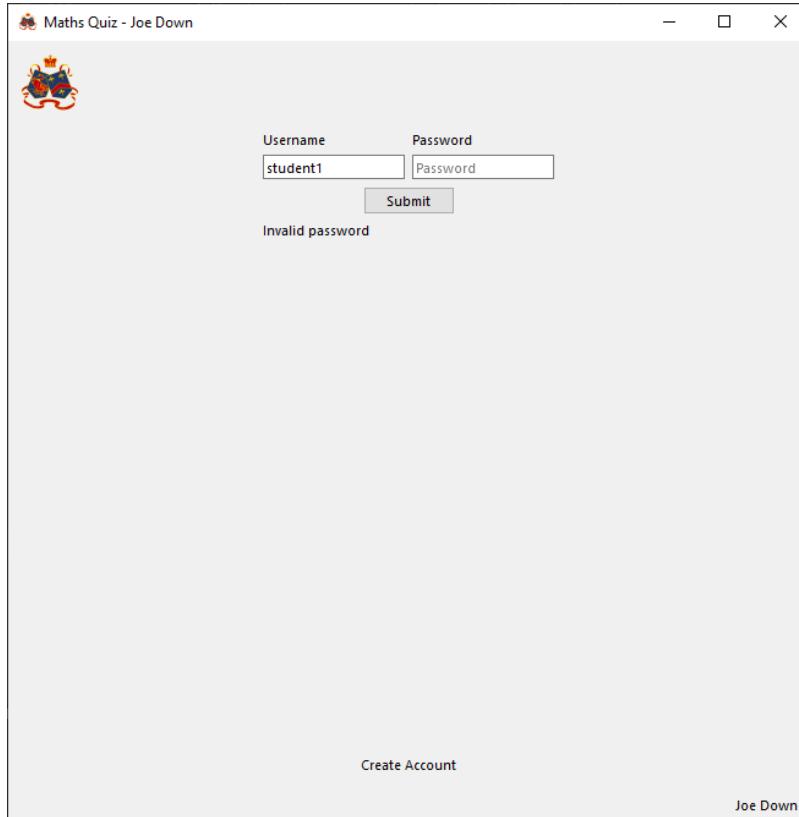
4.6.3.6.2.3.2 With Invalid data:

4.6.3.6.2.3.2.1 Invalid username



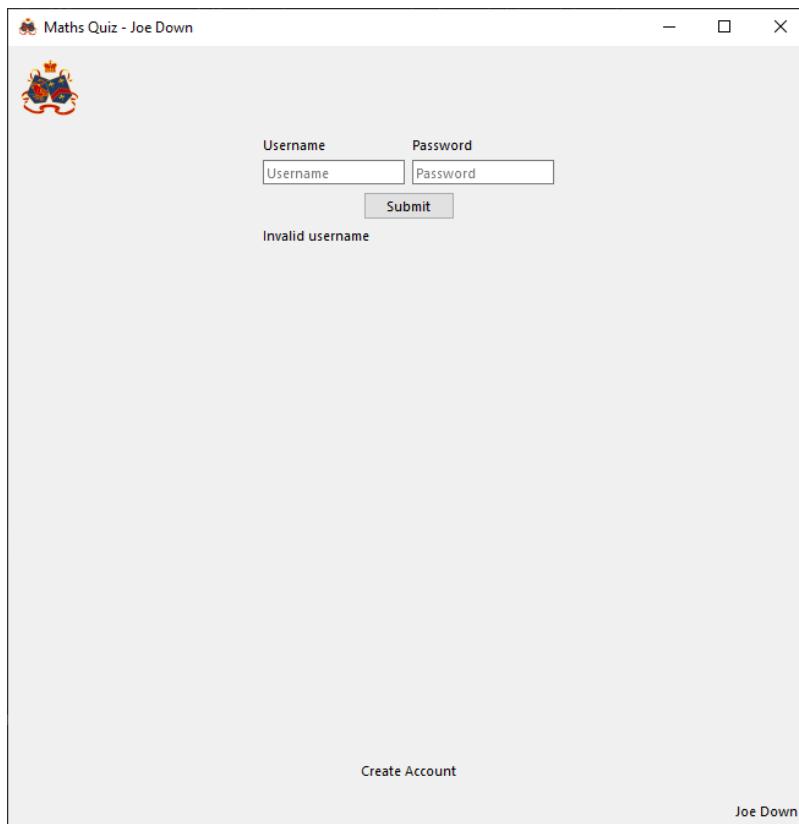
Successfully indicates invalid username and clears username input box

4.6.3.6.2.3.2.2 Invalid password



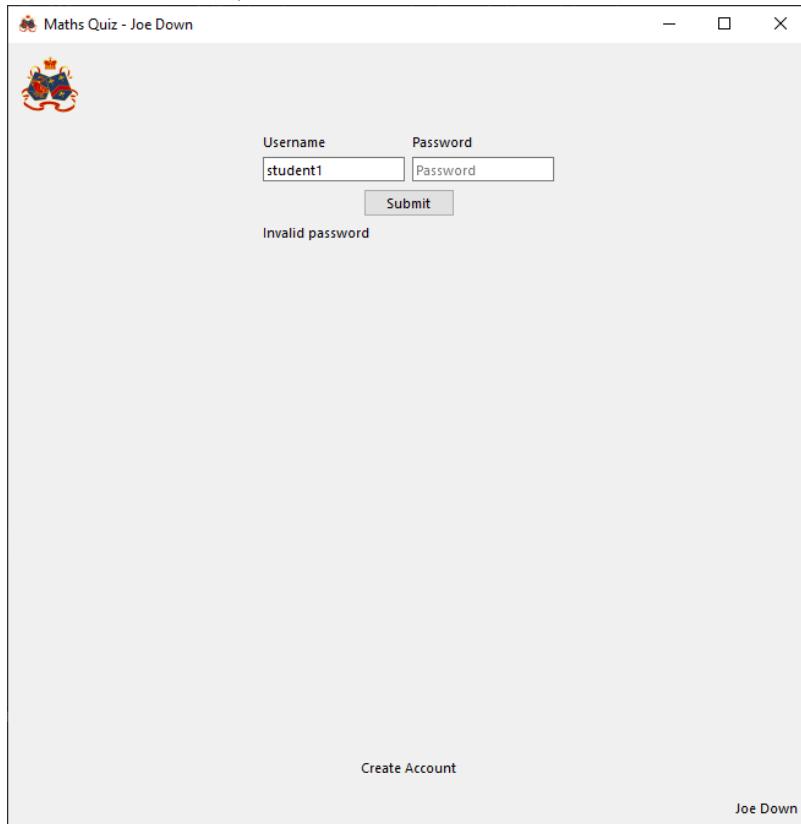
Successfully indicates invalid password and clears password input box

4.6.3.6.2.3.2.3 No username



Successfully indicates no valid username entered

4.6.3.6.2.3.2.4 No password



Successfully indicates invalid password

4.6.3.6.2.3.3 Evaluation

Correctly rejects login if not completely correct and outputs to user what is wrong with their login attempt. If login is valid, login takes user to the correct main menu for their user type and sets the logged in user id accordingly

4.6.3.6.3 Functions to set up page buttons

4.6.3.6.3.1 Issues during development

Initially the below code was implemented:

```
def login_page_button_setup(self):
    # If login submit button clicked runs scripts to verify login
    self.login_submit_button.clicked.connect(self.login)
    # If create account clicked runs scripts to change screen
    self.login_create_account_button.clicked.connect(self.change_page(self.page_dictionary['create_account_page']))
```

This would result in the following error:

```
C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe
C:/Users/Joe/PycharmProjects/Coursework/main.py
```

Traceback (most recent call last):

```
  File "C:/Users/Joe/PycharmProjects/Coursework/main.py", line 1096, in <module>
    window = Window()
  File "C:/Users/Joe/PycharmProjects/Coursework/main.py", line 106, in __init__
    self.button_setup()
  File "C:/Users/Joe/PycharmProjects/Coursework/main.py", line 154, in button_setup
    self.login_page_button_setup()
  File "C:/Users/Joe/PycharmProjects/Coursework/main.py", line 283, in login_page_button_setup
    self.login_create_account_button.clicked.connect(self.change_page(self.page_dictionary['create_account_page']))

  File "C:/Users/Joe/PycharmProjects/Coursework/main.py", line 139, in change_page
    if self.current_user != -1:
AttributeError: 'Window' object has no attribute 'current_user'
```

Process finished with exit code 1

As researched here (<https://stackoverflow.com/questions/45793966/clicked-connect-error>), this is due to the create account button's code in window.py being unable to call a function from externally with an argument and so the code was changed to utilise a lambda expression to essentially create a virtual function with no parameters that can be called by the button which internally is passed the given argument.

The create account button was changed to instead be defined with the following code:

```
# If create account clicked runs scripts to change screen
# Error passing arguments fixed using https://stackoverflow.com/questions/45793966/clicked-connect-error
self.login_create_account_button.clicked.connect(
    lambda: self.change_page(self.page_dictionary['create account page']))
```

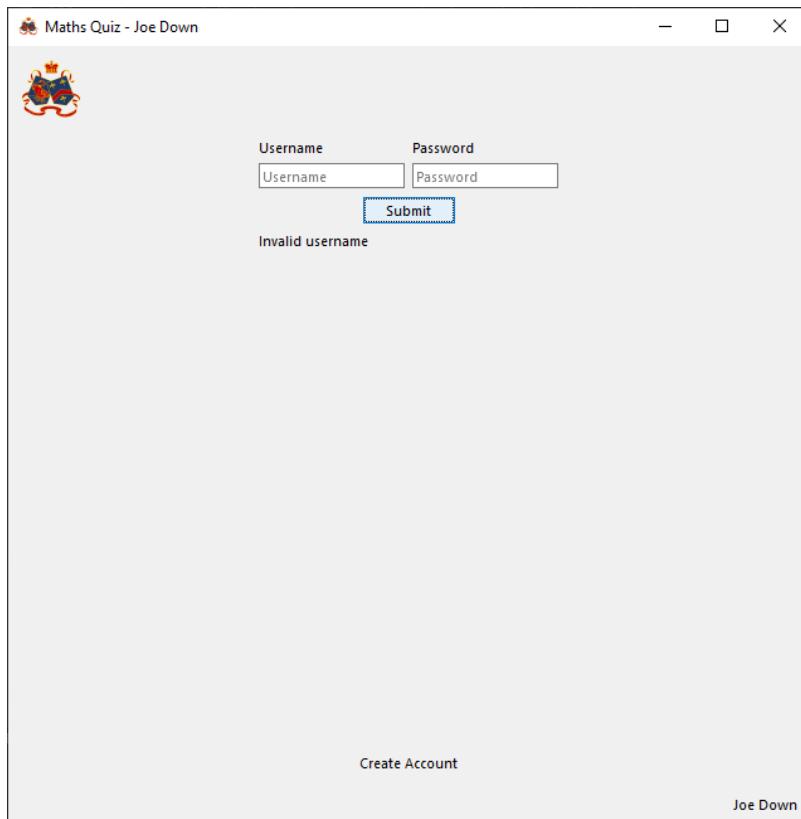
4.6.3.6.3.2 Final Code

```
def login_page_button_setup(self):
    # If login submit button clicked runs scripts to verify login
    self.login_submit_button.clicked.connect(self.login)
    # If create account clicked runs scripts to change screen
    # Error passing arguments fixed using https://stackoverflow.com/questions/45793966/clicked-connect-error
    self.login_create_account_button.clicked.connect(
        lambda: self.change_page(self.page_dictionary['create account page']))
```

4.6.3.6.3.3 Testing

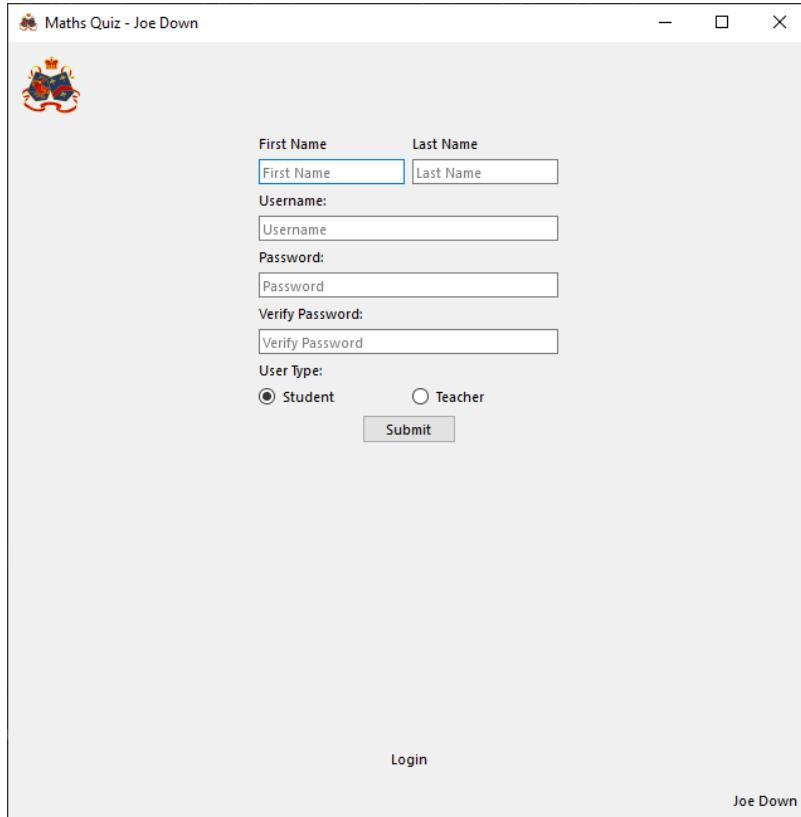
4.6.3.6.3.3.1 With valid data:

4.6.3.6.3.3.1.1 Submit button clicked



Successfully runs login function

4.6.3.6.3.3.1.2 Create account button clicked



Successfully loads create account page

4.6.3.6.3.3.2 With Invalid data:

N/A, the nature of a button means it is either pressed or not pressed so there is no way of making an incorrect data. Additionally, this function performs no data manipulation so will always execute the same way

4.6.3.6.3.4 Evaluation

Buttons successfully perform assigned tasks

4.6.3.7 Create account page

4.6.3.7.1 Functions to reset page to default state

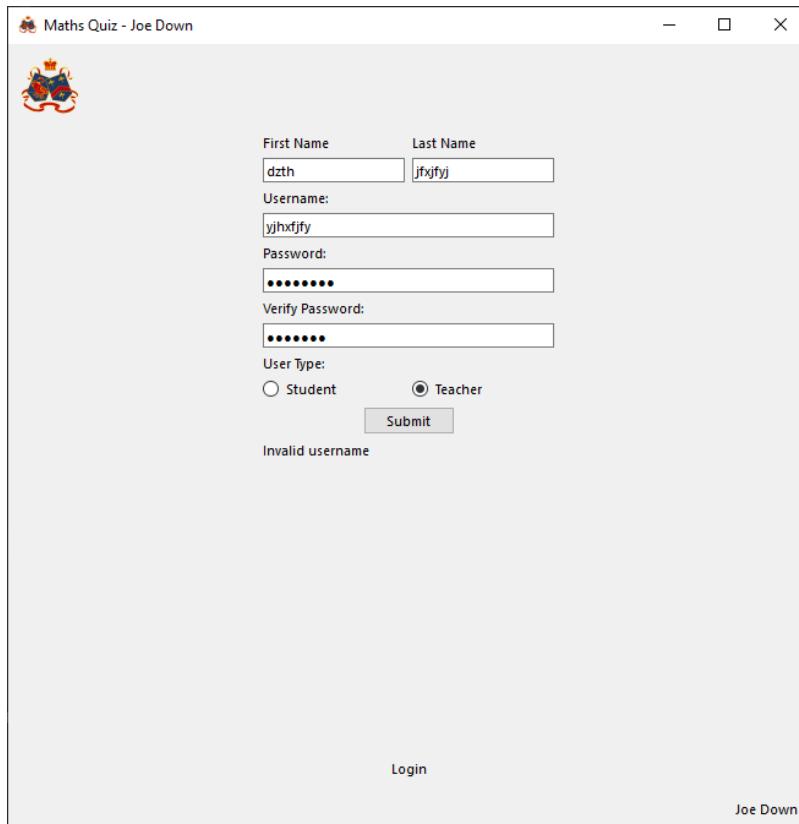
4.6.3.7.1.1 Final Code

```
def create_account_reset_page(self):
    # Sets radios to default selection
    self.create_account_radio_student.setChecked(True)
    # Sets input boxes to blank
    self.create_account_first_name_input.setText("")
    self.create_account_last_name_input.setText("")
    self.create_account_username_input.setText("")
    self.create_account_password_input.setText("")
    self.create_account_password_verify_input.setText("")
    # Sets output labels to blank
    self.create_account_success_output.setText("")
```

4.6.3.7.1.2 Testing

4.6.3.7.1.2.1 With valid data:

Input boxes have been filled, success output set to an error message and the non-default radio selection selected:



On leaving page then returning, page is successfully reset to default state (empty input boxes and output labels and radio button reset to default selection):

Maths Quiz - Joe Down



First Name Last Name

Username:

Password:

Verify Password:

User Type:
 Student Teacher

Login Joe Down

4.6.3.7.1.2.2 *With Invalid data:*

N/A, no data input or processing so function will always run the same way each time it is run

4.6.3.7.1.3 *Evaluation*

Correctly clears all labels, inputs and radio buttons as required when page is in its default state

4.6.3.7.2 Create an account

4.6.3.7.2.1 Final Code

```
def create_account_create_account(self):
    # Various error checks before creating account (error type is outputted in a
    label):
        # Error if username field is blank
        if self.create_account_username_input.text() == '':
            self.create_account_success_output.setText("Invalid username")
        # Error if username already taken
        elif db_scripts.check_user_exists(self.create_account_username_input.text()).case-
        selfold():
            self.create_account_success_output.setText("User already exists")
        # Error if no first name entered
        elif self.create_account_first_name_input.text() == '' or len(
            self.create_account_first_name_input.text()) > 100:
            self.create_account_success_output.setText("Invalid first name")
        # Error if no last name is entered
        elif self.create_account_last_name_input.text() == '' or len(self.create_ac-
        count_last_name_input.text()) > 100:
            self.create_account_success_output.setText("Invalid last name")
        # Error if password too short
        elif len(self.create_account_password_input.text()) < 8:
            self.create_account_success_output.setText("Invalid password (Must be at
            least 8 characters)")
        # Error if different password entered into second password box
        elif self.create_account_password_input.text() != self.create_account_pass-
        word_verify_input.text():
            self.create_account_success_output.setText("Passwords do not match")
        # Error if account type not selected
        elif not ((self.create_account_get_account_type_selected() == 's') or (
            self.create_account_get_account_type_selected() == 't')):
            self.create_account_success_output.setText("Account type not selected")
        # If passes all validation, account is created
        else:
            # Passes all relevant data into create account function
            db_scripts.create_account(self.create_account_username_input.text().case-
            fold(),
                self.create_account_password_input.text(),
                self.create_account_first_name_input.text(),
                self.create_account_last_name_input.text(),
                self.create_account_get_account_type_selected())
            # Resets create account page once account successfully created
            self.reset_page(self.page_dictionary['create_account_page'])
            # Account creation success outputted
            self.create_account_success_output.setText("Account created")

def create_account_get_account_type_selected(self) -> str:
    # Returns whether student or teacher is selected (or neither, though this
    should never occur)
    if self.create_account_radio_student.isChecked():
        return 's'
    elif self.create_account_radio_teacher.isChecked():
        return 't'
    else:
        return ''
```

4.6.3.7.2.2 Testing

4.6.3.7.2.2.1 With valid data:

4.6.3.7.2.2.1.1 Student

Username ‘student12345’ and password ‘password’ used

Maths Quiz - Joe Down



First Name Last Name
Stu Dent

Username:
student12345

Password:

Verify Password:

User Type:
 Student Teacher

Login Joe Down

Maths Quiz - Joe Down



First Name Last Name
First Name Last Name

Username:
Username

Password:
Password

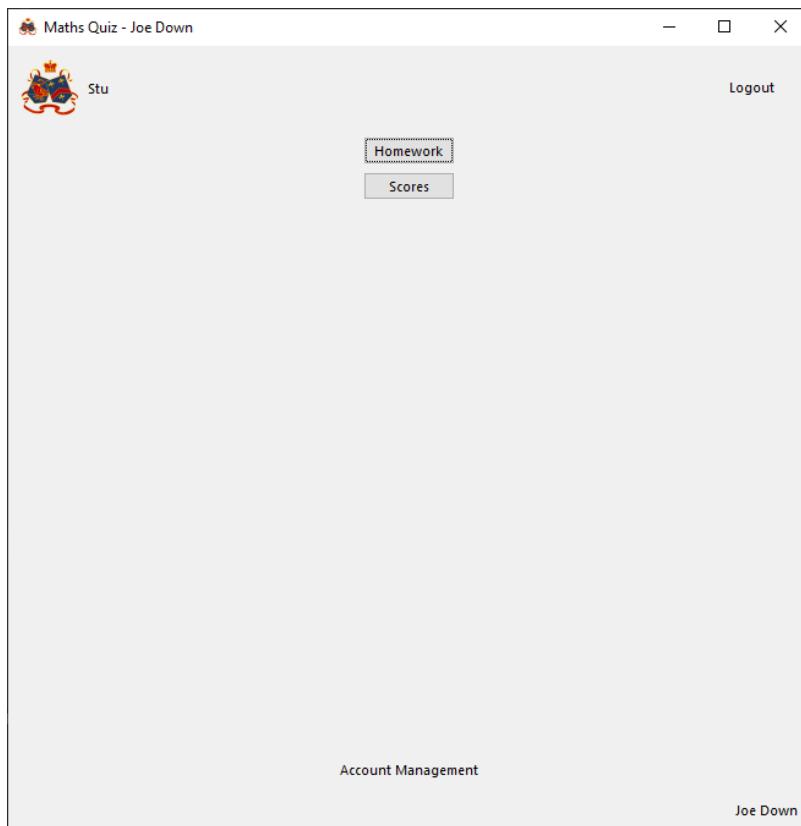
Verify Password:
Verify Password

User Type:
 Student Teacher

Account created

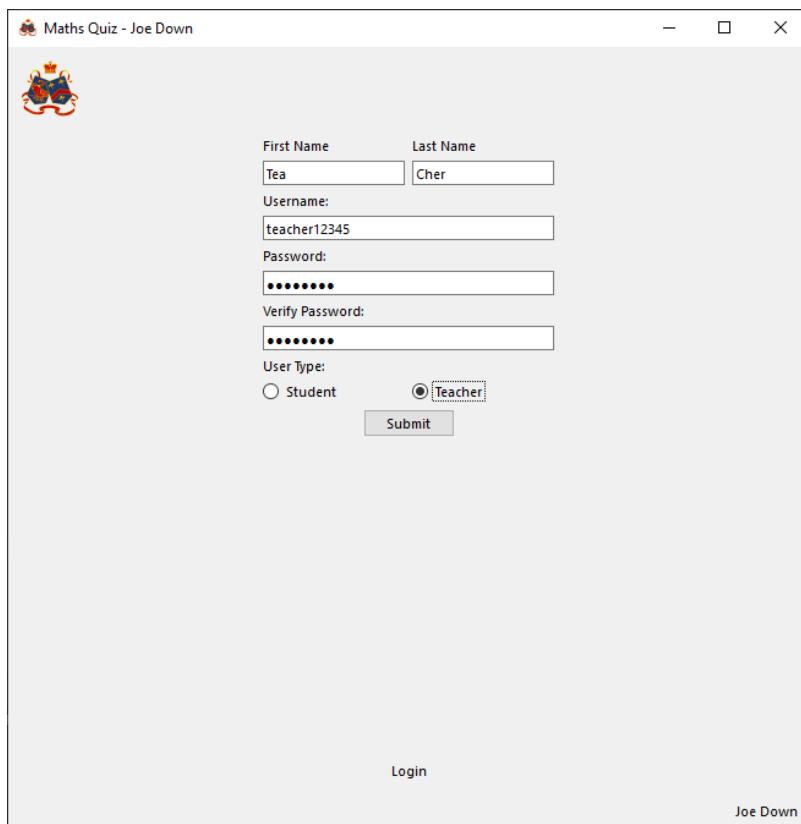
Login Joe Down

Account created output line runs so data should have been inserted into database. Logging in with the account details used now successfully goes to the student main menu page and the correct first name is shown:



4.6.3.7.2.2.1.2 Teacher

Username 'teacher12345' and password 'password' used



Maths Quiz - Joe Down



First Name Last Name

Username:

Password:

Verify Password:

User Type:
 Student Teacher

Account created

Login

Joe Down

Account created output line runs so data should have been inserted into database. Logging in with the account details used now successfully goes to the teacher main menu page and the correct first name is shown:

Maths Quiz - Joe Down



Tea Logout

Account Management

Joe Down

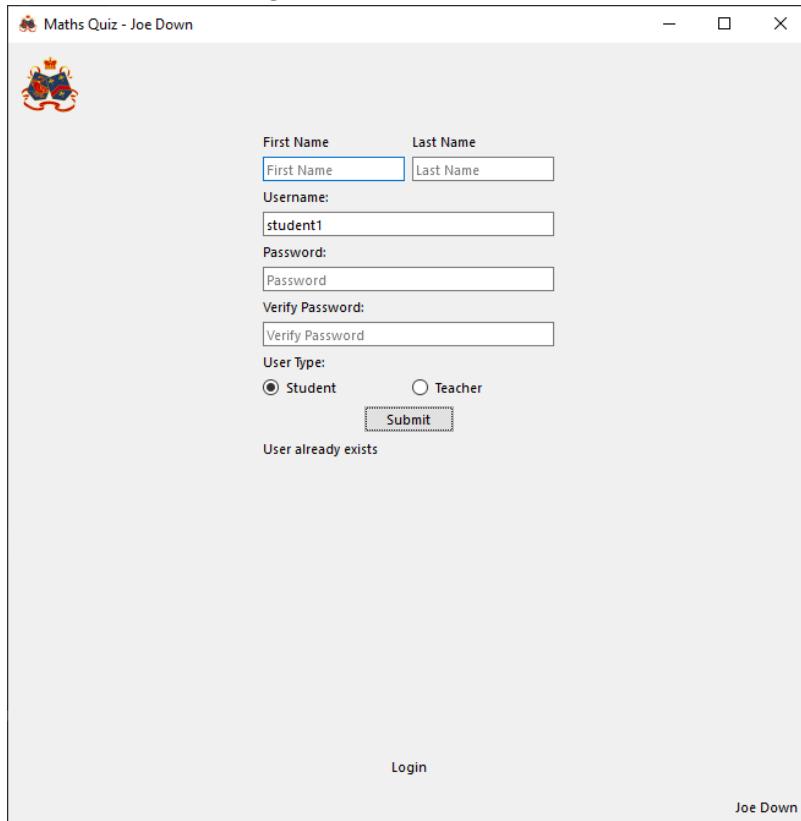
4.6.3.7.2.2.2 With Invalid data:

4.6.3.7.2.2.2.1 Invalid username (Blank)

The screenshot shows a Windows application window titled "Maths Quiz - Joe Down". Inside the window, there is a login form. The form includes fields for "First Name" and "Last Name" (both empty), "Username" (empty), "Password" (empty), "Verify Password" (empty), and "User Type" (radio buttons for "Student" and "Teacher", where "Student" is selected). Below the form is an error message: "Invalid username". At the bottom of the window, there is a "Login" link and the name "Joe Down".

Correctly outputs that an invalid username is entered

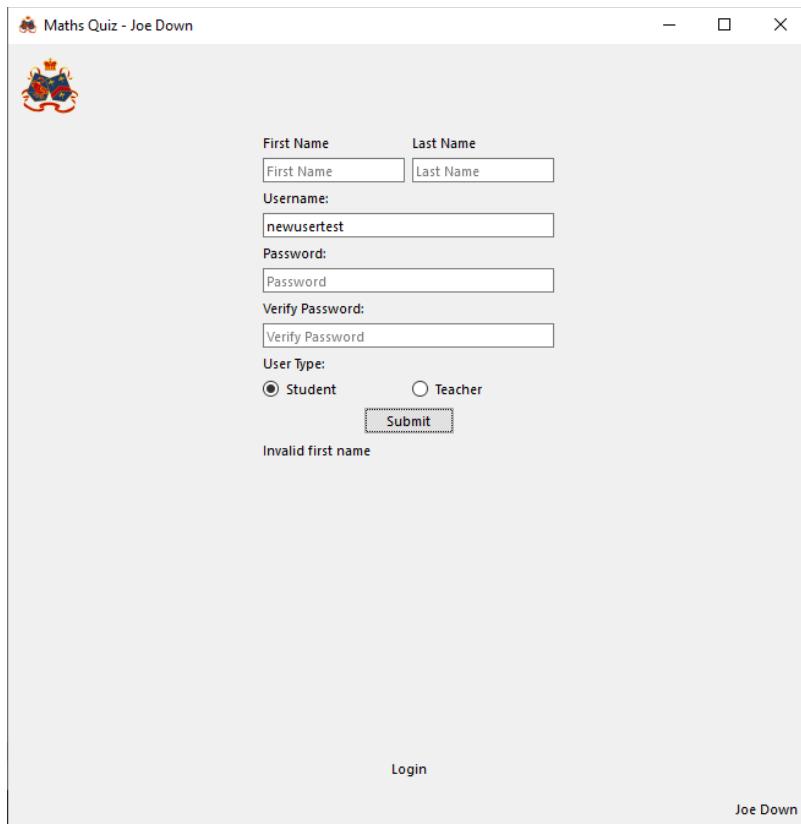
4.6.3.7.2.2.2.2 Existing username used



Correctly outputs that user already exists

4.6.3.7.2.2.2.3.3 Invalid first name

4.6.3.7.2.2.2.3.1 Blank



The screenshot shows a Windows application window titled "Maths Quiz - Joe Down". The window contains a registration form with the following fields:

- First Name: (empty)
- Last Name: (empty)
- Username: newusertest
- Password: (empty)
- Verify Password: (empty)
- User Type:
 Student Teacher
- Submit button

An error message "Invalid first name" is displayed below the form. At the bottom of the window, there are "Login" and "Joe Down" links.

Successfully outputs that first name is invalid

4.6.3.7.2.2.2.3.2 Name longer than 100 characters

Maths Quiz - Joe Down

First Name: fftxhbhxhhxhtxhzhz

Last Name:

Username: newusertest

Password:

Verify Password:

User Type:

Student Teacher

Invalid first name

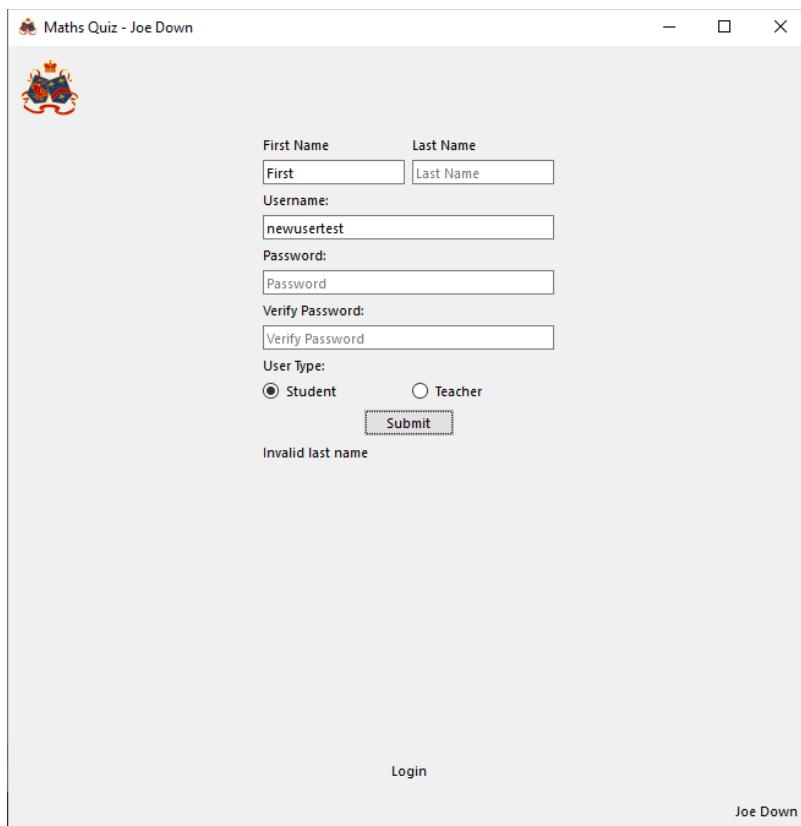
Login

Joe Down

Successfully outputs that first name is invalid

4.6.3.7.2.2.2.4.4 Invalid last name

4.6.3.7.2.2.2.4.1 Blank



The screenshot shows a Windows application window titled "Maths Quiz - Joe Down". The window contains a user registration form with the following fields:

- First Name: First
- Last Name: (empty)
- Username: newusertest
- Password: Password
- Verify Password: Verify Password
- User Type:
 Student Teacher
- Submit button

An error message "Invalid last name" is displayed below the "Last Name" field. At the bottom of the window, there are "Login" and "Joe Down" buttons.

Successfully outputs that last name is invalid

4.6.3.7.2.2.2.4.2 Name longer than 100 characters

Maths Quiz - Joe Down

First Name Last Name
First [txhthxhzzzzzzzzzzzz]
Username: newusertest
Password:
Verify Password:
User Type:
 Student Teacher
Submit

Invalid last name

Login Joe Down

Successfully outputs that last name is invalid

4.6.3.7.2.2.2.5 Invalid password (less than 8 characters)

Maths Quiz - Joe Down

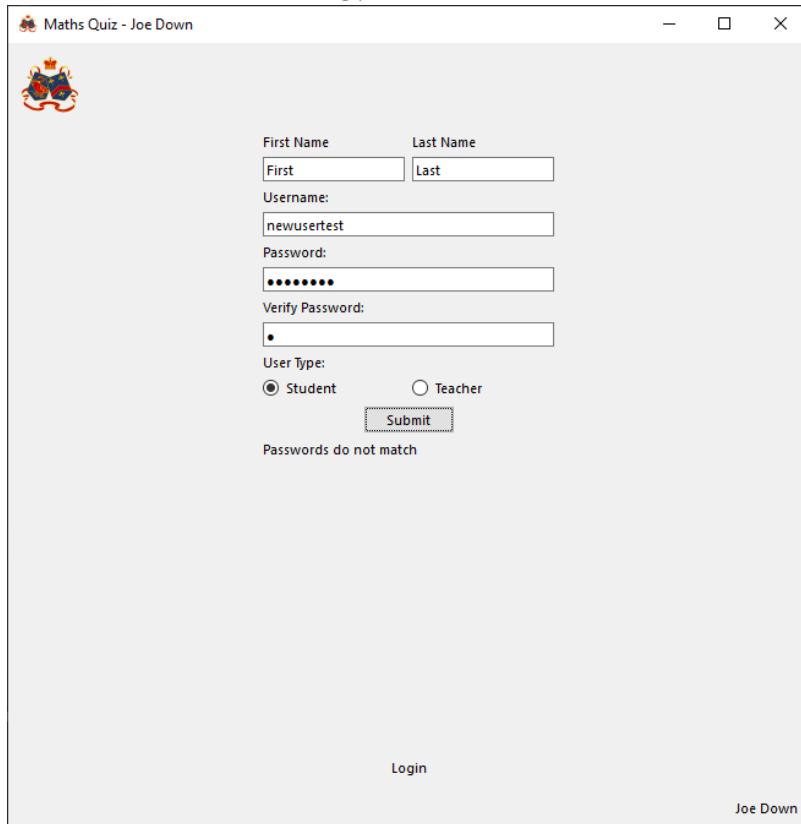
First Name Last Name
First [Last]
Username: newusertest
Password: [*****]
Verify Password:
User Type:
 Student Teacher
Submit

Invalid password (Must be at least 8 characters)

Login Joe Down

Successfully outputs that password is invalid

4.6.3.7.2.2.2.6 Non-matching password verification



Successfully outputs that passwords do not match

4.6.3.7.2.3 Evaluation

Successfully creates a new account if valid details entered

4.6.3.7.3 Function to set up page buttons

4.6.3.7.3.1 Final Code

```
def create_account_page_button_setup(self):
    # If create account submit button clicked runs scripts to create account
    self.create_account_submit_button.clicked.connect(self.create_account_create_account)
    # If return to login clicked runs scripts to change screen
    self.create_account_login_button.clicked.connect(lambda:
self.change_page(self.page_dictionary['login page']))
```

4.6.3.7.3.2 Testing

4.6.3.7.3.2.1 With valid data:

4.6.3.7.3.2.1.1 Submit button clicked

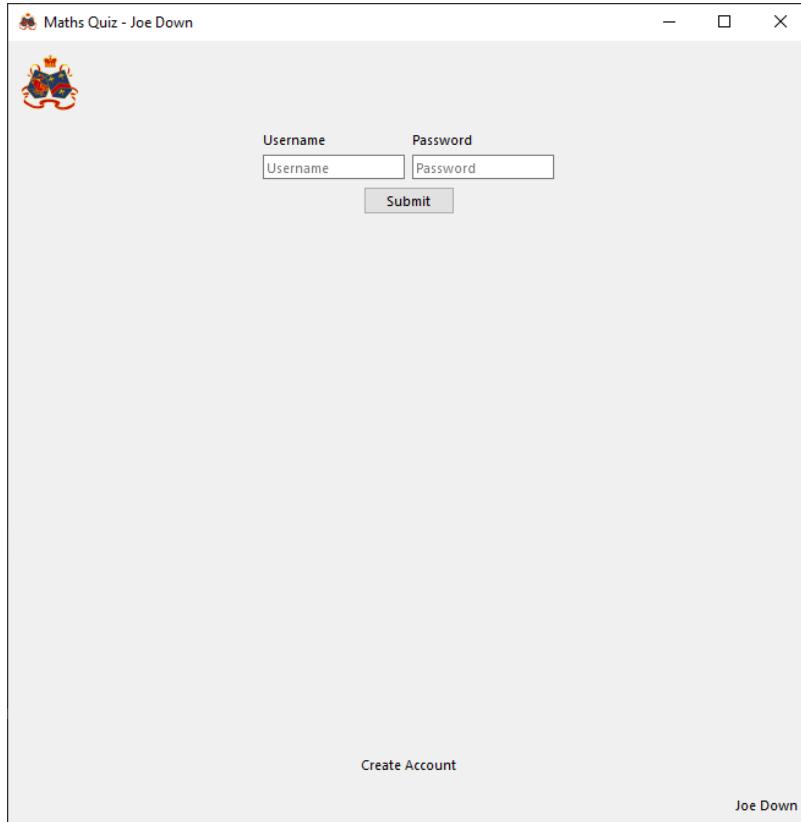
The screenshot shows a Windows application window titled "Maths Quiz - Joe Down". The window contains a user creation form with the following fields:

- First Name: [Text Box]
- Last Name: [Text Box]
- Username: [Text Box]
- Password: [Text Box]
- Verify Password: [Text Box]
- User Type:
 - Student
 - Teacher
- Submit [Button]

An error message "Invalid username" is displayed below the form. At the bottom of the window, there are "Login" and "Joe Down" links.

Successfully runs create account function

4.6.3.7.3.2.1.2 Login page button clicked



Successfully loads login page

4.6.3.7.3.2.2 With Invalid data:

N/A, the nature of a button means it is either pressed or not pressed so there is no way of making an incorrect data. Additionally, this function performs no data manipulation so will always execute the same way

4.6.3.7.3.3 Evaluation

Buttons successfully perform assigned tasks

4.6.3.8 Student main menu page

4.6.3.8.1 Set up buttons

4.6.3.8.1.1 Final Code

```
def student_main_menu_page_button_setup(self):
    # If homework clicked runs scripts to change screen
    self.student_main_menu_homework_button.clicked.connect(
        lambda: self.change_page(self.page_dictionary['homework_select_page']))
    # If previous scores clicked runs scripts to change screen
    self.student_main_menu_previous_scores_button.clicked.connect(
        lambda: self.change_page(self.page_dictionary['previous_scores_page']))
    # If account management clicked runs scripts to change screen
    self.student_main_menu_account_management_button.clicked.connect(
        lambda: self.change_page(self.page_dictionary['account management page']))
```

4.6.3.8.1.2 Evaluation

Clicking all buttons leads to the relevant page

4.6.3.9 Teacher main menu page

4.6.3.9.1 Set up buttons

4.6.3.9.1.1 Final Code

```
def teacher_main_menu_page_button_setup(self):
    # If 'set homework' clicked runs scripts to change screen
    self.teacher_main_menu_set_homework_button.clicked.connect(
        lambda: self.change_page(self.page_dictionary['set_homework_page']))
    # If 'view classes' clicked runs scripts to change screen
    self.teacher_main_menu_view_classes_button.clicked.connect(
        lambda: self.change_page(self.page_dictionary['view_classes_page']))
    # If 'account management' clicked runs scripts to change screen
    self.teacher_main_menu_account_management_button.clicked.connect(
        lambda: self.change_page(self.page_dictionary['account_management_page']))
    # If 'admin' clicked runs scripts to change screen
    self.teacher_main_menu_admin_button.clicked.connect(
        lambda: self.change_page(self.page_dictionary['admin_page']))
```

4.6.3.9.1.2 Evaluation

Clicking all buttons leads to the relevant page

4.6.3.10 Question page

4.6.3.10.1 Function to reset page

4.6.3.10.1.1 Final Code

```

def question_reset_page(self):
    # Gets the id of the question being loaded
    question_id = self.questions[self.current_question][0]
    # Sets selection of answer to initial position as a default
    self.question_radio_a.setChecked(True)
    # Sets all inputs to blank and outputs to what is required for the current
question
    self.question_topic_output.setText("")
    self.question_topic_output.setText(ui_scripts.get_question_type(question_id))
    self.question_question_output.setText(ui_scripts.get_question_text_of_ques-
tion(question_id))
    self.question_feedback_output.setText("")
    # Ensures all inputs are enabled
    self.question_submit_button.setEnabled(True)
    self.question_radio_a.setEnabled(True)
    self.question_radio_b.setEnabled(True)
    self.question_radio_c.setEnabled(True)
    self.question_radio_d.setEnabled(True)
    # Gets the correct answer and stores it
    correct_answer: str = ui_scripts.get_correct_answer_of_question(question_id)
    # Gets the incorrect answers and shuffles the order so if question is reloaded,
answer order is fully randomised
    incorrect_answers: list = ui_scripts.get_incorrect_answers_of_question(ques-
tion_id)
    random.shuffle(list(incorrect_answers))
    # Randomly selects which radio button will be labelled with the correct answer
and stores it's index. Other
    # radios are labelled with the incorrect answers
    self.correct_answer_location: int = random.randint(1, 4)
    if self.correct_answer_location == 1:
        self.question_radio_a.setText(correct_answer)
        self.question_radio_b.setText(incorrect_answers[0])
        self.question_radio_c.setText(incorrect_answers[1])
        self.question_radio_d.setText(incorrect_answers[2])
    elif self.correct_answer_location == 2:
        self.question_radio_b.setText(correct_answer)
        self.question_radio_a.setText(incorrect_answers[0])
        self.question_radio_c.setText(incorrect_answers[1])
        self.question_radio_d.setText(incorrect_answers[2])
    elif self.correct_answer_location == 3:
        self.question_radio_c.setText(correct_answer)
        self.question_radio_b.setText(incorrect_answers[0])
        self.question_radio_a.setText(incorrect_answers[1])
        self.question_radio_d.setText(incorrect_answers[2])
    elif self.correct_answer_location == 4:
        self.question_radio_d.setText(correct_answer)
        self.question_radio_b.setText(incorrect_answers[0])
        self.question_radio_c.setText(incorrect_answers[1])
        self.question_radio_a.setText(incorrect_answers[2])
    # Checks if user has already answered question correctly. If so, disables all
inputs so user cannot resubmit an
    # answer, sets the question feedback to "Correct" and selects the correct an-
swer's radio
    if ui_scripts.get_correct_status_of_question(self.current_user, question_id):
        if self.correct_answer_location == 1:
            self.question_radio_a.setChecked(True)
        elif self.correct_answer_location == 2:
            self.question_radio_b.setChecked(True)
        elif self.correct_answer_location == 3:
            self.question_radio_c.setChecked(True)
        elif self.correct_answer_location == 4:
            self.question_radio_d.setChecked(True)
        self.question_submit_button.setEnabled(False)
        self.question_radio_a.setEnabled(False)
        self.question_radio_b.setEnabled(False)
        self.question_radio_c.setEnabled(False)
        self.question_radio_d.setEnabled(False)
        self.question_feedback_output.setText("Correct")

```

```

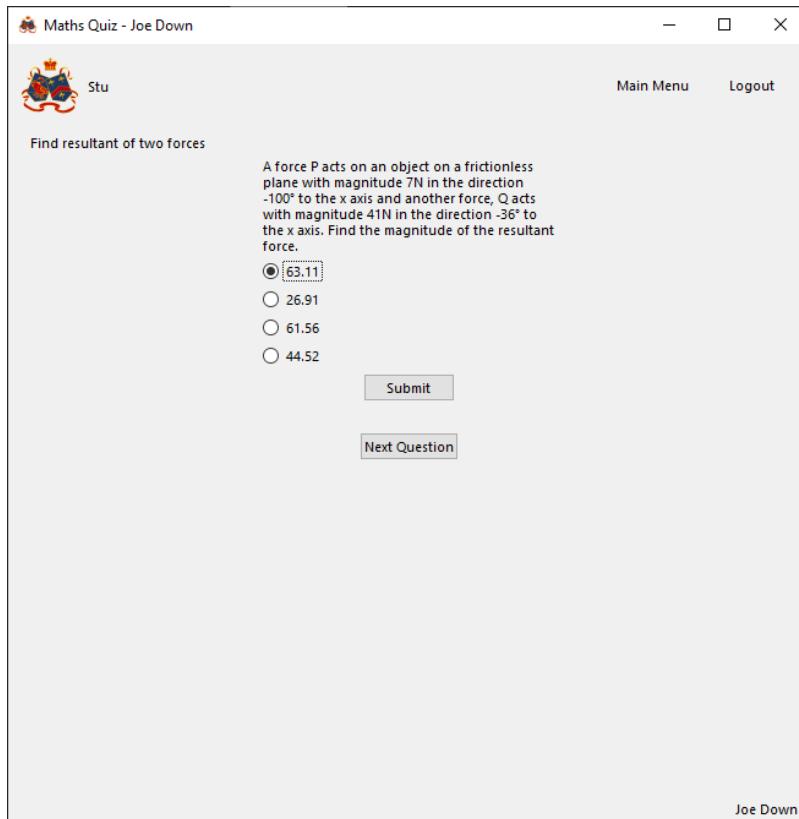
# If current question is first in homework, disables previous page button (as
there is none). Otherwise,
# enables it
if self.current_question == 0:
    self.question_previous_question_button.setEnabled(False)
    self.question_previous_question_button.setVisible(False)
else:
    self.question_previous_question_button.setEnabled(True)
    self.question_previous_question_button.setVisible(True)
# If current question is last in homework, disables next page button (as there
is none). Otherwise,
# enables it
if self.current_question >= len(self.questions) - 1:
    self.question_next_question_button.setEnabled(False)
    self.question_next_question_button.setVisible(False)
else:
    self.question_next_question_button.setEnabled(True)
    self.question_next_question_button.setVisible(True)
# Fetches details of graph for given question
graph_details = ui_scripts.get_question_graph(question_id)
# If no graph, class function to set up and output graph, otherwise hides graph
output area
if graph_details:
    self.chart_setup(required=True, function=graph_details[0], min_x=graph_de-
tails[1], max_x=graph_details[2])
else:
    self.chart_setup(required=False)

```

4.6.3.10.1.2 Testing

4.6.3.10.1.2.1 With valid data:

Loading first page, previous question button is hidden:



The screenshot shows a window titled "Maths Quiz - Joe Down". The user is identified as "Stu". In the top right corner are "Main Menu" and "Logout" buttons. The main content area displays a question: "Find resultant of two forces". Below the question is a text block: "A force P acts on an object on a frictionless plane with magnitude 7N in the direction -100° to the x axis and another force, Q acts with magnitude 41N in the direction -36° to the x axis. Find the magnitude of the resultant force." There are four radio buttons for the answer: 63.11 (selected), 26.91, 61.56, and 44.52. Below the radio buttons are "Submit" and "Next Question" buttons. At the bottom right of the window is the name "Joe Down".

Reloading page after correctly answering selects correct answer, disables all question response inputs and outputs that question is correct:

Maths Quiz - Joe Down

Stu

Main Menu Logout

Find resultant of two forces

A force P acts on an object on a frictionless plane with magnitude 25N in the direction 77° to the x axis and another force, Q acts with magnitude 83N in the direction -64° to the x axis. Find the direction relative to the x axis of the resultant force.

-44.0
 -50.1
 -43.1
 -58.1

Submit

Correct

[Previous Question](#) [Next Question](#)

Joe Down

Loading a question with a graph loads and draws the graph:

Maths Quiz - Joe Down

Stu

Main Menu Logout

Simpson's Rule

Approximate $\int 19*x \, dx$ using Simpson's Rule with 4 strips between $x = 3$ and $x = 5$

30.6
 218.1
 152.0
 52.8

Submit

[Previous Question](#) [Next Question](#)

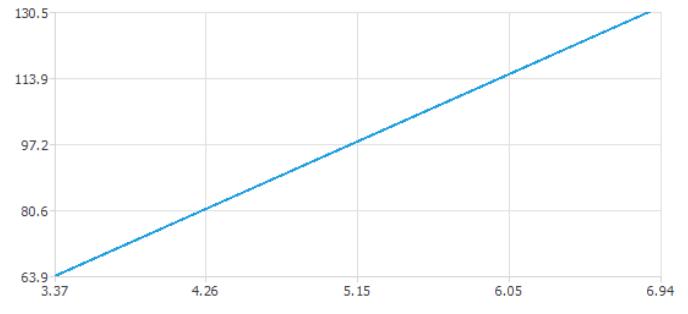
Joe Down

Reloading page (going to next page and returning) results in different answer ordering and selection is reset to first item:

Maths Quiz - Joe Down

 Stu Main Menu Logout

Simpson's Rule
Approximate $\int 19x^2 dx$ using Simpson's Rule with 4 strips between $x = 3$ and $x = 5$



52.8
 218.1
 30.6
 152.0

Submit

Previous Question **Next Question**

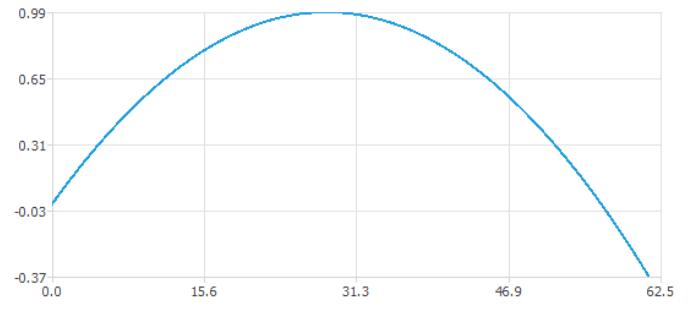
Joe Down

Loading last page, next question button is hidden:

Maths Quiz - Joe Down

 Stu Main Menu Logout

Projectile
A projectile is launched with velocity 63 at an angle of 45° to the horizontal. Find horizontal displacement after 0.89 seconds



61.60
 67.02
 54.11
 55.93

Submit

Previous Question

Joe Down

4.6.3.10.1.2.2 With Invalid data:

N/A, page is either loaded correctly or not, it cannot be made to load incorrectly as all question data is pre-validated by the question class

4.6.3.10.1.3 Evaluation

Question page correctly loads with buttons enabled/disabled as required, question order randomised, and a graph drawn where applicable

4.6.3.10.2 Function to set up buttons

4.6.3.10.2.1 Final Code

```
def question_page_button_setup(self):
    self.question_previous_question_button.clicked.connect(lambda: self.question_page_previous_page())
    self.question_next_question_button.clicked.connect(lambda: self.question_page_next_page())
    self.question_submit_button.clicked.connect(lambda: self.question_page_submit_response())
```

4.6.3.10.2.2 Testing

4.6.3.10.2.2.1 With valid data:

4.6.3.10.2.2.1.1 Previous question clicked

Maths Quiz - Joe Down

Main Menu Logout

Stu

Find resultant of two forces

A force P acts on an object on a frictionless plane with magnitude 25N in the direction 77° to the x axis and another force, Q acts with magnitude 83N in the direction -64° to the x axis. Find the direction relative to the x axis of the resultant force.

44.0
 -50.1
 -43.1
 -58.1

Submit

Correct

Previous Question Next Question

Joe Down

 Maths Quiz - Joe Down

Main Menu Logout

Stu

Find resultant of two forces

A force P acts on an object on a frictionless plane with magnitude 7N in the direction -100° to the x axis and another force, Q acts with magnitude 41N in the direction -36° to the x axis. Find the magnitude of the resultant force.

63.11
 26.91
 61.56
 44.52

Joe Down

Correctly loads previous page

4.6.3.10.2.2.1.2 Next question clicked

 Maths Quiz - Joe Down

Main Menu Logout

Stu

Find resultant of two forces

A force P acts on an object on a frictionless plane with magnitude 7N in the direction -100° to the x axis and another force, Q acts with magnitude 41N in the direction -36° to the x axis. Find the magnitude of the resultant force.

63.11
 26.91
 61.56
 44.52

Joe Down

 Maths Quiz - Joe Down

Main Menu Logout

Find resultant of two forces

A force P acts on an object on a frictionless plane with magnitude 25N in the direction 77° to the x axis and another force, Q acts with magnitude 83N in the direction -64° to the x axis. Find the direction relative to the x axis of the resultant force.

-50.1
 -44.0
 -43.1
 -58.1

Correct

Joe Down

Correctly loads next page

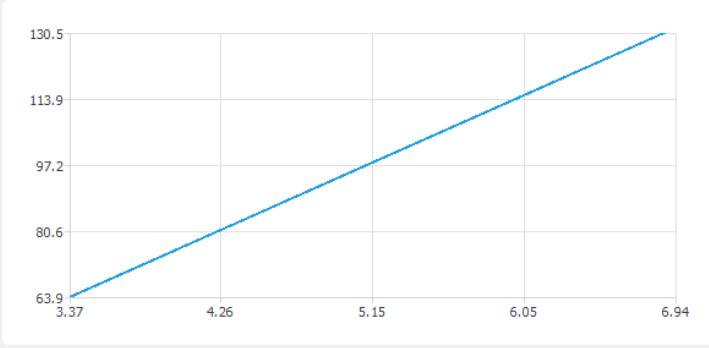
4.6.3.10.2.2.1.3 Submit button clicked

 Maths Quiz - Joe Down

Main Menu Logout

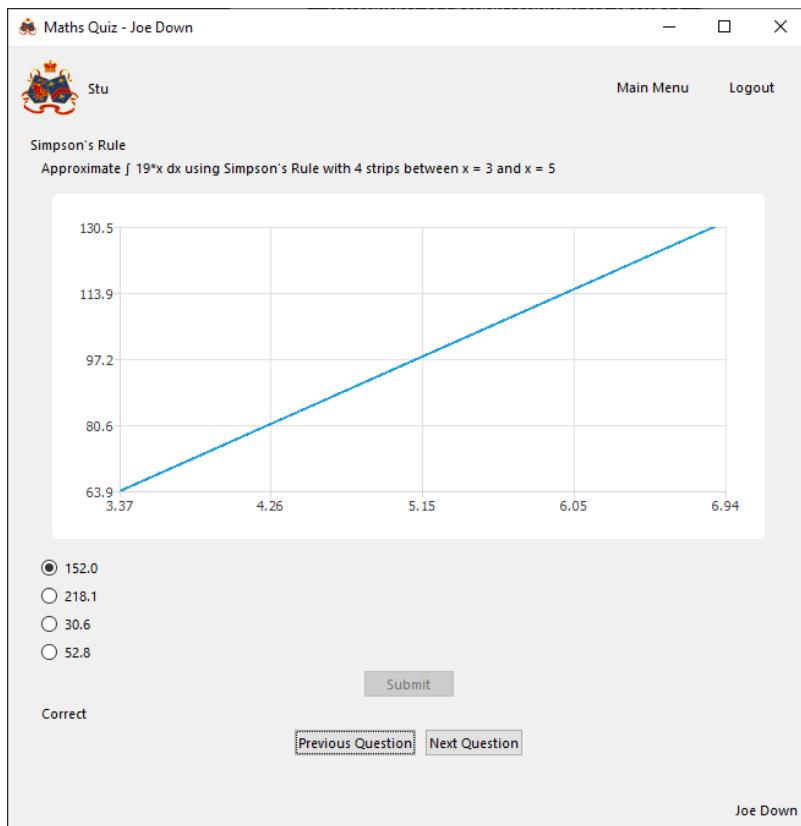
Simpson's Rule

Approximate $\int 19x^2 dx$ using Simpson's Rule with 4 strips between $x = 3$ and $x = 5$



152.0
 218.1
 30.6
 52.8

Joe Down



Correctly runs code to check given answer

4.6.3.10.2.2.2 With Invalid data:

N/A, button is either clicked or not clicked, cannot be given invalid input

4.6.3.10.2.2.3 Evaluation

All buttons on page correctly perform required functions

4.6.3.10.3 Function to load next question page

4.6.3.10.3.1 Final Code

```
def question_page_next_page(self):
    self.current_question += 1
    self.question_reset_page()
```

4.6.3.10.3.2 Testing

4.6.3.10.3.2.1 With valid data:

Maths Quiz - Joe Down

Stu

Main Menu Logout

Find resultant of two forces

A force P acts on an object on a frictionless plane with magnitude 7N in the direction -100° to the x axis and another force, Q acts with magnitude 41N in the direction -36° to the x axis. Find the magnitude of the resultant force.

63.11
 26.91
 61.56
 44.52

Joe Down

Maths Quiz - Joe Down

Stu

Main Menu Logout

Find resultant of two forces

A force P acts on an object on a frictionless plane with magnitude 25N in the direction 77° to the x axis and another force, Q acts with magnitude 83N in the direction -64° to the x axis. Find the direction relative to the x axis of the resultant force.

-50.1
 -44.0
 -43.1
 -58.1

Correct

Joe Down

Correctly loads next page

4.6.3.10.3.2.2 With Invalid data:

N/A, button is either clicked or not clicked, cannot be given invalid input

4.6.3.10.3.2.3 Evaluation

Next page correctly loaded

4.6.3.10.4 Function to load previous question page

4.6.3.10.4.1 Final Code

```
def question_page_previous_page(self):
    self.current_question -= 1
    self.question_reset_page()
```

4.6.3.10.4.2 Testing

4.6.3.10.4.2.1 With valid data:

Maths Quiz - Joe Down

Main Menu Logout

Stu

Find resultant of two forces

A force P acts on an object on a frictionless plane with magnitude 25N in the direction 77° to the x axis and another force, Q acts with magnitude 83N in the direction -64° to the x axis. Find the direction relative to the x axis of the resultant force.

-44.0
 -50.1
 -43.1
 -58.1

Submit

Correct

Previous Question Next Question

Joe Down

 Maths Quiz - Joe Down

Main Menu Logout

Stu

Find resultant of two forces

A force P acts on an object on a frictionless plane with magnitude 7N in the direction -100° to the x axis and another force, Q acts with magnitude 41N in the direction -36° to the x axis. Find the magnitude of the resultant force.

63.11
 26.91
 61.56
 44.52

Joe Down

Correctly loads previous page

4.6.3.10.4.2.2 With Invalid data:

N/A, button is either clicked or not clicked, cannot be given invalid input

4.6.3.10.4.2.3 Evaluation

Previous page correctly loaded

4.6.3.10.5 Function to submit question response

4.6.3.10.5.1 Final Code

```
def question_page_submit_response(self):
    # Gets id of current question
    question_id = self.questions[self.current_question][0]
    # Adds 1 to number of attempts as an attempt has been made
    ui_scripts.increment_user_attempts_at_question(self.current_user, question_id)
    # If radio button currently selected is the index of the correct answer, sets
    correct status in database to
        # True, disables ability to submit another answer and outputs that user was
    correct
    if (self.correct_answer_location == 1 and self.question_radio_a.isChecked()) or
    (
        self.correct_answer_location == 2 and self.question_radio_b.isChecked()) or (
        self.correct_answer_location == 3 and self.question_radio_c.isChecked()) or (
        self.correct_answer_location == 4 and self.question_radio_d.isChecked()):
        ui_scripts.mark_question_as_correct(self.current_user, question_id)
        self.question_submit_button.setEnabled(False)
        self.question_feedback_output.setText("Correct")
    # If radio button currently selected is not the index of the correct answer,
    outputs user incorrect
    else:
        self.question_submit_button.setEnabled(True)
        self.question_feedback_output.setText("Incorrect")
```

4.6.3.10.5.2 Testing

4.6.3.10.5.2.1 With valid data:

4.6.3.10.5.2.1.1 Incorrect response



Maths Quiz - Joe Down

Stu Main Menu Logout

Trapezium Rule
Approximate $\int 42x^4 + 3x^3 + x^2 dx$ using the Trapezium Rule with 6 strips between $x = 5$ and $x = 6$

39471.0
 34346.2
 39637.6
 41218.2

Submit

Incorrect

Previous Question **Next Question**

Joe Down

Correctly outputs that response was incorrect

4.6.3.10.5.2.1.2 Correct response

Maths Quiz - Joe Down

Stu Main Menu Logout

Trapezium Rule
Approximate $\int 42x^4 + 3x^3 + x^2 dx$ using the Trapezium Rule with 6 strips between $x = 5$ and $x = 6$

39471.0
 34346.2
 39637.6
 41218.2

Submit

Incorrect

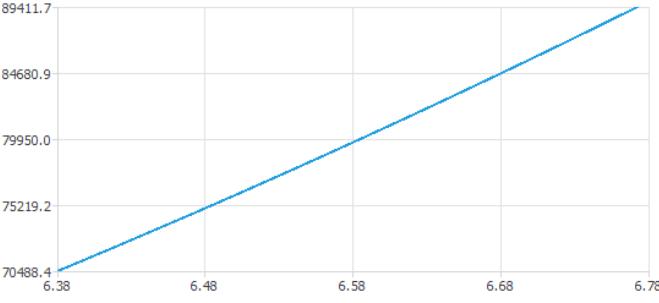
Previous Question **Next Question**

Joe Down

Maths Quiz - Joe Down

 Stu Main Menu Logout

Trapezium Rule
Approximate $\int 42x^4 + 3x^3 + x^2 dx$ using the Trapezium Rule with 6 strips between $x = 5$ and $x = 6$



39471.0
 34346.2
 39637.6
 41218.2

Correct

Joe Down

Correctly outputs response was correct and stores status in database

4.6.3.10.5.2.2 With Invalid data:

N/A, can only select either a correct radio button or an incorrect one (cannot select none) and submit button is either clicked or not, cannot be clicked incorrectly

4.6.3.10.5.2.3 Evaluation

Function correctly marks submission to question and stores result as needed

4.6.3.10.6 Function to set up and output chart

4.6.3.10.6.1 Issues during development

Initially the following code was implemented:

```

def chart_setup(self, required=False, function=None, min_x=0, max_x=10):
    # Sets up x as a variable in SymPy
    x = sympy.symbols('x')
    # Converts stores string equation to SymPy function
    function = sympy.sympify(function)

    # Enables and shows output box
    self.question_chart.setEnabled(True)
    self.question_chart.setVisible(True)
    # Defines structure of data to be added to graph using generic code. A spline
    series will interpolate values
    # between given points as a curve rather than simply drawing straight lines
    series: QtChart.QSplineSeries = QtChart.QSplineSeries()
    # Iterates through and finds values for the y axis for 100 x values spread
    evenly between the specified
    # minimum an maximum x values
    x_value: float = min_x
    precision: int = 100
    for counter in range(0, precision):
        y_value: float = function.subs(x, x_value)
        series.append(x_value, y_value)
        x_value: float = min_x + (((max_x - min_x) / precision) * counter)
    # Defines instance of chart and specifies axis ranges and inserts all data
    # noinspection PyArgumentList
    chart: QtChart.QChart = QtChart.QChart()
    chart.legend().hide()
    chart.addSeries(series)
    chart.createDefaultAxes()
    chart.axisX(series).setRange(min_x, max_x)
    # Sets chart output area to the generated chart
    self.question_chart.setChart(chart)

```

This would result in the following error when loading a question with no graph:

The screenshot shows the PyCharm IDE's 'Run' tab with a single run configuration named 'main (1)'. The output window displays a stack trace from a file named 'main.py' at line 529:

```

C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe C:/Users/Joe/PycharmProjects/Coursework/main.py
Traceback (most recent call last):
  File "C:/Users/Joe/PycharmProjects/Coursework/main.py", line 1175, in homework_select_table_clicked
    self.change_page(self.page_dictionary['question_page'])
  File "C:/Users/Joe/PycharmProjects/Coursework/main.py", line 134, in change_page
    self.reset_page(index)
  File "C:/Users/Joe/PycharmProjects/Coursework/main.py", line 174, in reset_page
    self.question_reset_page()
  File "C:/Users/Joe/PycharmProjects/Coursework/main.py", line 473, in question_reset_page
    self.chart_setup(required=False)
  File "C:/Users/Joe/PycharmProjects/Coursework/main.py", line 529, in chart_setup
    y_value: float = function.subs(x, x_value)
AttributeError: 'NoneType' object has no attribute 'subs'

Process finished with exit code -1073740791 (0xC0000409)

```

This is because there is no graph equation to be found and so when inserting data into the graph there is no equation from which to derive y values and so the code breaks. This was fixed by implementing the following if statement with code contained within the else for drawing the graph so it is only attempted if a graph is present:

```

# If no function required or a function is not passed, disables and hides output
box
if not required or function is None:
    self.question_chart.setEnabled(False)
    self.question_chart.setVisible(False)
# If graph equation given:
else:

```

4.6.3.10.6.2 Final Code

```
def chart_setup(self, required=False, function=None, min_x=0, max_x=10):
    # Sets up x as a variable in SymPy
    x = sympy.symbols('x')
    # Converts stores string equation to SymPy function
    function = sympy.sympify(function)
    # If no function required or a function is not passed, disables and hides output box
    if not required or function is None:
        self.question_chart.setEnabled(False)
        self.question_chart.setVisible(False)
    # If graph equation given:
    else:
        # Enables and shows output box
        self.question_chart.setEnabled(True)
        self.question_chart.setVisible(True)
        # Defines structure of data to be added to graph using generic code. A spline series will interpolate values
        # between given points as a curve rather than simply drawing straight lines
        series: QtChart.QSplineSeries = QtChart.QSplineSeries()
        # Iterates through and finds values for the y axis for 100 x values spread evenly between the specified
        # minimum an maximum x values
        x_value: float = min_x
        precision: int = 100
        for counter in range(0, precision):
            y_value: float = function.subs(x, x_value)
            series.append(x_value, y_value)
            x_value: float = min_x + (((max_x - min_x) / precision) * counter)
        # Defines instance of chart and specifies axis ranges and inserts all data
        # noinspection PyArgumentList
        chart: QtChart.QChart = QtChart.QChart()
        chart.legend().hide()
        chart.addSeries(series)
        chart.createDefaultAxes()
        chart.axisX(series).setRange(min_x, max_x)
        # Sets chart output area to the generated chart
        self.question_chart.setChart(chart)
```

4.6.3.10.6.3 Testing

4.6.3.10.6.3.1 With valid data:

4.6.3.10.6.3.1.1 No graph provided

Maths Quiz - Joe Down

Stu

Main Menu Logout

Find resultant of two forces

A force P acts on an object on a frictionless plane with magnitude 25N in the direction 77° to the x axis and another force, Q acts with magnitude 83N in the direction -64° to the x axis. Find the direction relative to the x axis of the resultant force.

-58.1
 -44.0
 -43.1
 -50.1

Correct

[Previous Question](#) [Next Question](#)

Joe Down

Graph output is disabled

4.6.3.10.6.3.1.2 Graph provided

Generated graph:

Maths Quiz - Joe Down

Stu

Main Menu Logout

Projectile
A projectile is launched with velocity 39 at an angle of 49 to the horizontal. Find vertical displacement after 4.51 seconds

44.2
33.1
22.1
11.0
0.0

0.0 32.9 65.8 98.7 131.6

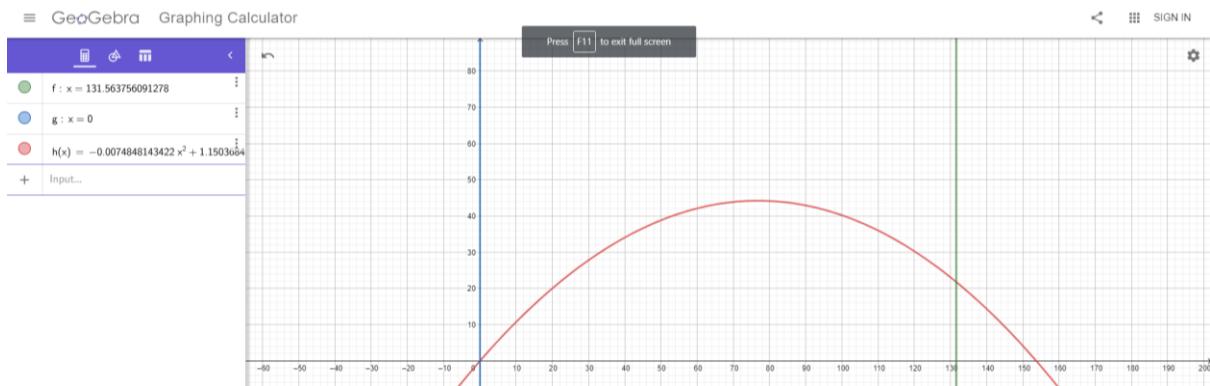
44.03
 24.20
 46.32
 33.08

Submit

Previous Question Next Question

Joe Down

Correct graph according to online graphing tool (<https://www.geogebra.org/graphing>):



Correct graph is successfully output within correct range (-0.0074848143421616*x**2 + 1.15036840722101*x between x = 0 and x = 131.563756091278)

4.6.3.10.6.3.2 With Invalid data:

N/A, graph is either provided or not provided and equation and limits are verified when storing graph

4.6.3.10.6.3.3 Evaluation

Function correctly outputs a graph under defined parameters, or hides the graph if none is defined for given question

4.6.3.11 Set homework page

4.6.3.11.1 Function to remove question

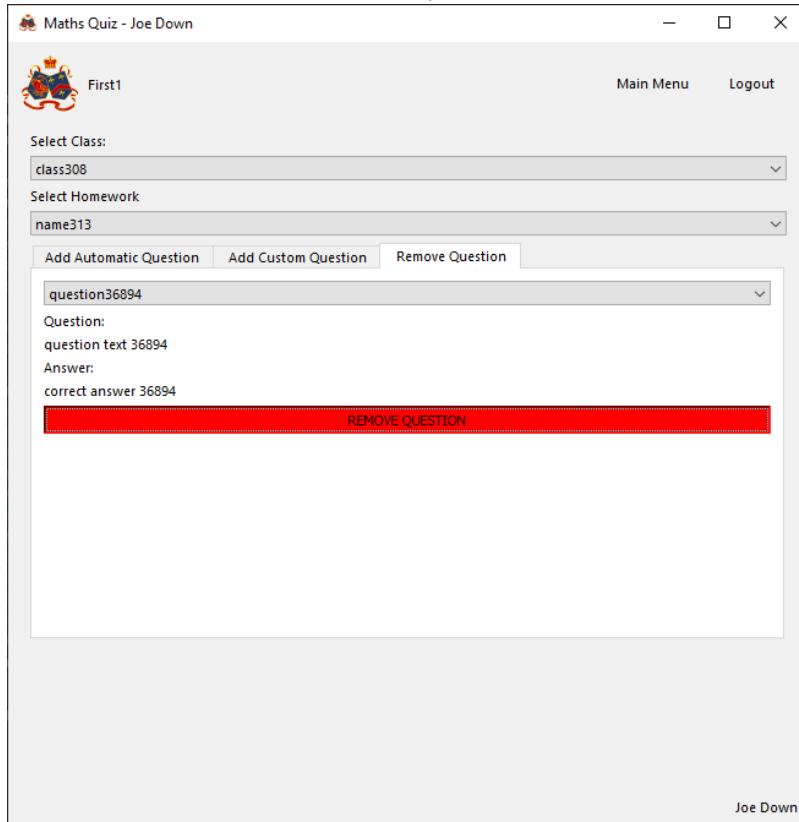
4.6.3.11.1.1 Final Code

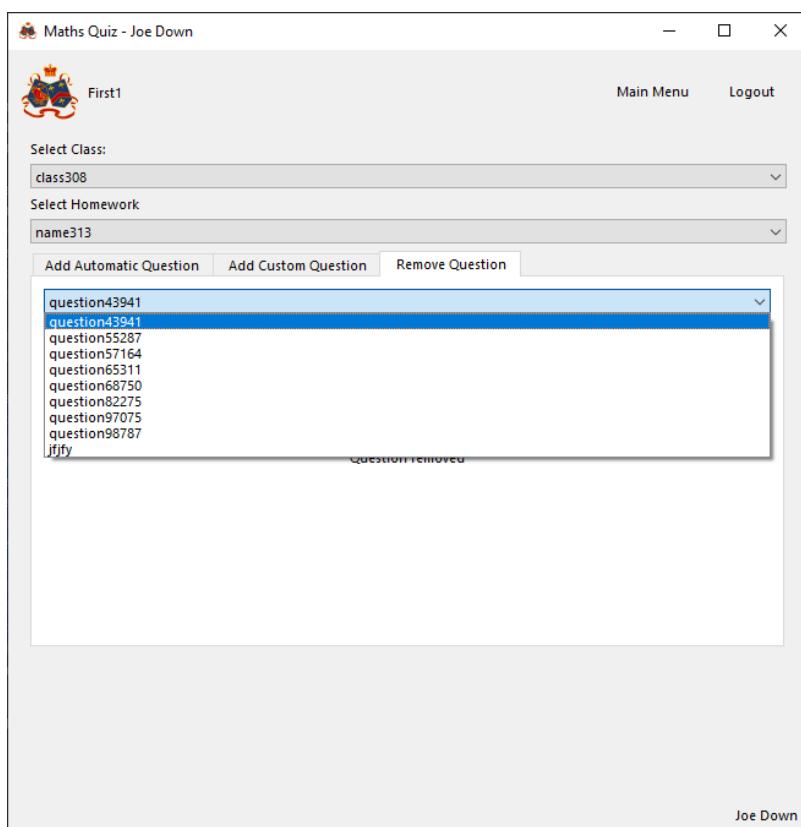
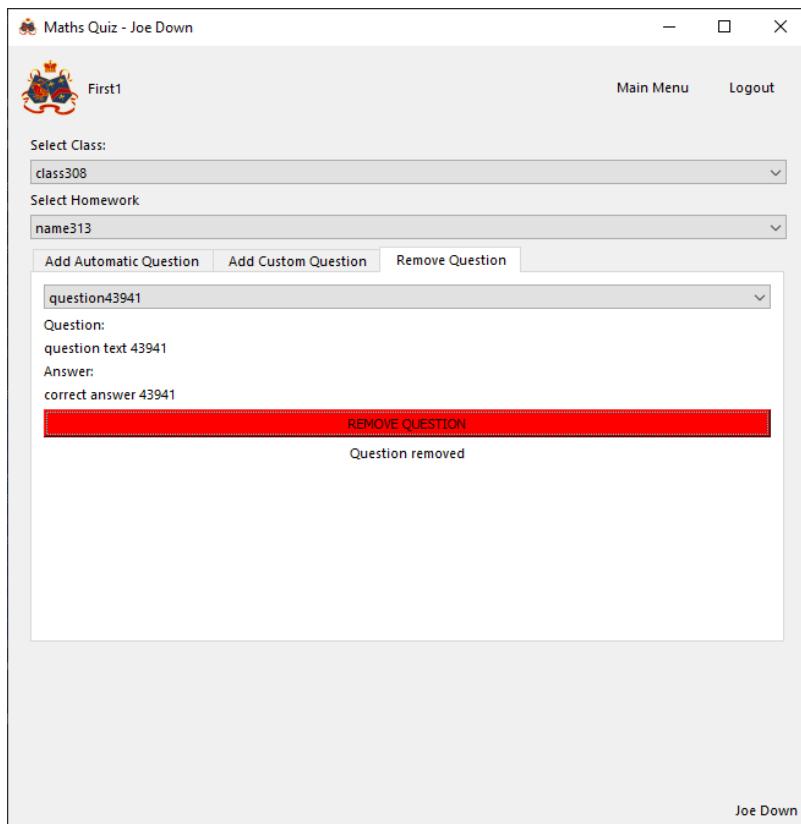
```
def set_homework_remove_question(self):
    # If homework contains questions:
    if len(self.questions) > 0:
        # Gets the ids of the currently selected homework and question
        question_id = self.questions[self.set_homework_question_combo_box.currentIndex()][0]
        homework_id = self.homework[self.set_homework_homework_combo_box.currentIndex()][0]
        # Calls function to remove question from homework from class
        ui_scripts.remove_question_from_homework(question_id, homework_id)
        # Runs function to reload question list
        self.set_homework_homework_change()
        self.set_homework_removed_output.setText("Question removed")
    # If homework contains no questions, outputs error message to tell user there
    are no questions which can be
    # removed
    else:
        self.set_homework_removed_output.setText("No question to remove")
```

4.6.3.11.1.2 Testing

4.6.3.11.1.2.1 With valid data:

4.6.3.11.1.2.1.1 Homework contains questions





Successfully removes question

4.6.3.11.1.2.1.2 Homework has no questions

The screenshot shows a Windows application window titled "Maths Quiz - Joe Down". The window has a title bar with standard minimize, maximize, and close buttons. In the top-left corner is a logo of a crown and a book. The top-right corner contains "Main Menu" and "Logout" links. The main area is titled "First1". It includes dropdown menus for "Select Class" (set to "class308") and "Select Homework" (set to "name8360"). Below these are three buttons: "Add Automatic Question", "Add Custom Question", and "Remove Question". A large text input field labeled "Question:" is present, with a red "REMOVE QUESTION" button at the bottom. The entire window is framed by a thick black border.

This screenshot is identical to the one above, showing the same application window and interface. However, the "Question:" input field now displays the message "No question to remove" in a small, dark font, indicating that there are no questions available for removal.

Informs user there are no questions to remove

4.6.3.11.1.2.2 With Invalid data:

N/A, invalid data cannot be entered intentionally by nature of a dropdown menu and a button blocking invalid data input

4.6.3.11.1.3 Evaluation

Function correctly removes the selected question from the selected homework

4.6.3.11.2 Function to add a custom question

4.6.3.11.2.1 *Final Code*

```

def set_homework_add_custom_question(self):
    # Clears status output for adding question
    self.set_homework_custom_question_added_output.setText("")
    # If no homework or class selected, informs user question cannot be added
    if len(self.current_classes) == 0 or len(self.homework) == 0:
        self.set_homework_custom_question_added_output.setText("No homework selected")
        return
    # If no question name given, informs user question cannot be added
    if self.set_homework_question_name_input.text() == "":
        self.set_homework_custom_question_added_output.setText("No question name entered")
        return
    # If no question text given, informs user question cannot be added
    if self.set_homework_question_input.toPlainText() == "":
        self.set_homework_custom_question_added_output.setText("No question text entered")
        return
    # If no correct answer given, informs user question cannot be added
    if self.set_homework_correct_answer_input.toPlainText() == "":
        self.set_homework_custom_question_added_output.setText("Correct answer required")
        return
    # If no first incorrect answer given, informs user question cannot be added
    if self.set_homework_answer_b_input.toPlainText() == "":
        self.set_homework_custom_question_added_output.setText("Incorrect answer 1 required")
        return
    # If first incorrect answer is not unique from correct answer, informs user question cannot be added
    if self.set_homework_answer_b_input.toPlainText() == self.set_homework_correct_answer_input.toPlainText():
        self.set_homework_custom_question_added_output.setText("Answer choices cannot match")
        return
    # If second incorrect answer not given, sets it to none
    if self.set_homework_answer_c_input.toPlainText() == "":
        answer_c = None
    # If second incorrect answer given:
    else:
        # If incorrect answer is not unique from correct answer or other incorrect answers, informs user
        # question cannot be added
        if self.set_homework_answer_c_input.toPlainText() in [self.set_homework_correct_answer_input.toPlainText(),
                                                          self.set_homework_answer_b_input.toPlainText()]:
            self.set_homework_custom_question_added_output.setText("Answer choices cannot match")
        return
    # If incorrect answer valid, stores it
    answer_c = self.set_homework_answer_c_input.toPlainText()
    # If second incorrect answer not given, sets it to none
    if self.set_homework_answer_d_input.toPlainText() == "":
        answer_d = None
    else:
        # If incorrect answer is not unique from correct answer or other incorrect answers, informs user
        # question cannot be added
        if self.set_homework_answer_d_input.toPlainText() in [self.set_homework_correct_answer_input.toPlainText(),
                                                          self.set_homework_answer_b_input.toPlainText(),
                                                          self.set_homework_answer_c_input.toPlainText()]:
            self.set_homework_custom_question_added_output.setText("Answer choices cannot match")
        return

```

```

# If incorrect answer valid, stores it
    answer_d = self.set_homework_answer_d_input.toPlainText()
# Creates instance of question class from given question data
    question = question_scripts.Question(self.set_homework_question_name_in-
put.text().casefold(), 1, 1,
                                         self.set_homework_question_input.toP-
lainText(),
                                         self.set_homework_correct_answer_in-
put.toPlainText(),
                                         self.set_homework_answer_b_input.toP-
lainText(), answer_c, answer_d)
# Saves question to database and stores it's index in a variable
    question_position: int = (question.save_question())
# Inserts reference to question in database to homework data
    ui_scripts.insert_question_into_homework(
        self.current_classes[self.set_homework_class_combo_box.currentIndex()][0],
        self.homework[self.set_homework_homework_combo_box.currentIndex()][0],
question_position)
# Resets page and informs user attempt was a success
    self.set_homework_homework_change()
    self.set_homework_custom_question_added_output.setText("Question Added")

```

4.6.3.11.2.2 Testing

4.6.3.11.2.2.1 With valid data

4.6.3.11.2.2.1.1 1 incorrect answer given

The screenshot shows a Windows-style application window titled "Maths Quiz - Joe Down". The window has a logo of a cartoon character at the top left. The main area contains several input fields and buttons:

- Select Class:** A dropdown menu showing "class308".
- Select Homework:** A dropdown menu showing "name313".
- Add Buttons:** Three buttons: "Add Automatic Question", "Add Custom Question" (which is highlighted in blue), and "Remove Question".
- Question Name:** An input field containing "qwerty".
- Question:** A large text area containing "asdf".
- Answer Fields:** A grid of four input fields labeled "Correct Answer:" (containing "a"), "Incorrect Answer 1:" (containing "b"), "Incorrect Answer 2:", and "Incorrect Answer 3:". All three incorrect answer fields are empty.
- Buttons at Bottom:** A blue button labeled "Add Custom Question" and a "Logout" link at the bottom right.

Maths Quiz - Joe Down

First1

Main Menu Logout

Select Class:
class308

Select Homework
name313

Add Automatic Question Add Custom Question Remove Question

Question Name:
qwerty

Question:
asdf

Correct Answer: Incorrect Answer 1: Incorrect Answer 2: Incorrect Answer 3:

a b Incorrect Answer Incorrect Answer

Add Custom Question

Question Added

Joe Down

Maths Quiz - Joe Down

First1

Main Menu Logout

Select Class:
class308

Select Homework
name313

Add Automatic Question Add Custom Question Remove Question

qwerty

Question:
asdf

Answer:
a

REMOVE QUESTION

Joe Down

Correctly informs user adding question was a success and inserts it into database

4.6.3.11.2.2.1.2 3 incorrect answers given

Maths Quiz - Joe Down

First1 Main Menu Logout

Select Class: class308

Select Homework name313

Add Automatic Question Add Custom Question Remove Question

Question Name: qwerty2

Question: asdf

Correct Answer: Incorrect Answer 1: Incorrect Answer 2: Incorrect Answer 3:

a b c d

Add Custom Question

Joe Down

Maths Quiz - Joe Down

First1 Main Menu Logout

Select Class: class308

Select Homework name313

Add Automatic Question Add Custom Question Remove Question

Question Name: qwerty2

Question: asdf

Correct Answer: Incorrect Answer 1: Incorrect Answer 2: Incorrect Answer 3:

a b c d

Add Custom Question

Question Added

Joe Down

 Maths Quiz - Joe Down

Main Menu Logout

First1

Select Class:
class308

Select Homework
name313

Add Automatic Question Add Custom Question Remove Question

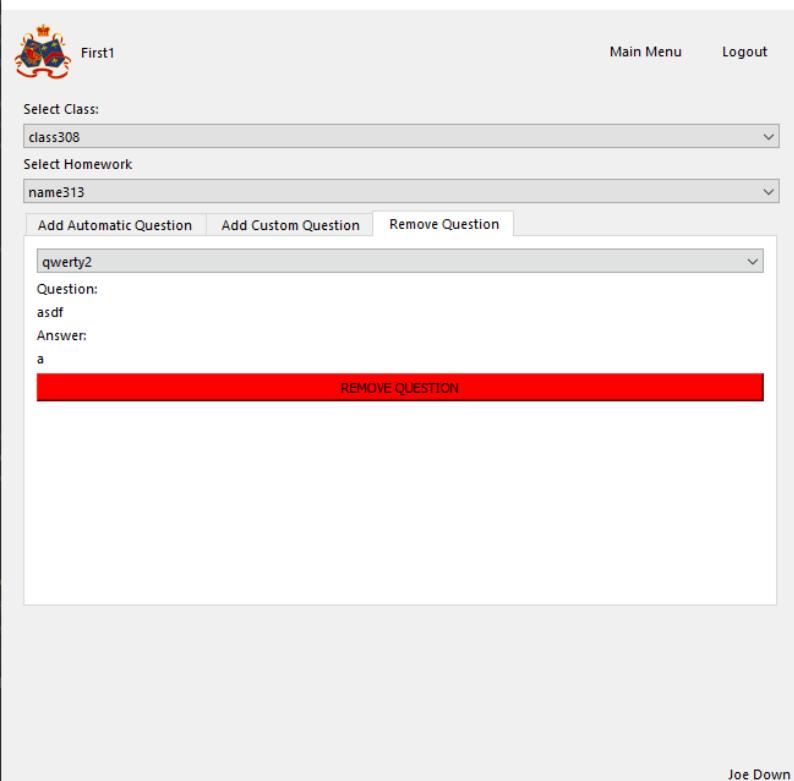
qwerty2

Question:
asdf

Answer:
a

REMOVE QUESTION

Joe Down



Correctly informs user adding question was a success and inserts it into database

4.6.3.11.2.2.2 With invalid data

4.6.3.11.2.2.2.1 No name given

 Maths Quiz - Joe Down

Main Menu Logout

First1

Select Class:
class308

Select Homework
name313

Add Automatic Question Add Custom Question Remove Question

Question Name:

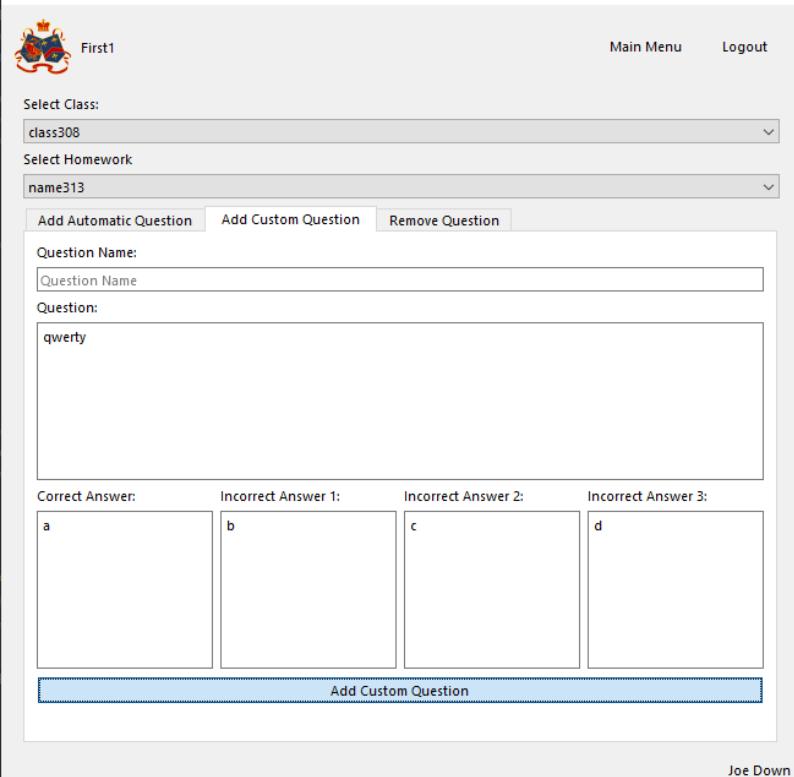
Question:
qwerty

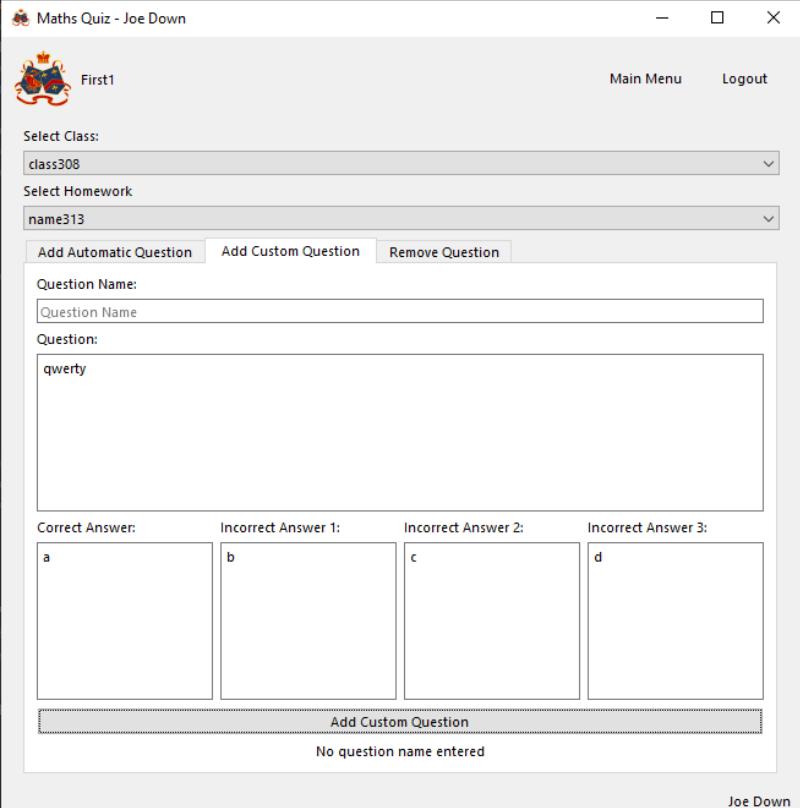
Correct Answer: Incorrect Answer 1: Incorrect Answer 2: Incorrect Answer 3:

a	b	c	d
---	---	---	---

Add Custom Question

Joe Down



 Maths Quiz - Joe Down

First1 Main Menu Logout

Select Class:
class308

Select Homework
name313

Add Automatic Question Add Custom Question Remove Question

Question Name:
Question Name

Question:
qwertv

Correct Answer: Incorrect Answer 1: Incorrect Answer 2: Incorrect Answer 3:
a b c d

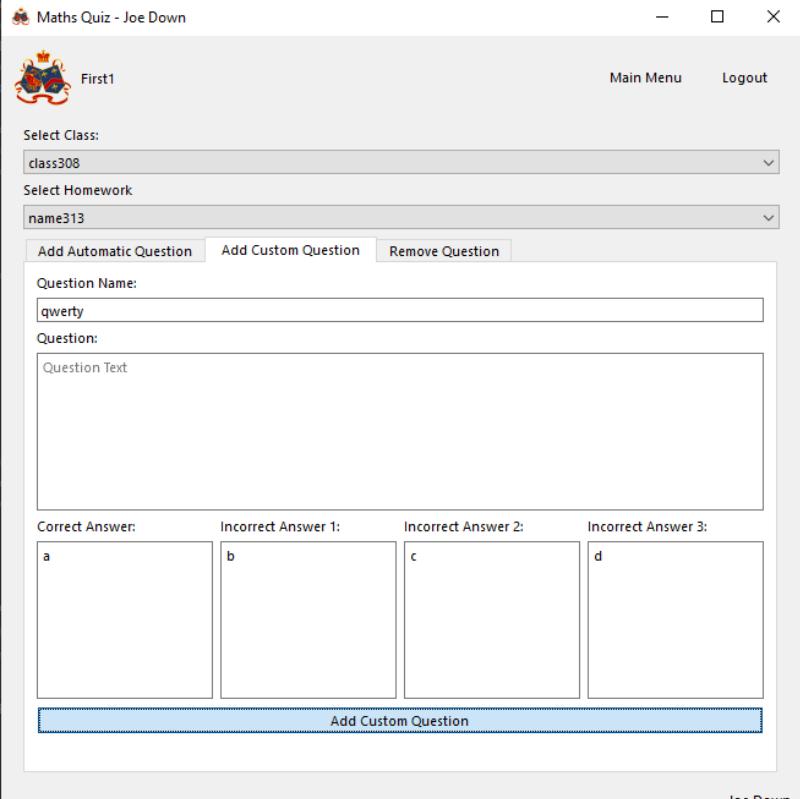
Add Custom Question

No question name entered

Joe Down

Correctly informs user that a question name is required

4.6.3.11.2.2.2.2 No question text given

 Maths Quiz - Joe Down

First1 Main Menu Logout

Select Class:
class308

Select Homework
name313

Add Automatic Question Add Custom Question Remove Question

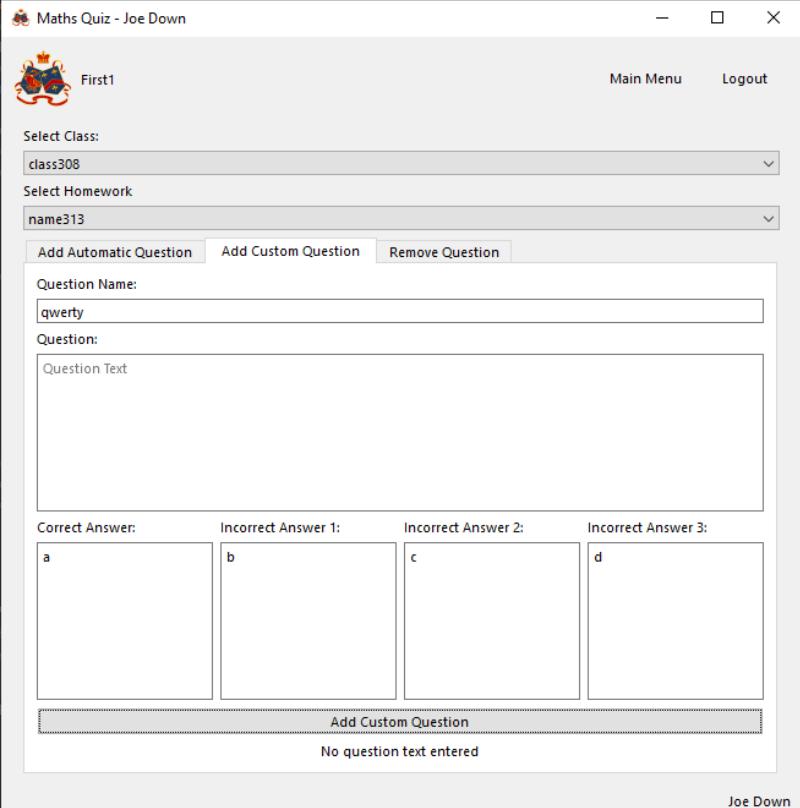
Question Name:
qwertv

Question:
Question Text

Correct Answer: Incorrect Answer 1: Incorrect Answer 2: Incorrect Answer 3:
a b c d

Add Custom Question

Joe Down

 Maths Quiz - Joe Down

Main Menu Logout

First1

Select Class:
class308

Select Homework
name313

Add Automatic Question Add Custom Question Remove Question

Question Name:
qwerty

Question:

Question Text

Correct Answer: Incorrect Answer 1: Incorrect Answer 2: Incorrect Answer 3:

a b c d

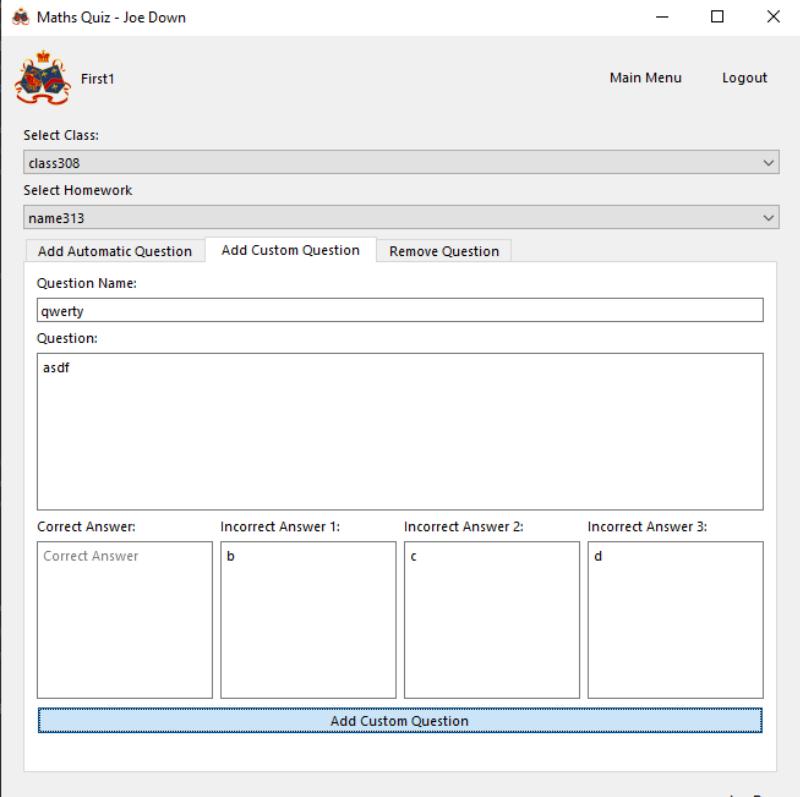
Add Custom Question

No question text entered

Joe Down

Correctly informs user that question text is required

4.6.3.11.2.2.2.3 No correct answer given

 Maths Quiz - Joe Down

Main Menu Logout

First1

Select Class:
class308

Select Homework
name313

Add Automatic Question Add Custom Question Remove Question

Question Name:
qwerty

Question:

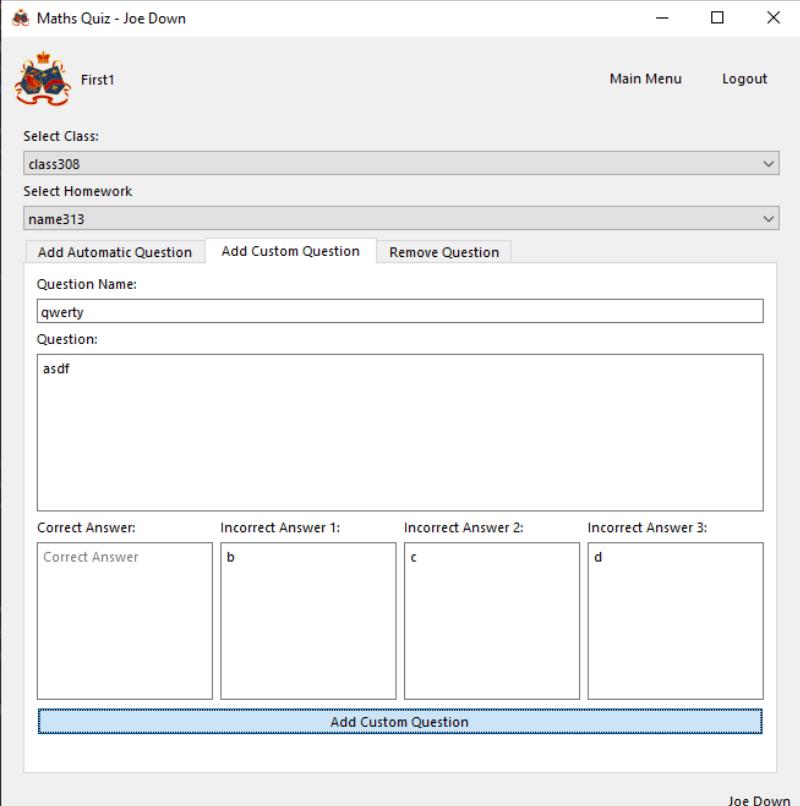
Question Text
asdf

Correct Answer: Incorrect Answer 1: Incorrect Answer 2: Incorrect Answer 3:

Correct Answer b c d

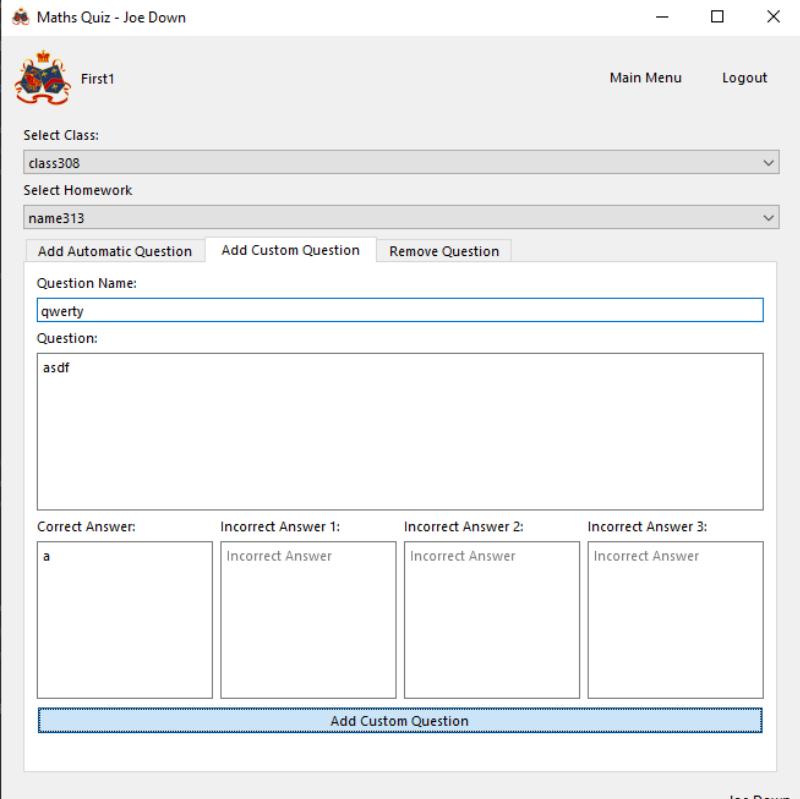
Add Custom Question

Joe Down

A screenshot of a Windows application window titled "Maths Quiz - Joe Down". The window has a title bar with icons for minimize, maximize, and close. Below the title bar is a menu bar with "Main Menu" and "Logout". On the left side, there is a user icon labeled "First1". The main area contains several input fields and buttons. At the top, there are dropdown menus for "Select Class" (set to "class308") and "Select Homework" (set to "name313"). Below these are three buttons: "Add Automatic Question", "Add Custom Question", and "Remove Question". A large text input field for "Question Name" contains "qwerty". Another text input field for "Question" contains "asdf". Below these are four input fields for "Correct Answer" and "Incorrect Answer 1", "Incorrect Answer 2", and "Incorrect Answer 3", each containing "b", "c", and "d" respectively. At the bottom is a blue button labeled "Add Custom Question". The bottom right corner of the window displays the name "Joe Down".

Correctly informs user that a correct answer is required

4.6.3.11.2.2.2.4 No incorrect answers given

A screenshot of the same "Maths Quiz - Joe Down" application window. The interface is identical to the first screenshot, but the validation message "Correct Answer is required" is displayed above the "Add Custom Question" button. The "Correct Answer" field contains "a", while the other three fields ("Incorrect Answer 1", "Incorrect Answer 2", and "Incorrect Answer 3") are empty.

Maths Quiz - Joe Down

First1 Main Menu Logout

Select Class: class308

Select Homework name313

Add Automatic Question Add Custom Question Remove Question

Question Name: qwerty

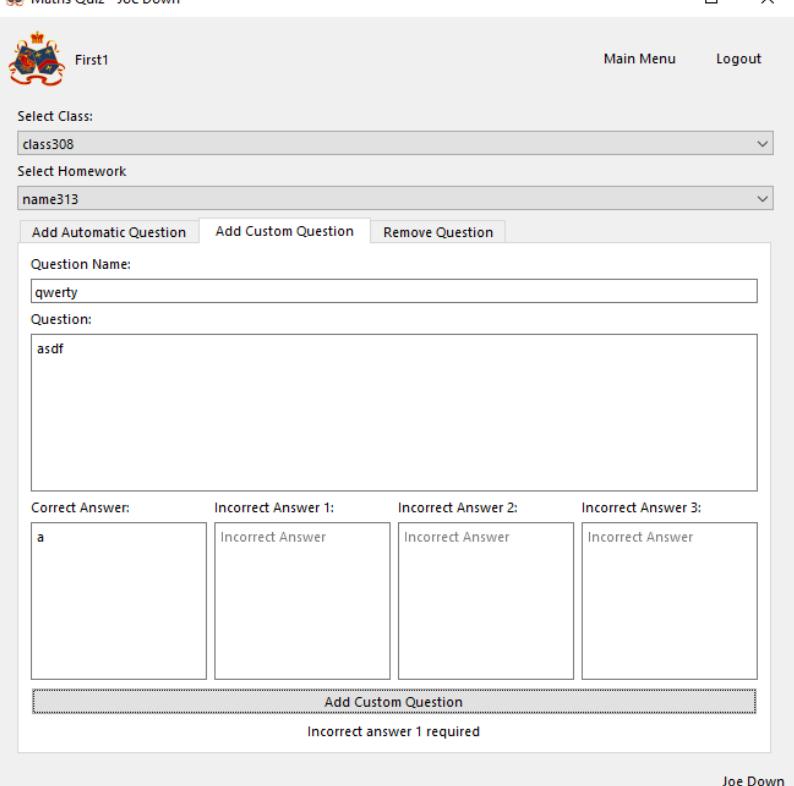
Question: asdf

Correct Answer: Incorrect Answer 1: Incorrect Answer 2: Incorrect Answer 3:

a	Incorrect Answer	Incorrect Answer	Incorrect Answer
---	------------------	------------------	------------------

Add Custom Question
Incorrect answer 1 required

Joe Down



Correctly informs user that an incorrect answer is required

4.6.3.11.2.2.2.5 Matching answers inputted

Maths Quiz - Joe Down

First1 Main Menu Logout

Select Class: class308

Select Homework name313

Add Automatic Question Add Custom Question Remove Question

Question Name: **qwerty**

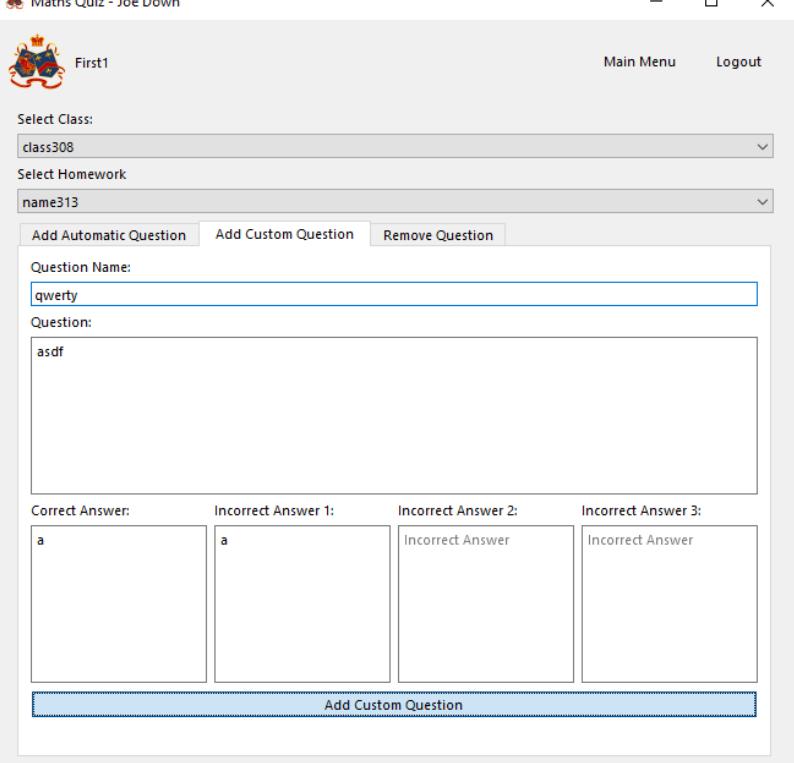
Question: asdf

Correct Answer: Incorrect Answer 1: Incorrect Answer 2: Incorrect Answer 3:

a	a	Incorrect Answer	Incorrect Answer
---	---	------------------	------------------

Add Custom Question

Joe Down



Maths Quiz - Joe Down

First1 Main Menu Logout

Select Class:
class308

Select Homework
name313

Add Automatic Question Add Custom Question Remove Question

Question Name:
qwerty

Question:
asdf

Correct Answer: Incorrect Answer 1: Incorrect Answer 2: Incorrect Answer 3:

a a Incorrect Answer Incorrect Answer

Add Custom Question

Answer choices cannot match

Joe Down

The screenshot shows a window titled "Maths Quiz - Joe Down". At the top right are standard window controls for minimize, maximize, and close. Below that, the user is identified as "First1" with links to "Main Menu" and "Logout". Under "Select Class:", "class308" is chosen from a dropdown. Under "Select Homework", "name313" is chosen from another dropdown. Below these are three buttons: "Add Automatic Question", "Add Custom Question", and "Remove Question". The main area contains fields for "Question Name" (containing "qwerty") and "Question" (containing "asdf"). Below these are four input fields labeled "Correct Answer", "Incorrect Answer 1", "Incorrect Answer 2", and "Incorrect Answer 3". The "Correct Answer" field contains "a", while both "Incorrect Answer 1" and "Incorrect Answer 2" also contain "a". The "Incorrect Answer 3" field is empty. At the bottom of the form, there is a button labeled "Add Custom Question" and a message "Answer choices cannot match". At the very bottom right of the window is the name "Joe Down".

Correctly informs user that all potential answers should be unique

4.6.3.11.2.3 Evaluation

Successfully validates inputs and, if valid, inserts question into database and establishes question to homework relationship

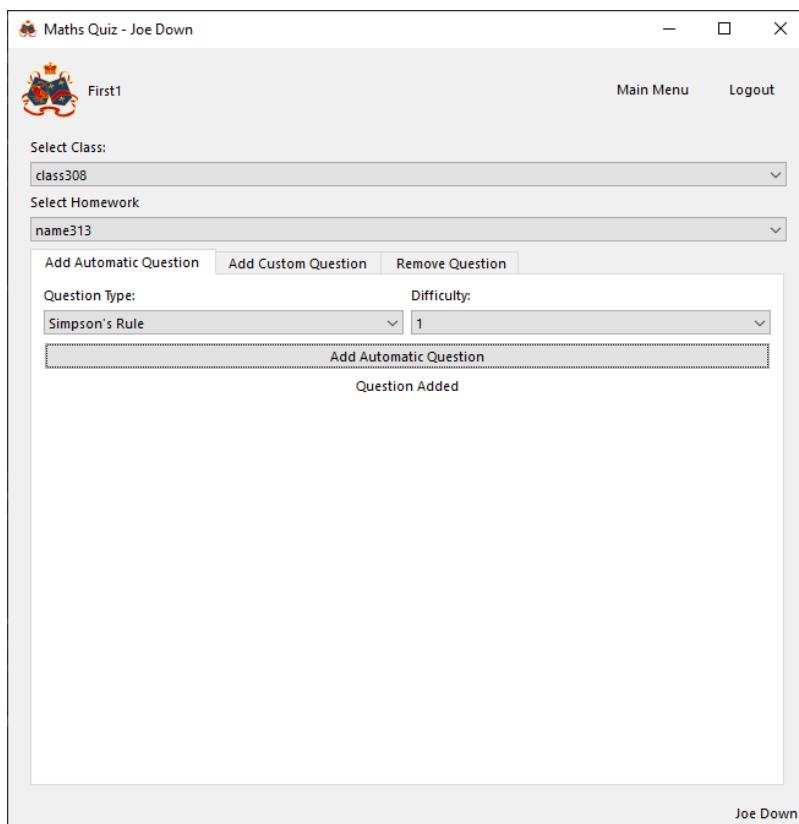
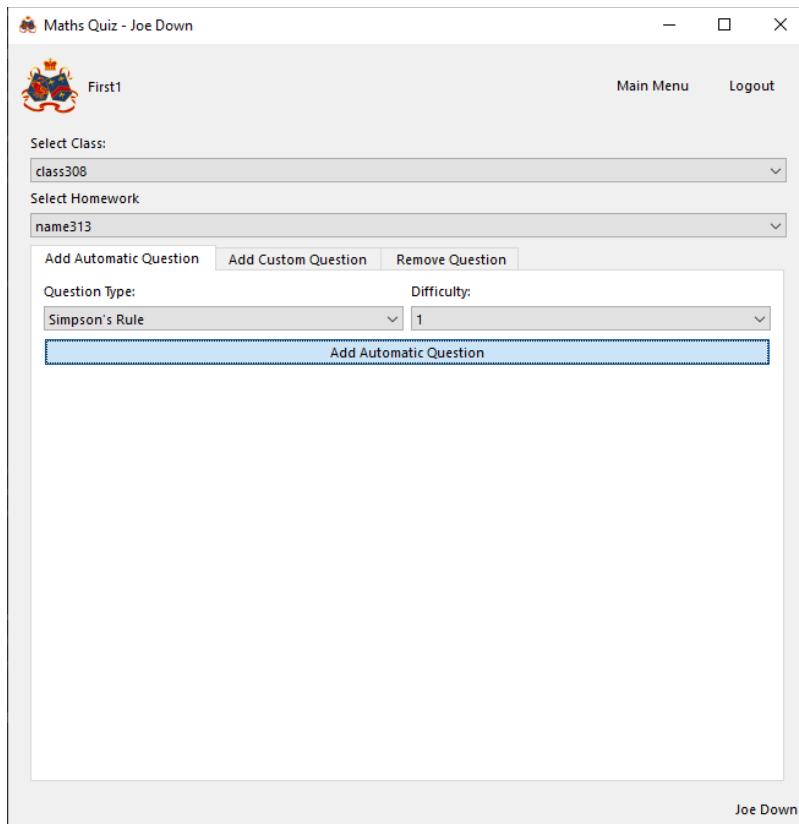
4.6.3.11.3 Function to add automatic question

4.6.3.11.3.1 Final code

```
def set_homework_add_automatic_question(self):
    # Clears status output for adding question
    self.set_homework_auto_question_added_output.setText("")
    # If no homework or class selected, informs user question cannot be added
    if len(self.current_classes) == 0 or len(self.homework) == 0:
        self.set_homework_auto_question_added_output.setText("No homework selected")
    return
    # Depending on index selected, runs scripts to generate a question of the selected type:
    if self.set_homework_type_combo_box.currentIndex() == 0:
        data: list = [questions.mechanics.find_resultant_of_two_forces(
            self.set_homework_difficulty_combo_box.currentIndex() + 1), None, None,
        None]
    elif self.set_homework_type_combo_box.currentIndex() == 1:
        data: list = questions.calculus.simpsons_rule(self.set_homework_diffi-
culty_combo_box.currentIndex() + 1)
    elif self.set_homework_type_combo_box.currentIndex() == 2:
        data: list = questions.calculus.trapezium_rule(self.set_homework_diffi-
culty_combo_box.currentIndex() + 1)
    elif self.set_homework_type_combo_box.currentIndex() == 3:
        data: list = questions.calculus.definite_integral(self.set_homework_diffi-
culty_combo_box.currentIndex() + 1)
    elif self.set_homework_type_combo_box.currentIndex() == 4:
        data: list = questions.mechanics.projectile(self.set_homework_diffi-
culty_combo_box.currentIndex() + 1)
    else:
        # Program errors if invalid index selected, though nature of a dropdown menu means this cannot ever occur
        raise IndexError
    # Gets back data needed to set up graph and link question to homework
    question: question_scripts.Question = data[0]
    function: str = data[1]
    minimum_x: float = data[2]
    maximum_x: float = data[3]
    question_position: int = (question.save_question())
    # Inserts reference to question in database to homework data
    ui_scripts.insert_question_into_homework(
        self.current_classes[self.set_homework_class_combo_box.currentIndex()][0],
        self.homework[self.set_homework_homework_combo_box.currentIndex()][0],
        question_position)
    # If graph details were returned during question generation, inserts an entry into the graph table to give
    # graph data for question added
    if function is not None and minimum_x is not None and maximum_x is not None:
        ui_scripts.set_question_graph(question_position, str(function), float(mini-
mum_x), float(maximum_x))
    # Resets page and informs user attempt was a success
    self.set_homework_homework_change()
    self.set_homework_auto_question_added_output.setText("Question Added")
    return
```

4.6.3.11.3.2 Testing

4.6.3.11.3.2.1 With valid data



Maths Quiz - Joe Down

First1 Main Menu Logout

Select Class:
class308

Select Homework
name313

Add Automatic Question Add Custom Question Remove Question

Simpson's Rule

Question:
Approximate $\int 92^x dx$ using Simpson's Rule with 6 strips between $x = 2$ and $x = 10$

Answer:
4416.0

REMOVE QUESTION

Joe Down

Correctly generates and inserts question of selected topic into homework and informs user adding question was successful

4.6.3.11.3.2.2 With invalid data

N/A, nature of dropdown box means an incorrect input cannot be given

4.6.3.11.3.3 Evaluation

Successfully generates question of the selected type, inserts question into database and establishes question to homework relationship

4.6.3.12 Previous scores page

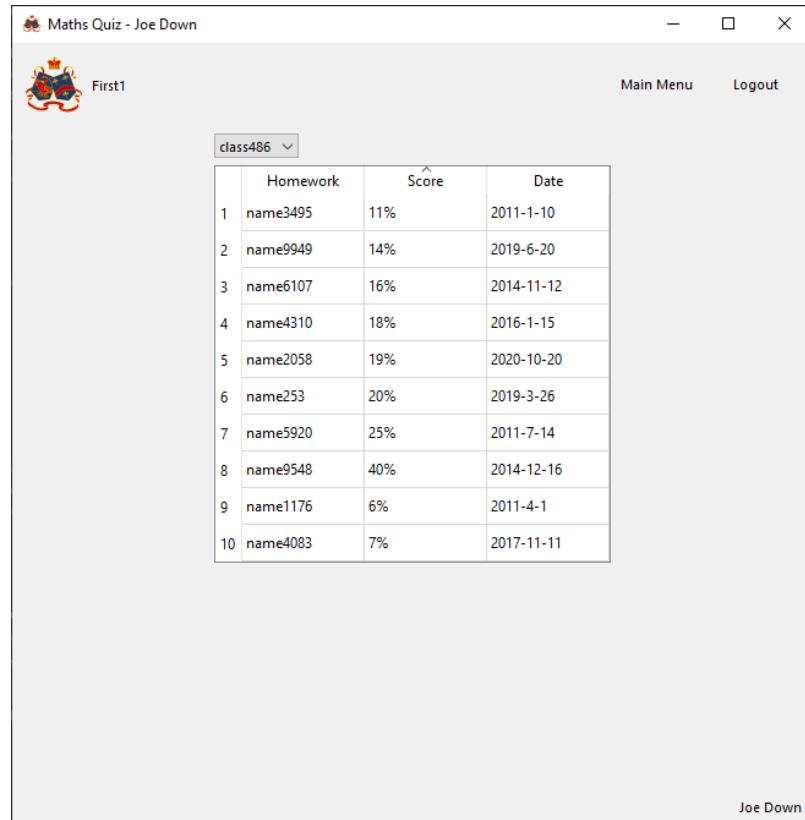
4.6.3.12.1 Function to fill scores table

4.6.3.12.1.1 Final Code

```
def previous_scores_update_table(self):
    # Clears table to allow for new values
    self.previous_scores_table.clearContents()
    # If user is any classes, populates table with homework scores
    if len(self.current_classes) != 0:
        homework_ids: list = []
        # For selected, id of every homework is appended to an array
        for each_homework in ui_scripts.get_homework_of_class(
            self.current_classes[self.previous_scores_class_combo_box.currentIndex()][0]):
            homework_ids.append(each_homework[0])
        # Inserts all homework data for class into table
        self.previous_scores_table.setRowCount(len(homework_ids))
        row_counter: int = -1
        for each_homework in homework_ids:
            row_counter += 1
            data: tuple = ui_scripts.get_homework_score(self.current_user,
each_homework, self.current_classes[
                self.previous_scores_class_combo_box.currentIndex()][0])
            self.previous_scores_table.setItem(row_counter, 0, QtWidgets.QTable-
WidgetItem(data[0]))
            self.previous_scores_table.setItem(row_counter, 1, QtWidgets.QTable-
WidgetItem("{}%".format(data[1])))
            self.previous_scores_table.setItem(row_counter, 2, QtWidgets.QTable-
WidgetItem(data[2]))
```

4.6.3.12.1.2 Testing

4.6.3.12.1.2.1 With valid data



	Homework	Score	Date
1	name3495	11%	2011-1-10
2	name9949	14%	2019-6-20
3	name6107	16%	2014-11-12
4	name4310	18%	2016-1-15
5	name2058	19%	2020-10-20
6	name253	20%	2019-3-26
7	name5920	25%	2011-7-14
8	name9548	40%	2014-12-16
9	name1176	6%	2011-4-1
10	name4083	7%	2017-11-11

Loading page successfully lists all homework for the given class along with the date and the correctly calculated score (for detail on score calculation see database scripts section)

4.6.3.12.1.2.2 With invalid data

N/A, no user inputs

4.6.3.12.1.3 Evaluation

Successfully fills scores table with appropriate data relating to the current student for each homework in the class

4.6.3.13 Admin page

4.6.3.13.1 Function to add user to class

4.6.3.13.1.1 Final Code

```
def admin_add_user_to_class(self):
    user: str = self.admin_username_input.text()
    # Stops scripts being able to run if the teacher has no classes to add a student to
    if len(self.current_classes) > 0:
        # If username exists runs scripts to add user to class
        if db_scripts.check_user_exists(user):
            user_id: int = db_scripts.get_user_id(user)
            class_id: int = self.current_classes[self.admin_class_user_combo_box.currentIndex()][0]
            # If user already in class outputs error
            if ui_scripts.check_student_in_class(user_id, class_id):
                self.admin_add_user_status_label.setText("User already in class")
            # If user not already in class runs scripts to add to class and outputs success message when done
            else:
                ui_scripts.add_student_to_class(user_id, class_id)
                self.admin_reset_page()
                self.admin_add_user_status_label.setText("Success")
            # If username does not exist outputs error
        else:
            self.admin_clear_labels()
            self.admin_add_user_status_label.setText("User does not exist")
    # Clears input box
    self.admin_username_input.setText("")
```

4.6.3.13.1.2 Testing

4.6.3.13.1.2.1 With valid data

Maths Quiz - Joe Down

First1

Main Menu Logout

Create Class:

Class Name Submit

Manage Class:

class308

Add User Remove User Add Homework Remove Homework

student1 Submit

Delete Class:

class308 REMOVE

Joe Down

This screenshot shows the Maths Quiz application interface. At the top, it displays 'Maths Quiz - Joe Down' and the user's name 'First1'. There are links for 'Main Menu' and 'Logout'. Below this, there are three main sections: 'Create Class', 'Manage Class', and 'Delete Class'. In the 'Create Class' section, there is a text input for 'Class Name' and a 'Submit' button. In the 'Manage Class' section, a dropdown menu shows 'class308' and buttons for 'Add User', 'Remove User', 'Add Homework', and 'Remove Homework'. A text input field contains 'student1' and a 'Submit' button. In the 'Delete Class' section, a dropdown menu shows 'class308' and a red 'REMOVE' button. At the bottom right, the user's name 'Joe Down' is displayed.

Maths Quiz - Joe Down

First1

Main Menu Logout

Create Class:

Class Name Submit

Manage Class:

class308

Add User Remove User Add Homework Remove Homework

Username Submit

Success

Delete Class:

class308 REMOVE

Joe Down

This screenshot shows the Maths Quiz application interface. It is similar to the previous one but includes a 'Success' message below the 'Manage Class' section. The 'Manage Class' section now includes a 'Username' input field and a 'Submit' button. The other sections and the user information at the bottom remain the same.

 Maths Quiz - Joe Down

First1 Main Menu Logout

Create Class:
 Submit

Manage Class:
class308

Delete Class:
class308

Joe Down

Successfully adds given student to class and informs user of success

4.6.3.13.1.2.2 With invalid data

4.6.3.13.1.2.2.1 Non-existent user

 Maths Quiz - Joe Down

First1 Main Menu Logout

Create Class:
 Submit

Manage Class:
class308

Delete Class:
class308

Joe Down

Maths Quiz - Joe Down

First1 Main Menu Logout

Create Class:

Manage Class:
class308

User does not exist

Delete Class:
class308

Joe Down

Successfully informs user that no such student exists

4.6.3.13.1.2.2.2 User already in class

Maths Quiz - Joe Down

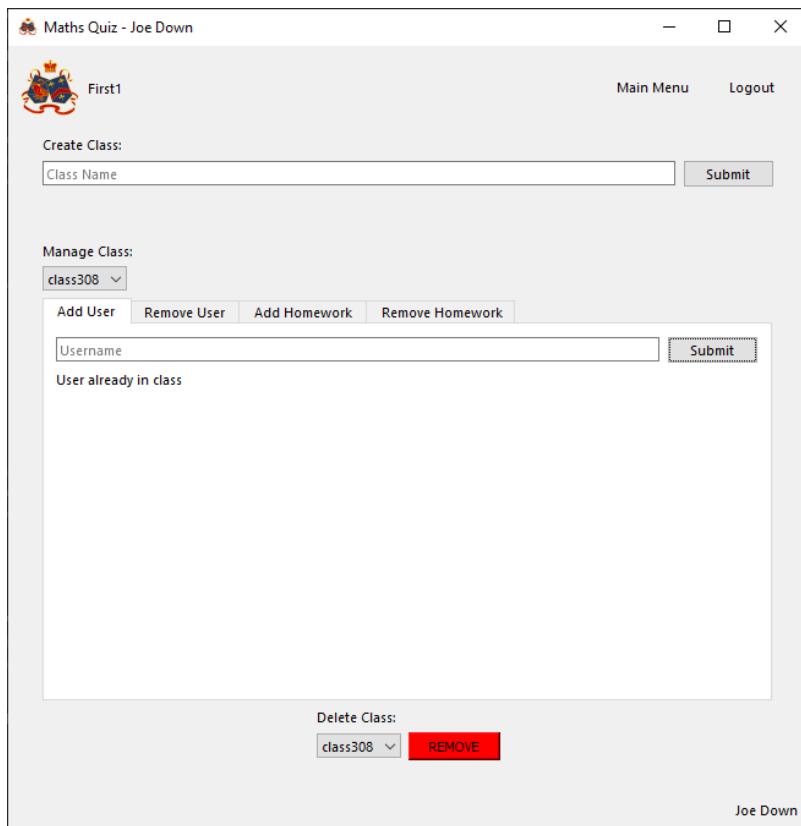
First1 Main Menu Logout

Create Class:

Manage Class:
class308

Delete Class:
class308

Joe Down



Successfully informs user that student is already in class

4.6.3.13.1.3 Evaluation

Successfully validates and then adds the user matching the inputted username to the selected class

4.6.3.13.2 Function to remove user from class

4.6.3.13.2.1 Final Code

```
def admin_remove_user_from_class(self):
    # Stops scripts being able to run if a student and/or class is not selected
    if len(self.class_users) > 0 and len(self.current_classes) > 0:
        # Gets the currently selected user and class and runs scripts to remove
        user from the class
        user_id: int = self.class_users[self.admin_username_combo_box.currentIndex()][0]
        class_id: int = self.current_classes[self.admin_class_user_combo_box.currentIndex()][0]
        ui_scripts.remove_student_from_class(user_id, class_id)
        # Outputs success message
        self.admin_reset_page()
        self.admin_remove_user_status_label.setText("Success")
    # If student and/or class not selected outputs error
    else:
        self.admin_clear_labels()
        self.admin_remove_user_status_label.setText("Select class and user")
```

4.6.3.13.2.2 Testing

4.6.3.13.2.2.1 With valid data

Maths Quiz - Joe Down

First1

Main Menu Logout

Create Class:

Class Name Submit

Manage Class:

class308

Add User Remove User Add Homework Remove Homework

student1 REMOVE

Delete Class:

class308 REMOVE

Joe Down

This screenshot shows the 'Manage Class' section of the Maths Quiz application. A dropdown menu is open for 'class308'. Under the 'Remove User' tab, 'student1' is selected. A red 'REMOVE' button is highlighted. Below the dropdown, there is a 'Delete Class:' section with a dropdown for 'class308' and a red 'REMOVE' button.

Maths Quiz - Joe Down

First1

Main Menu Logout

Create Class:

Class Name Submit

Manage Class:

class308

Add User Remove User Add Homework Remove Homework

student92 REMOVE

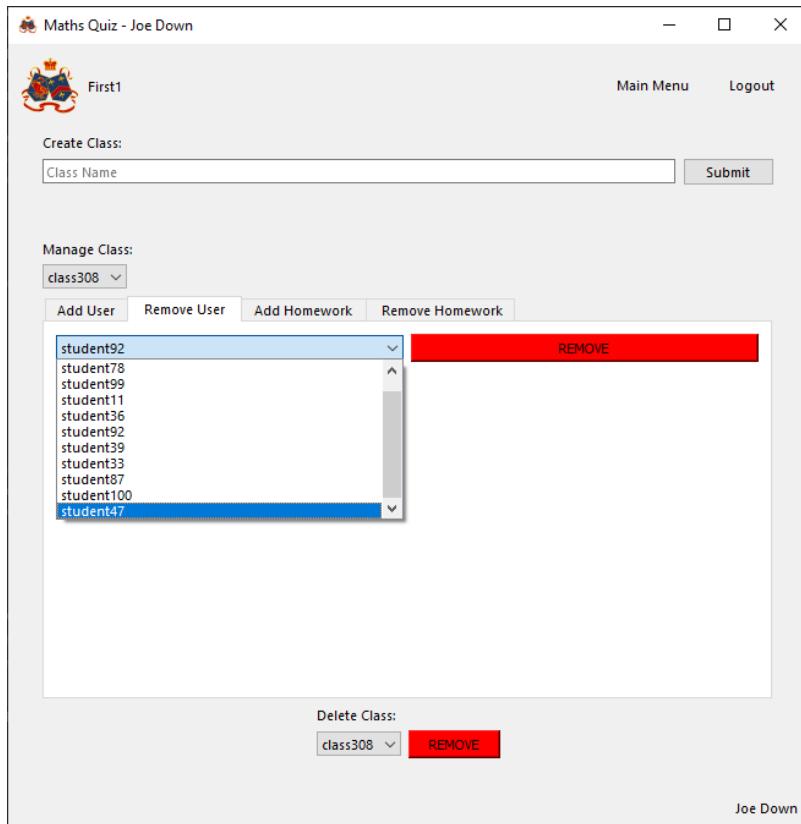
Success

Delete Class:

class308 REMOVE

Joe Down

This screenshot shows the 'Manage Class' section after the removal of 'student92'. The 'student92' entry in the dropdown has been removed, and a 'Success' message is displayed below the dropdown. The 'Delete Class:' section remains at the bottom of the page.



User selected is successfully removed from class

4.6.3.13.2.2.2 With invalid data

N/A, nature of dropdown menu means invalid removal cannot occur

4.6.3.13.2.3 Evaluation

Successfully removes the selected user from the class

4.6.3.13.3 Function to create class

4.6.3.13.3.1 Final Code

```
def admin_create_class(self):
    # If a name is entered for the class runs scripts to set it up in database and
    # outputs success message when done
    if self.admin_class_input.text() != '':
        ui_scripts.create_class(self.current_user, self.admin_class_input.text())
        self.admin_reset_page()
        self.admin_create_class_status_label.setText("Success")
    # If no class name is entered outputs error
    else:
        self.admin_clear_labels()
        self.admin_create_class_status_label.setText("Class must have name")
```

4.6.3.13.3.2 Testing

4.6.3.13.3.2.1 With valid data

Maths Quiz - Joe Down

First1

Main Menu Logout

Create Class:

Manage Class:

Add User Remove User Add Homework Remove Homework

Delete Class:

Joe Down

This screenshot shows the Maths Quiz application interface. At the top, there's a header with the title 'Maths Quiz - Joe Down' and a user icon labeled 'First1'. Below the header are 'Main Menu' and 'Logout' links. The main area has three sections: 'Create Class', 'Manage Class', and 'Delete Class'. In the 'Create Class' section, a text input field contains 'testclass' and a 'Submit' button is visible. In the 'Manage Class' section, a dropdown menu shows 'class308' and buttons for 'Add User', 'Remove User', 'Add Homework', and 'Remove Homework'. A 'Username' input field and a 'Submit' button are also present. In the 'Delete Class' section, a dropdown menu shows 'class308' and a red 'REMOVE' button. At the bottom right, the name 'Joe Down' is displayed.

Maths Quiz - Joe Down

First1

Main Menu Logout

Create Class:

Success

Manage Class:

Add User Remove User Add Homework Remove Homework

Delete Class:

Joe Down

This screenshot shows the Maths Quiz application interface, similar to the one above but with different content. It features a 'Create Class' section with a 'Class Name' input field and a 'Submit' button. Below it, a 'Success' message is displayed. The 'Manage Class' and 'Delete Class' sections are identical to the first screenshot, showing a dropdown menu with 'class308' and their respective buttons. The bottom right corner shows 'Joe Down'.

Maths Quiz - Joe Down

First1 Main Menu Logout

Create Class:
Class Name Submit

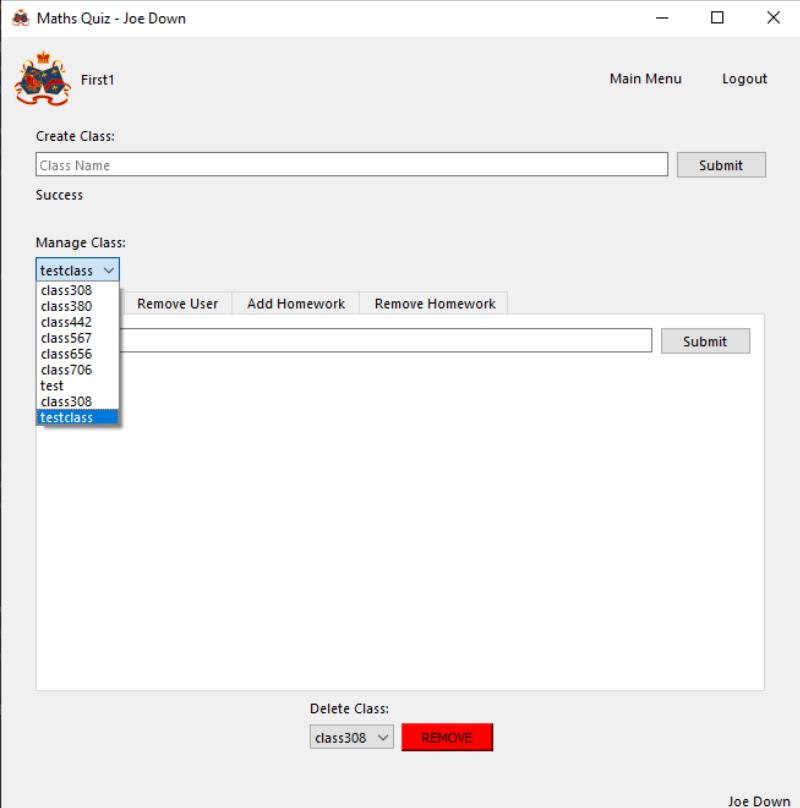
Success

Manage Class:
testclass

testclass
class308
class380
class442
class567
class656
class706
test
class308
testclass

Delete Class:
class308

Joe Down

A screenshot of a web-based application titled "Maths Quiz - Joe Down". The interface includes a header with the teacher's name and links for "Main Menu" and "Logout". Below the header, there's a section for creating a new class, which has been successfully completed ("Success"). A dropdown menu shows the created class "testclass". There are buttons for managing users, homework, and a submit button. At the bottom, there's a delete class section with "class308" selected and a red "REMOVE" button.

Successfully creates new class with given name under current teacher

4.6.3.13.3.2.2 With invalid data

4.6.3.13.3.2.2.1 No name given

Maths Quiz - Joe Down

First1 Main Menu Logout

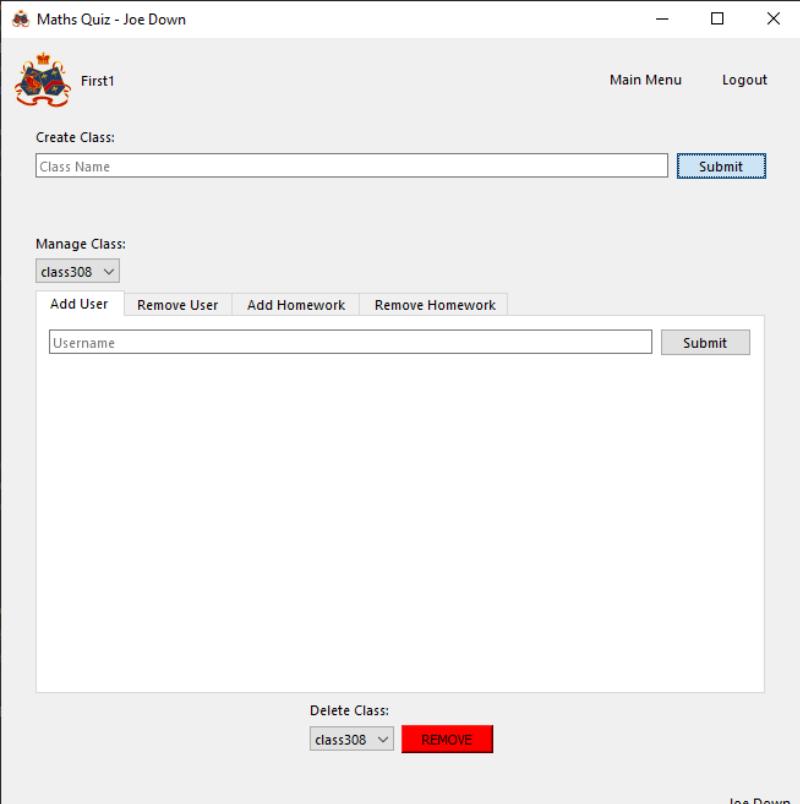
Create Class:
Class Name Submit

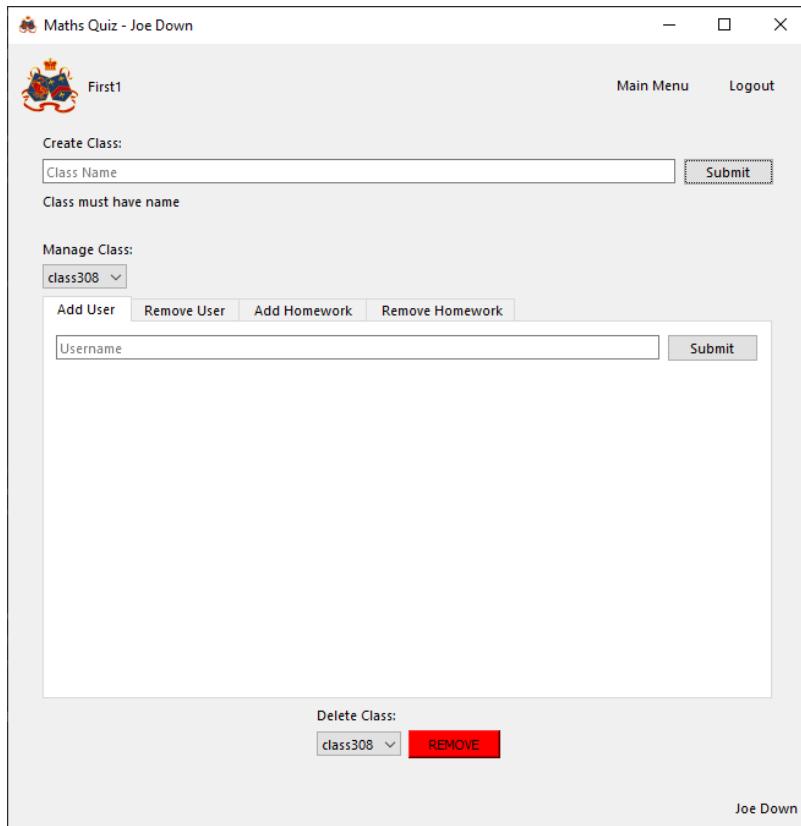
Manage Class:
class308

class308

Delete Class:
class308

Joe Down

A screenshot of the same application interface as the previous one, but with an error. The "Create Class" section shows an empty "Class Name" input field, which is highlighted with a blue dashed border. The "Submit" button is also highlighted with a blue dashed border, indicating it cannot be clicked until a valid name is provided.



Successfully informs user that class to be added must be named

4.6.3.13.3.3 Evaluation

Successfully validates class name and then creates class using given name with the current user as teacher

4.6.3.13.4 Function to remove class

4.6.3.13.4.1 Final Code

```
def admin_remove_class(self):
    # If there are no classes to select does not attempt to remove class
    if len(self.current_classes) > 0:
        # Runs scripts to remove class from database
        ui_scripts.remove_class(self.current_classes[self.admin_delete_class_combo_box.currentIndex()][0])
        self.admin_reset_page()
        # Outputs success message
        self.admin_delete_class_status_label.setText("Success")
    # If no class selected outputs error
    else:
        self.admin_reset_page()
        self.admin_delete_class_status_label.setText("No class selected")
```

4.6.3.13.4.2 Testing

4.6.3.13.4.2.1 With valid data

Maths Quiz - Joe Down

First1

Main Menu Logout

Create Class:

Class Name Submit

Manage Class:

class380

Add User Remove User Add Homework Remove Homework

Username Submit

Delete Class:

class380 REMOVE

Joe Down

Maths Quiz - Joe Down

First1

Main Menu Logout

Create Class:

Class Name Submit

Manage Class:

class442

Add User Remove User Add Homework Remove Homework

Username Submit

Delete Class:

class442 REMOVE

Success

Joe Down

Maths Quiz - Joe Down

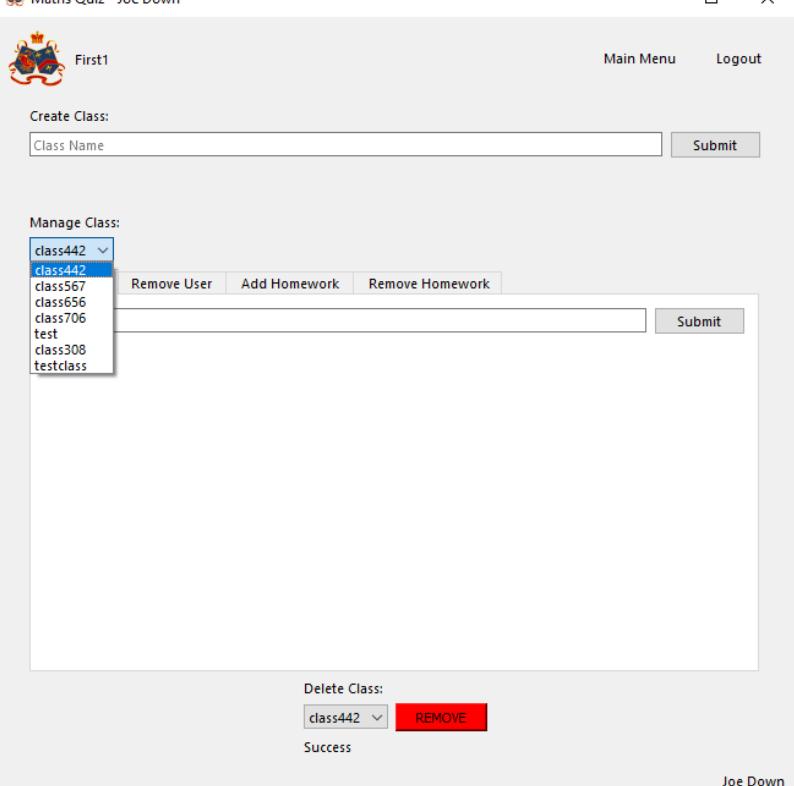
First1 Main Menu Logout

Create Class:
Class Name Submit

Manage Class:
class442 Add Homework Remove Homework
 Submit

Delete Class:
class442 Success

Joe Down



Successfully removes selected class

4.6.3.13.4.2.2 With invalid data

N/A, nature of dropdown menu means invalid removal cannot occur

4.6.3.13.4.3 Evaluation

Successfully removes the selected class

4.6.3.13.5 Function to create homework

4.6.3.13.5.1 Final Code

```
def admin_create_homework(self):
    # If class selected:
    if len(self.current_classes) > 0:
        # Creates a datetime formatted due date from the items selected in the calendar widget
        due_date: datetime.date = datetime.date(self.admin_due_date_calendar.selectedDate().year(),
                                                self.admin_due_date_calendar.selectedDate().month(),
                                                self.admin_due_date_calendar.selectedDate().day())
        # If no homework name given, outputs error and does not commit changes
        if self.admin_homework_name_input.text() == "":
            self.admin_add_homework_status_output.setText("Homework name required")
        # If no homework description given, outputs error and does not commit changes
        elif self.admin_homework_description_input.toPlainText() == "":
            self.admin_add_homework_status_output.setText("Homework description required")
        # If due date is in the past, outputs error and does not commit changes
        elif due_date <= datetime.date.today():
            self.admin_add_homework_status_output.setText("Homework due date must be in the future")
        else:
            # Creates homework entry in database and stores id in variable
            homework_id: int = ui_scripts.insert_new_homework(self.admin_homework_name_input.text(),
                                                               self.admin_home-
                                                               work_description_input.toPlainText())
            # Creates class to homework relationship in database
            ui_scripts.add_homework_to_class(
                self.current_classes[self.admin_class_user_combo_box.currentIndex()][0], homework_id, due_date)
            # Outputs success message and resets page
            self.admin_add_homework_status_output.setText("Homework Added")
            self.admin_homework_name_input.setText("")
            self.admin_homework_description_input.setText("")
            # noinspection PyArgumentList
            self.admin_due_date_calendar.setMinimumDate(QtCore.QDate.currentDate().addDays(1))
            # noinspection PyArgumentList
            self.admin_due_date_calendar.setSelectedDate(QtCore.QDate.currentDate().addDays(1))
            # If no class selected, outputs error and makes no changes
        else:
            self.admin_add_homework_status_output.setText("Class must be selected")
```

4.6.3.13.5.2 Testing

4.6.3.13.5.2.1 With valid data

Maths Quiz - Joe Down

First1

Main Menu Logout

Create Class:

Class Name Submit

Manage Class:

class442

Add User Remove User Add Homework Remove Homework

Homework Name: Due Date:

Description

qwerty

January, 2020

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	30	31	1	2	3	4	5
2	6	7	8	9	10	11	12
3	13	14	15	16	17	18	19
4	20	21	22	23	24	25	26
5	27	28	29	30	31	1	2
6	3	4	5	6	7	8	9

Add Homework

Delete Class:

class442 REMOVE

Joe Down

Maths Quiz - Joe Down

First1

Main Menu Logout

Create Class:

Class Name Submit

Manage Class:

class442

Add User Remove User Add Homework Remove Homework

Homework Name: Due Date:

Description

Homework Name

Homework Description

April, 2019

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
13	25	26	27	28	29	30	31
14	1	2	3	4	5	6	7
15	8	9	10	11	12	13	14
16	15	16	17	18	19	20	21
17	22	23	24	25	26	27	28
18	29	30	1	2	3	4	5

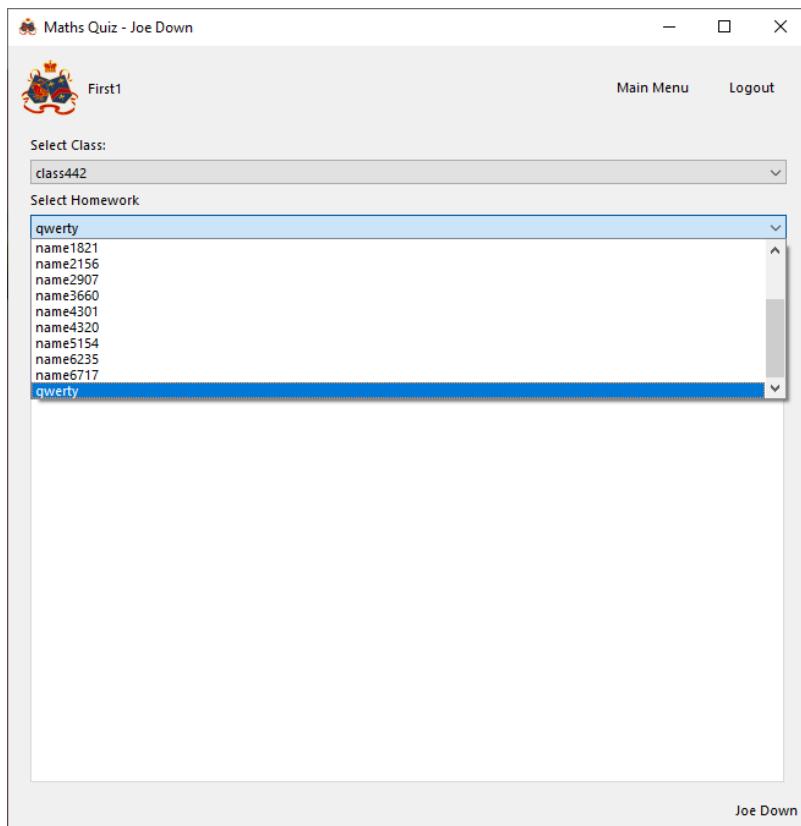
Add Homework

Homework Added

Delete Class:

class442 REMOVE

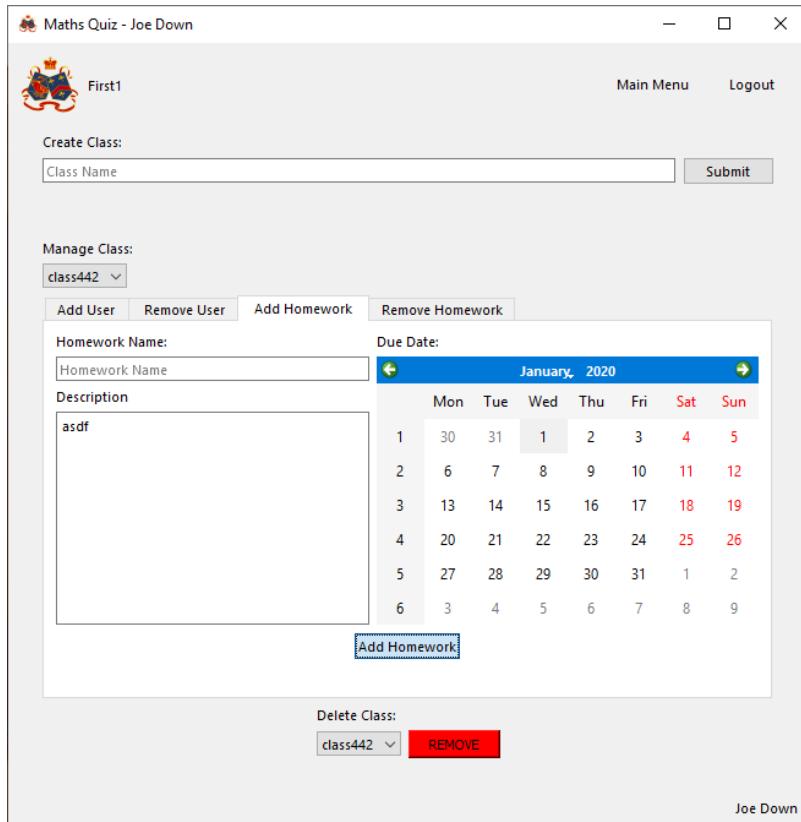
Joe Down



Successfully inserts new homework into class with given name, description and due date

4.6.3.13.5.2.2 With invalid data

4.6.3.13.5.2.2.1 No name



Maths Quiz - Joe Down



Main Menu Logout

Create Class:

Class Name Submit

Manage Class:

class442

Add User Remove User Add Homework Remove Homework

Homework Name:	Due Date:
<input type="text" value="Homework Name"/>	 January, 2020 
Description	Mon Tue Wed Thu Fri Sat Sun
asdf	1 30 31 1 2 3 4 5
	2 6 7 8 9 10 11 12
	3 13 14 15 16 17 18 19
	4 20 21 22 23 24 25 26
	5 27 28 29 30 31 1 2
	6 3 4 5 6 7 8 9

Homework name required

Delete Class:

class442 REMOVE

Joe Down

Successfully informs user that a name is required

4.6.3.13.5.2.2.2 No description

Maths Quiz - Joe Down



Main Menu Logout

Create Class:

Class Name Submit

Manage Class:

class442

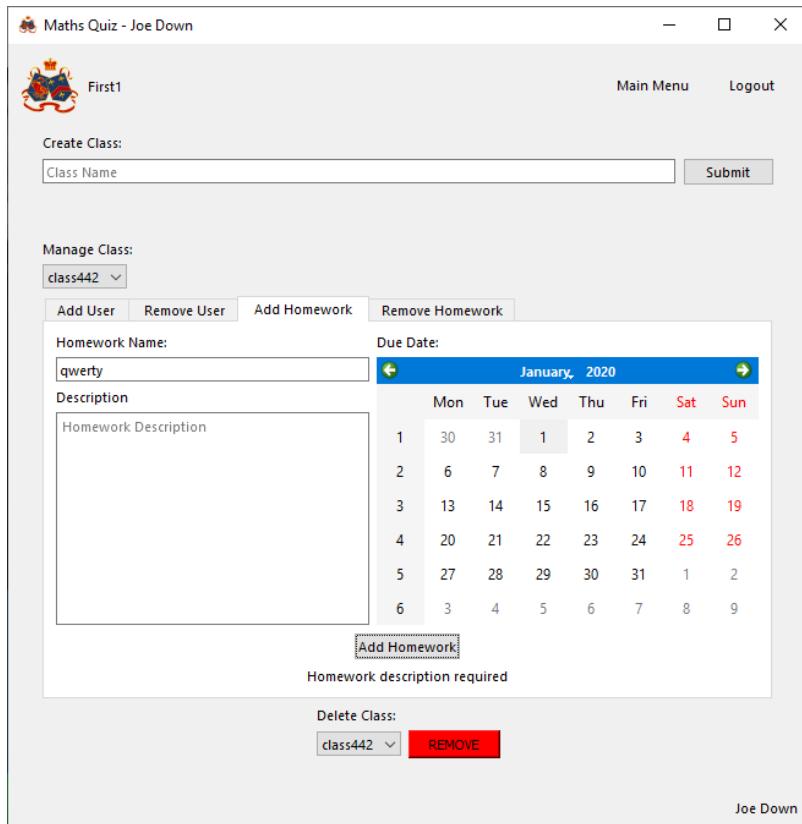
Add User Remove User Add Homework Remove Homework

Homework Name:	Due Date:
<input type="text" value="qwerty"/>	 January, 2020 
Description	Mon Tue Wed Thu Fri Sat Sun
Homework Description	1 30 31 1 2 3 4 5
	2 6 7 8 9 10 11 12
	3 13 14 15 16 17 18 19
	4 20 21 22 23 24 25 26
	5 27 28 29 30 31 1 2
	6 3 4 5 6 7 8 9

Delete Class:

class442 REMOVE

Joe Down



Successfully informs user that a description is required

4.6.3.13.5.2.2.3 Due date in the past

Dates in the past cannot be selected

4.6.3.13.5.3 Evaluation

Successfully validates all inputs and then creates a new homework in the selected class using the user inputted parameters

4.6.3.13.6 Function to delete homework

4.6.3.13.6.1 Final Code

```
def admin_remove_homework(self):
    # If homework selected runs functions to remove selected homework and outputs success message
    if len(self.homework) > 0:
        ui_scripts.remove_homework(self.homework[self.admin_remove_homework_combo_box.currentIndex()][0])
        self.admin_update_remove_homework_combo_box()
        self.admin_remove_homework_status_label.setText("Homework removed")
    # If no homework selected, outputs error and makes no changes
    else:
        self.admin_remove_homework_status_label.setText("Class has no homework")
```

4.6.3.13.6.2 Testing

4.6.3.13.6.2.1 With valid data

Maths Quiz - Joe Down

First1

Main Menu Logout

Create Class:

Class Name Submit

Manage Class:

class442

Add User Remove User Add Homework Remove Homework

qwerty REMOVE

Delete Class:

class442 REMOVE

Joe Down

Maths Quiz - Joe Down

First1

Main Menu Logout

Create Class:

Class Name Submit

Manage Class:

class442

Add User Remove User Add Homework Remove Homework

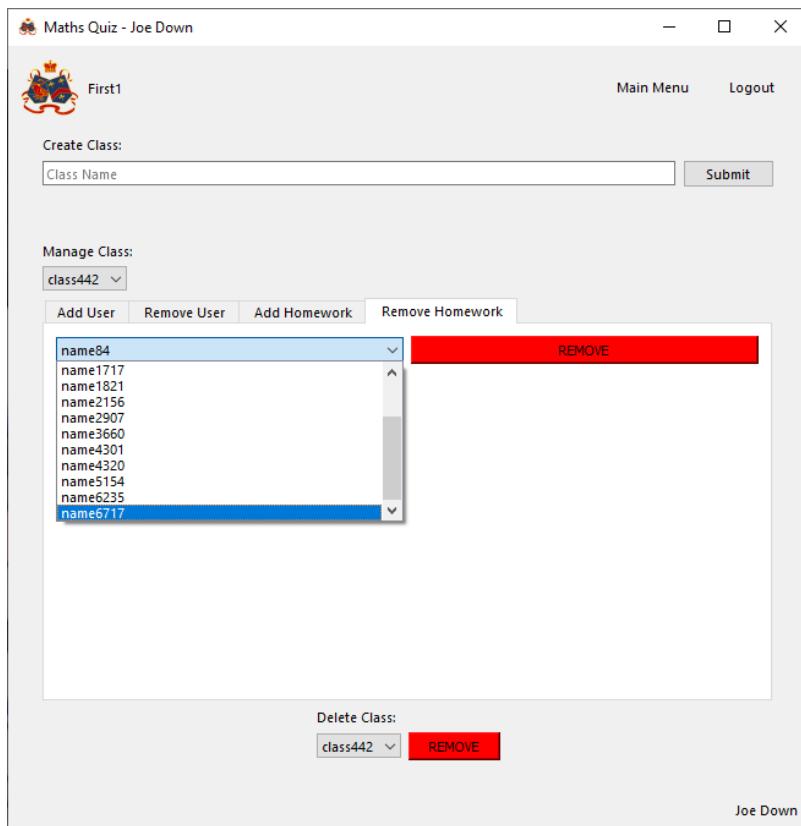
name84 REMOVE

Homework removed

Delete Class:

class442 REMOVE

Joe Down



Successfully removes selected homework from class

4.6.3.13.6.2.2 With invalid data

N/A, nature of dropdown menu means invalid removal cannot occur

4.6.3.13.6.3 Evaluation

Successfully removes the relationship between the selected class and homework

4.6.3.14 Account management page

4.6.3.14.1 Function to update account details

4.6.3.14.1.1 Final Code

```
def account_management_detail_update(self):
    # User details will only update if current password is correct
    if db_scripts.check_password(self.current_user, self.account_management_old_password_input.text()):
        # If a potential new password is entered, runs scripts to validate
        if (self.account_management_new_password_input.text() != ''
            or self.account_management_new_password_verify_input.text() != ''):
            # If new password and new password verification match goes to scripts
            to validate new password
            if (self.account_management_new_password_input.text()
                == self.account_management_new_password_verify_input.text()):
                # If new password is invalid outputs error
                if len(self.account_management_new_password_input.text()) < 8:
                    self.account_management_reset_page()
                    self.account_management_success_output.setText(
                        "Invalid password (Must be at least 8 characters)")
                # If new password entered is valid updates user data
            else:
                # Updates first and last names using function
                self.account_management_update_first_and_last_names()
                # Generates new hashed password and stores
                db_scripts.update_password(self.current_user,
                                           self.account_management_new_password_input.text())
                # Outputs success
                self.account_management_reset_page()
                self.account_management_success_output.setText("Success")
            # If new password and verification mismatch outputs error
        else:
            self.account_management_reset_page()
            self.account_management_success_output.setText("New password fields
must match")
        # If no new password is entered runs scripts to update first and last name
    else:
        first_or_last_updated: bool = self.account_management_up-
date_first_and_last_names()
        self.account_management_reset_page()
        # If any data was updated outputs a success message
        if first_or_last_updated:
            self.account_management_success_output.setText("Success")
        # If no changes were made no output
        else:
            self.account_management_success_output.setText("")
    # If current password incorrect outputs and error
else:
    self.account_management_reset_page()
    self.account_management_success_output.setText("Correct current password
required to change user data")
```

4.6.3.14.1.2 Testing

4.6.3.14.1.2.1 With valid data

4.6.3.14.1.2.1.1 No new first name

Maths Quiz - Joe Down

First1

Main Menu Logout

New First Name New Last Name

First Name Last Name

Current Password:

New Password

New Password

Verify New Password:

Verify New Password

Submit

Joe Down

Maths Quiz - Joe Down

First1

Logout

Class Management

Homework Management

Scores

Account Management

Joe Down

Successfully leaves first name unchanged

4.6.3.14.1.2.1.2 No new last name

Maths Quiz - Joe Down

First1

Main Menu Logout

New First Name New Last Name

First Name Last Name

Current Password:

New Password

Verify New Password:

Verify New Password

Submit

Joe Down

Maths Quiz - Joe Down

First1

Main Menu Logout

View Type:

Class View

Class:

test

Homework:

	First Name	Last Name	Score	Percentage	Attempts
1	first1	last1	0	N/A	0

Joe Down

Successfully leaves last name unchanged

4.6.3.14.1.2.1.3 No new password

Maths Quiz - Joe Down

First1

Main Menu Logout

New First Name New Last Name
 First Name Last Name

Current Password:

New Password
 New Password

Verify New Password:
 Verify New Password

Joe Down

Maths Quiz - Joe Down

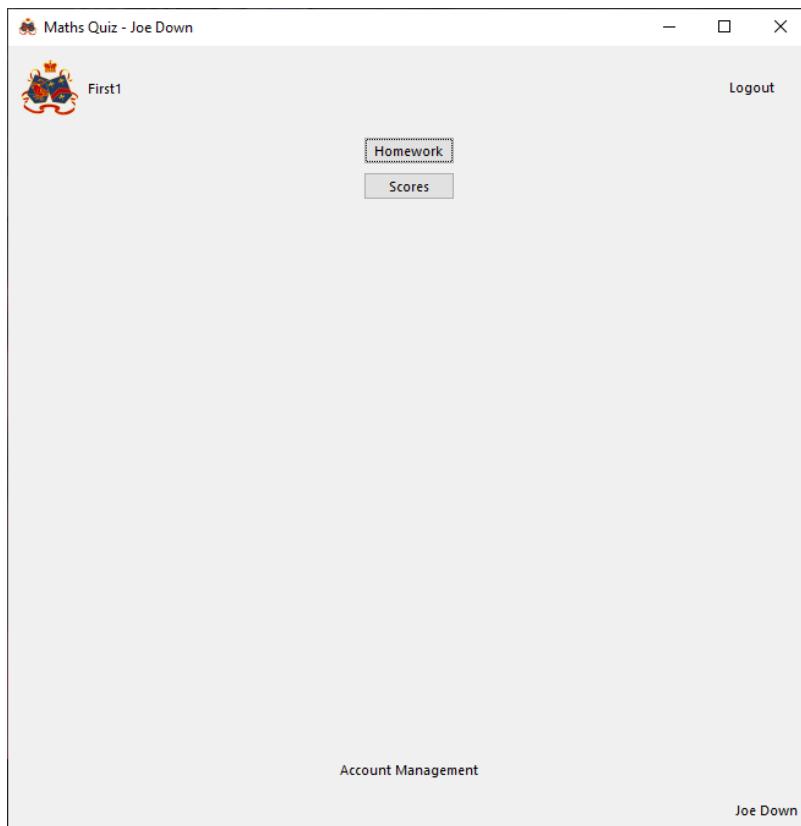
student1

Username Password
 student1 *********

Logout successful

Create Account

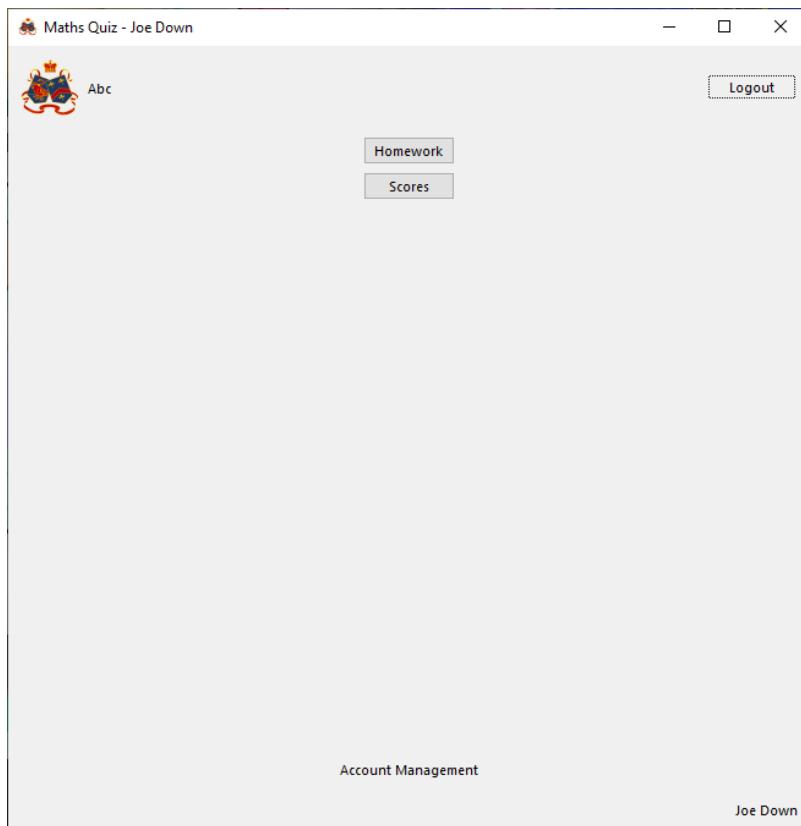
Joe Down



Successfully leaves password unchanged

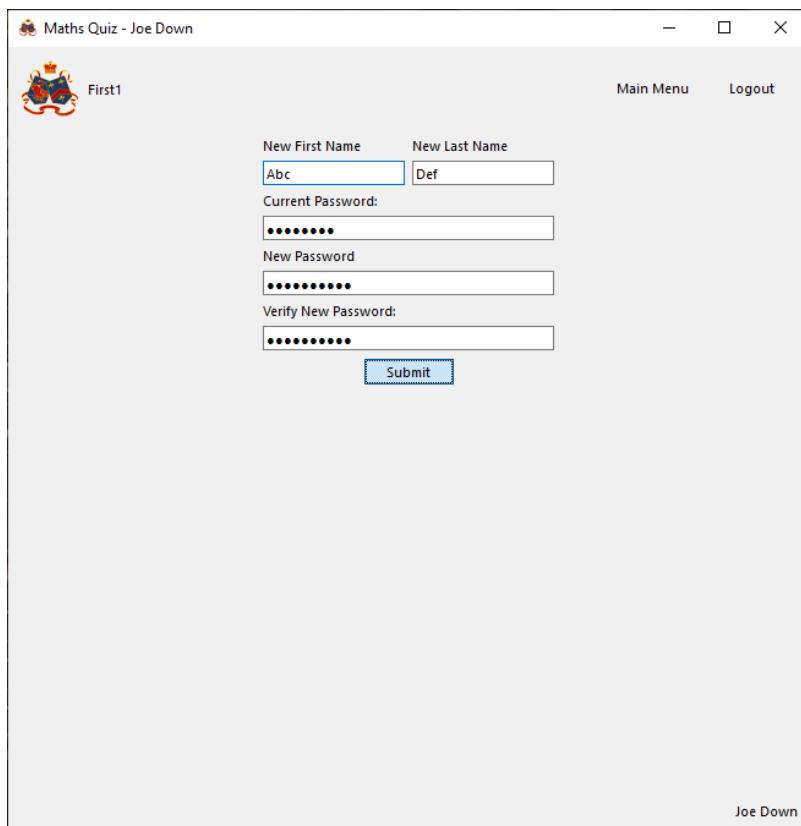
4.6.3.14.1.2.1.4 New first name

The screenshot shows the "Account Management" form. At the top, it says "New First Name" and "New Last Name". Below these are two input fields: the first contains "Abc" and the second contains "Def". Underneath these fields is a label "Current Password:" followed by a field containing six dots ("....."). Below that is a label "New Password:" followed by a field containing nine dots ("....."). Underneath that is a label "Verify New Password:" followed by a field containing nine dots ("....."). At the bottom of the form is a blue "Submit" button. The user's name "Joe Down" is visible at the bottom right of the window.



Successfully updates first name

4.6.3.14.1.2.1.5 New last name



Maths Quiz - Joe Down

First1

Main Menu Logout

View Type: Class View

Class: test

Homework:

	First Name	Last Name	Score	Percentage	Attempts
1	abc	def	0	N/A	0

Joe Down

Successfully updates last name

4.6.3.14.1.2.1.6 New password

Maths Quiz - Joe Down

First1

Main Menu Logout

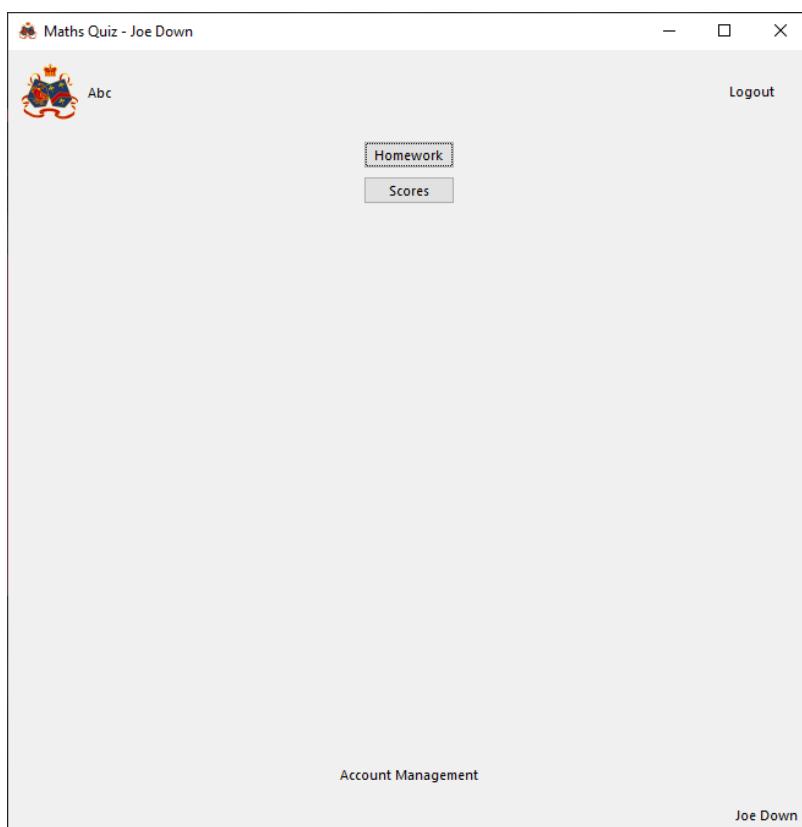
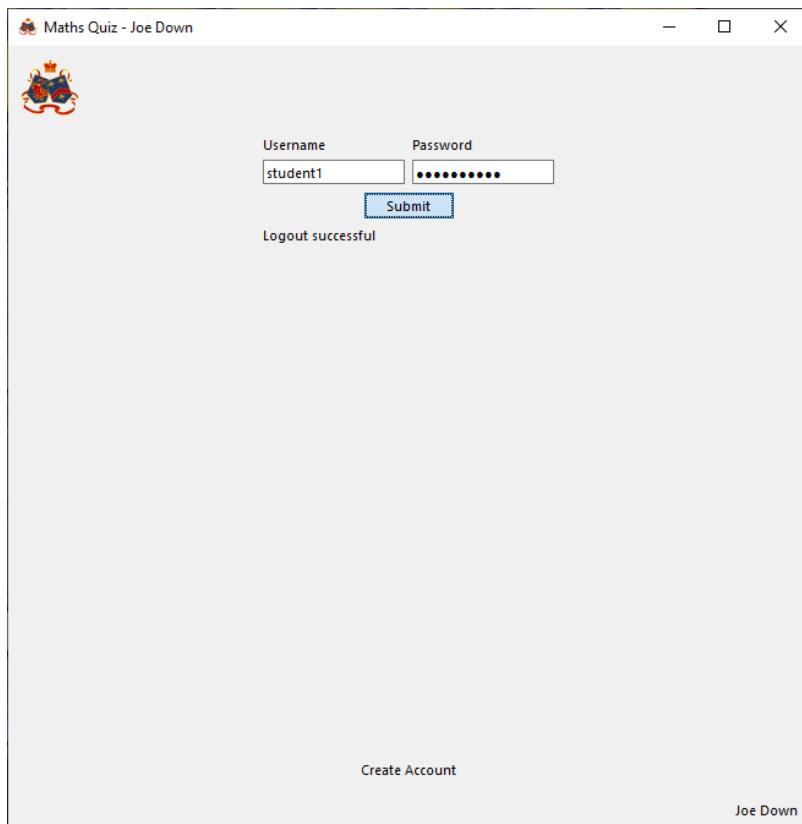
New First Name: Abc New Last Name: Def

Current Password:

New Password:

Verify New Password:

Joe Down



Successfully updates password

4.6.3.14.1.2.2 With invalid data

4.6.3.14.1.2.2.1 Current password incorrect

The screenshot shows a Windows application window titled "Maths Quiz - Joe Down". The window contains a user profile section with the name "First1" and a profile picture. Below the profile are input fields for "New First Name" (containing "Abc") and "New Last Name" (containing "Def"). There are also fields for "Current Password" (containing "*****"), "New Password", and "Verify New Password". A "Submit" button is at the bottom. The status bar at the bottom of the window displays "Joe Down".

This screenshot shows the same application window after a submission attempt. The "Current Password" field now contains "Current Password" instead of the previous value. A new error message "Correct current password required to change user data" is displayed below the "Submit" button. The status bar still shows "Joe Down".

Successfully refuses to apply any changes and informs user password is incorrect

4.6.3.14.1.2.2.2 Mismatching new password verification

Maths Quiz - Joe Down

First1 Main Menu Logout

New First Name New Last Name
 First Name Last Name

Current Password:
 ••••••••••

New Password
 ••••••

Verify New Password:
 ••••••

Joe Down

Maths Quiz - Joe Down

First1 Main Menu Logout

New First Name New Last Name
 First Name Last Name

Current Password:
 Current Password

New Password
 New Password

Verify New Password:
 Verify New Password

New password fields must match

Joe Down

Successfully informs user new passwords do not match and does not update password

4.6.3.14.1.2.2.3 New password invalid

Maths Quiz - Joe Down

First1 Main Menu Logout

New First Name New Last Name
 First Name Last Name

Current Password:
 #####

New Password
 #####

Verify New Password:
 #####

Joe Down

Maths Quiz - Joe Down

First1 Main Menu Logout

New First Name New Last Name
 First Name Last Name

Current Password:
 Current Password

New Password
 New Password

Verify New Password:
 Verify New Password

Invalid password (Must be at least 8 characters)

Joe Down

Successfully informs user password is invalid

4.6.3.14.1.3 Evaluation

Successfully validates that the password given is correct and if so, validates all inputs and updates the user's details using the given parameters (only if given)

4.6.3.15 View classes page

4.6.3.15.1 Function to update results table

4.6.3.15.1.1 Final Code

```
def view_classes_update_table(self):
    # Resets table to blank state
    self.view_classes_score_table.clear()
    # Gets the current class id
    current_class: int = self.current_classes[self.view_classes_class_combo_box.currentIndex()][0]
    # Gets the id of the selected homework or student
    current_student_or_homework: int = self.view_classes_students_or_homework[
        self.view_classes_homework_or_student_combo_box.currentIndex()]
    # If view homework scores selected:
    if self.view_classes_view_type_combo_box.currentIndex() == 0:
        # Sets required column headers
        self.view_classes_score_table.setColumnCount(5)
        self.view_classes_score_table.setHorizontalHeaderLabels([
            "First Name", "Last Name", "Score", "Percentage", "Attempts"])
        # Gets all scores from database (see function for more detail)
        scores: list = ui_scripts.get_results_of_homework(current_class, current_student_or_homework)
        self.view_classes_score_table.setRowCount(len(scores))
        # Inserts each set of score data into the database
        if len(scores) != 0:
            row_counter: int = -1
            for each_score in scores:
                row_counter += 1
                self.view_classes_score_table.setItem(row_counter, 0,
QtWidgets.QTableWidgetItem(each_score[0]))
                self.view_classes_score_table.setItem(row_counter, 1,
QtWidgets.QTableWidgetItem(each_score[1]))
                self.view_classes_score_table.setItem(row_counter, 2,
QtWidgets.QTableWidgetItem(str(each_score[2])))
                self.view_classes_score_table.setItem(row_counter, 3,
QtWidgets.QTableWidgetItem(str(each_score[3])))
                self.view_classes_score_table.setItem(row_counter, 4,
QtWidgets.QTableWidgetItem(str(each_score[4])))
            # If view student scores selected:
        else:
            # Sets required column headers
            self.view_classes_score_table.setColumnCount(4)
            self.view_classes_score_table.setHorizontalHeaderLabels(["Homework",
"Score", "Percentage", "Attempts"])
            # Gets all scores from database (see function for more detail)
            scores: list = ui_scripts.get_scores_of_student_in_class(current_class,
current_student_or_homework)
            self.view_classes_score_table.setRowCount(len(scores))
            # Inserts each set of score data into the database
            if len(scores) != 0:
                row_counter: int = -1
                for each_score in scores:
                    row_counter += 1
                    self.view_classes_score_table.setItem(row_counter, 0,
QtWidgets.QTableWidgetItem(each_score[0]))
                    self.view_classes_score_table.setItem(row_counter, 1,
QtWidgets.QTableWidgetItem(str(each_score[1])))
                    self.view_classes_score_table.setItem(row_counter, 2,
QtWidgets.QTableWidgetItem(str(each_score[2])))
                    self.view_classes_score_table.setItem(row_counter, 3,
QtWidgets.QTableWidgetItem(str(each_score[3])))
```

4.6.3.15.1.2 Testing

4.6.3.15.1.2.1 With valid data

4.6.3.15.1.2.1.1 Class view

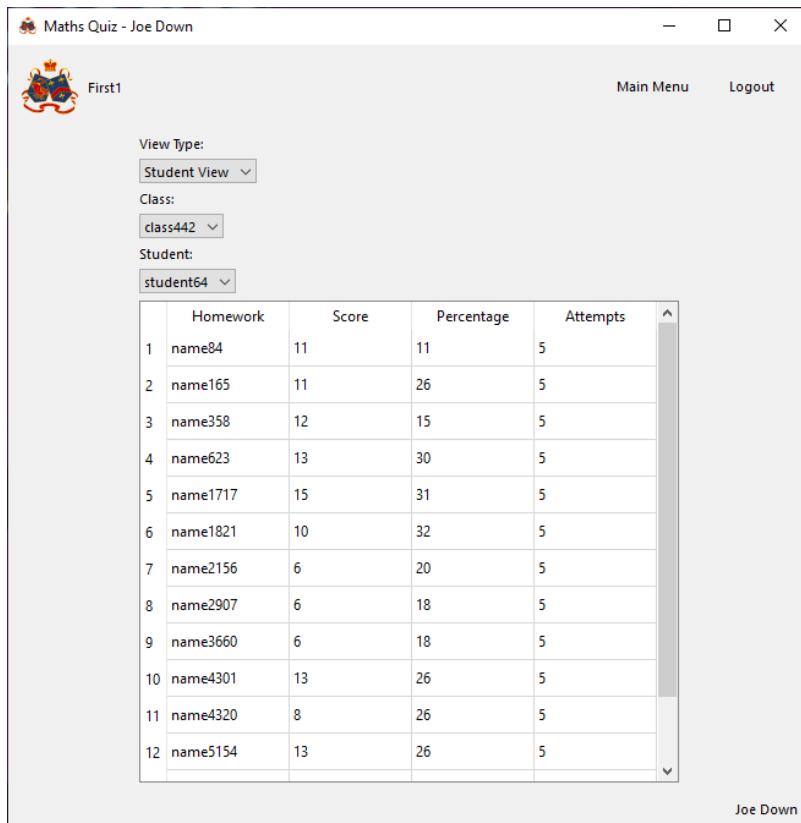
The screenshot shows a window titled "Maths Quiz - Joe Down". At the top left is a logo of a person with a crown. The title bar also displays "First1". On the right side of the title bar are "Main Menu" and "Logout" buttons. Below the title bar, there are three dropdown menus: "View Type" (set to "Class View"), "Class" (set to "class442"), and "Homework" (set to "name84"). The main area contains a table with 12 rows of student data. The columns are labeled "First Name", "Last Name", "Score", "Percentage", and "Attempts". The data is as follows:

	First Name	Last Name	Score	Percentage	Attempts
1	first64	last64	11	11	5
2	first64	last64	11	11	5
3	first55	last55	6	24	4
4	first6	last6	5	15	5
5	first71	last71	2	6	4
6	first17	last17	12	23	5
7	first60	last60	6	13	5
8	first37	last37	6	19	5
9	first56	last56	6	23	5
10	first72	last72	6	29	4
11	first17	last17	12	23	5
12	first27	last27	7	15	5

At the bottom center of the window, it says "Joe Down".

Successfully calculates and outputs all required score data for selected class and homework

4.6.3.15.1.2.1.2 Student view



The screenshot shows a Windows application window titled "Maths Quiz - Joe Down". In the top left corner is a logo of a cartoon character. The title bar also displays "First1". On the right side of the title bar are "Main Menu" and "Logout" buttons. The main content area has a header "View Type:" followed by a dropdown menu set to "Student View". Below this are three more dropdown menus: "Class:" set to "class442", and "Student:" set to "student64". The central part of the window contains a table with 12 rows of data. The columns are labeled "Homework", "Score", "Percentage", and "Attempts". The data is as follows:

	Homework	Score	Percentage	Attempts
1	name84	11	11	5
2	name165	11	26	5
3	name358	12	15	5
4	name623	13	30	5
5	name1717	15	31	5
6	name1821	10	32	5
7	name2156	6	20	5
8	name2907	6	18	5
9	name3660	6	18	5
10	name4301	13	26	5
11	name4320	8	26	5
12	name5154	13	26	5

In the bottom right corner of the window, the name "Joe Down" is displayed.

Successfully calculates and outputs all required score data for selected class and student

4.6.3.15.1.2.2 With invalid data

N/A, nature of dropdown menu means invalid selections cannot occur

4.6.3.15.1.3 Evaluation

Successfully takes all the data for either students in a class or homework of a student and displays it in a table, with what is displayed correctly matching the selected view type

4.6.3.16 Homework select page

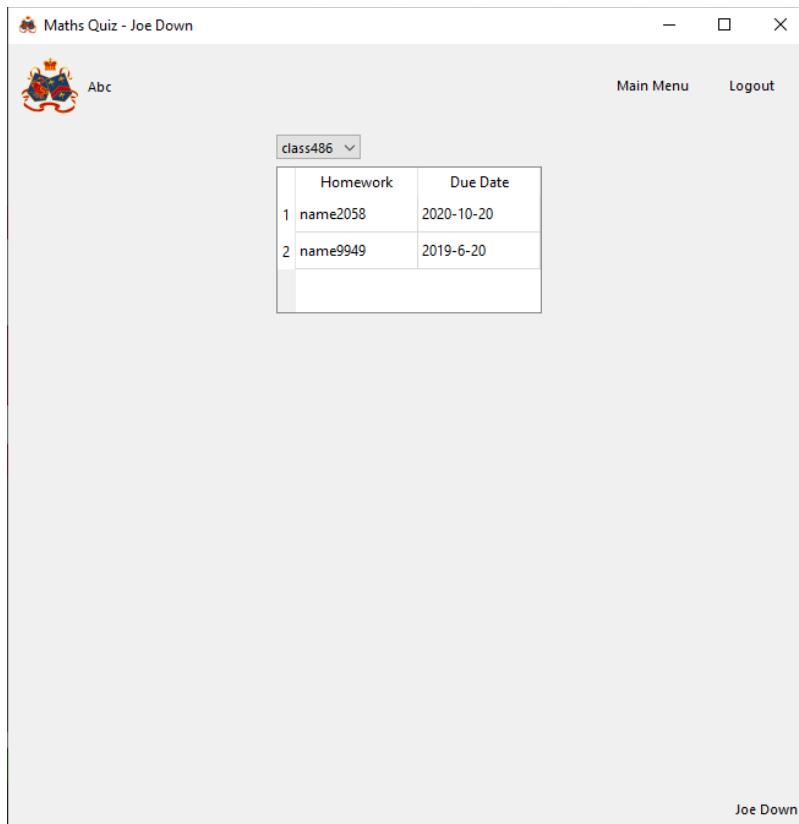
4.6.3.16.1 Function to update homework list

4.6.3.16.1.1 Final Code

```
def homework_select_update_table(self):
    # Resets and clears table
    self.homework_select_table.clearContents()
    # Unless there is no class selected:
    if len(self.current_classes) != 0:
        # Gets all homework ids for given class and stores them
        homework_ids: list = []
        self.homework_ids: list = []
        for each_homework in ui_scripts.get_homework_of_class(
            self.current_classes[self.homework_select_class_combo_box.cur-
rentIndex()][0]):
            homework_ids.append(each_homework[0])
        # Gets homework data for each homework id, converts date stored to datetime
        and compares to current date. If
            # due date has not passed, appends homework data to homework list
            for each_homework in homework_ids:
                data: tuple = ui_scripts.get_homework_name_and_due_date(each_homework,
self.current_classes[
                    self.homework_select_class_combo_box.currentIndex()][0])
                due_year_month_day: list = data[1].split('-')
                due_date: datetime.date = datetime.date(int(due_year_month_day[0]),
int(due_year_month_day[1]),
int(due_year_month_day[2]))
                if datetime.date.today() < due_date:
                    self.homework_ids.append([each_homework, data[0], data[1]])
    # Sets up table and inserts all stored homework data with valid due date
    self.homework_select_table.setRowCount(len(self.homework_ids))
    row_counter: int = -1
    for each_homework in self.homework_ids:
        row_counter += 1
        self.homework_select_table.setItem(row_counter, 0, QtWidgets.QTable-
WidgetItem(each_homework[1]))
        self.homework_select_table.setItem(row_counter, 1, QtWidgets.QTable-
WidgetItem(each_homework[2]))
```

4.6.3.16.1.2 Testing

4.6.3.16.1.2.1 With valid data



Successfully loads all homework for selected class with future due date

4.6.3.16.1.2.2 With invalid data

N/A, no user inputs

4.6.3.16.1.3 Evaluation

Successfully collects data relating to each homework and outputs each homework in the table for which the due date has not yet passed

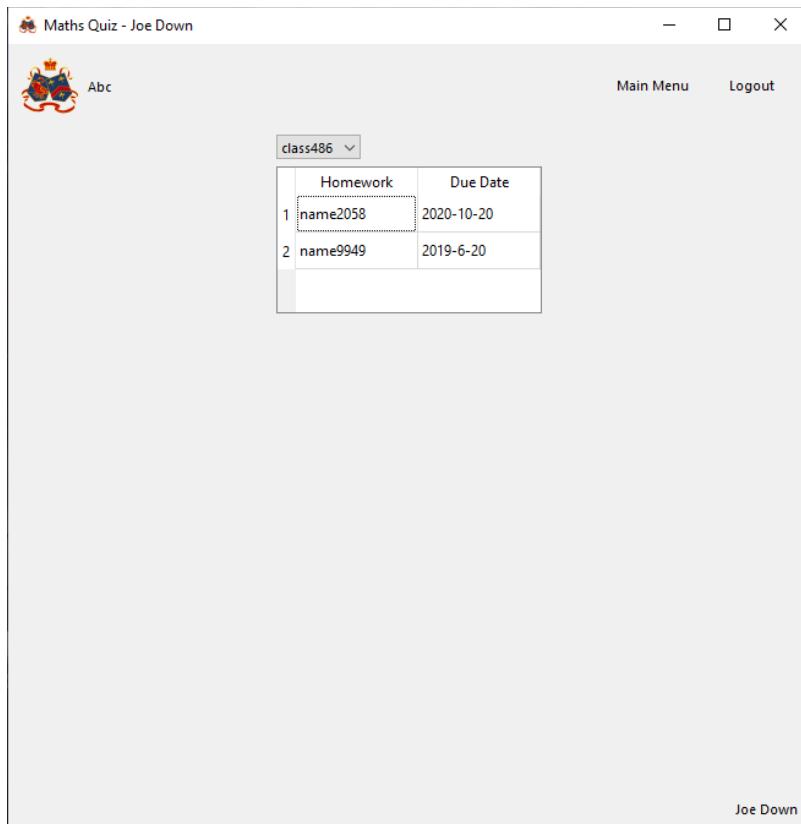
4.6.3.16.2 Function to load homework

4.6.3.16.2.1 Final Code

```
def homework_select_table_clicked(self):  
    # Gets the id of the selected homework  
    homework_id = self.homework_ids[self.homework_select_table.currentRow()][0]  
    # Stores all questions for the selected homework in a list  
    self.questions: list = ui_scripts.get_questions_of_homework(homework_id)  
    # If homework contains questions, goes to question page  
    if len(self.questions) > 0:  
        self.change_page(self.page_dictionary['question_page'])  
    # If homework is empty does nothing  
    else:  
        return
```

4.6.3.16.2.2 Testing

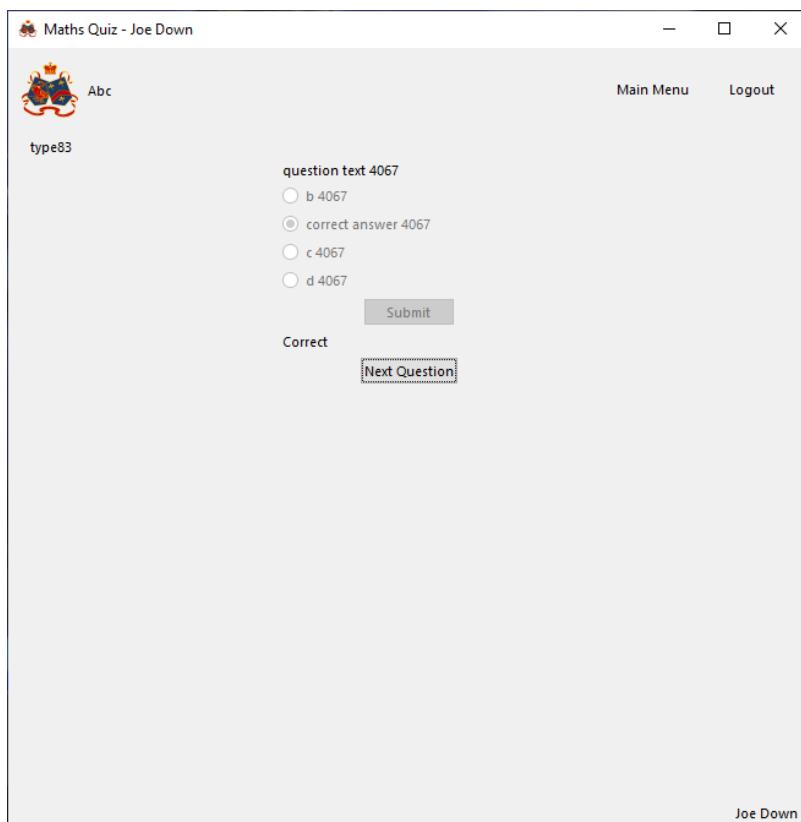
4.6.3.16.2.2.1 With valid data



The screenshot shows a window titled "Maths Quiz - Joe Down". At the top left is a cartoon character icon and the text "Abc". On the right are "Main Menu" and "Logout" buttons. Below the title bar is a dropdown menu showing "class486". A table displays two rows of homework data:

Homework	Due Date
1 name2058	2020-10-20
2 name9949	2019-6-20

At the bottom right of the window is the name "Joe Down".



The screenshot shows a window titled "Maths Quiz - Joe Down". At the top left is a cartoon character icon and the text "Abc". On the right are "Main Menu" and "Logout" buttons. Below the title bar is the text "type83". A question is displayed with the text "question text 4067" and four options:

- b 4067
- correct answer 4067
- c 4067
- d 4067

A "Submit" button is below the options. The text "Correct" is displayed above a "Next Question" button.

At the bottom right of the window is the name "Joe Down".

Successfully loads selected homework

4.6.3.16.2.2.2 With invalid data

N/A, no invalid selections possible due to nature of table

4.6.3.16.2.3 Evaluation

Successfully loads the first question in a matching homework when a homework in the table is selected unless there are no questions (in which case no attempt is made to load)

4.7 SERIES SCRIPTS (MATHS_SCRIPTS.SERIES)

4.7.1 Taylor Series Class

For further explanation of how a Taylor series works, see design section

4.7.1.1 Function to initialise class

4.7.1.1.1 Issues during development

Initially, the below code was used:

```
# Class to generate a taylor series and approximate values for a given function and
centre
class Taylor:
    def __init__(self, function: sympy.add.Add, centre: int):
        # Sets up x as a variable in SymPy
        x = sympy.symbols('x')
        # Defines constants containing the function to be expanded and the centre
        self.function = function
        self.centre = centre
```

This resulted in errors later when developing code to evaluate functions as it meant invalid centres of expansion could be used, resulting in any value substituted into the function being approximated as equal to infinity:

```
if __name__ == '__main__':
    x = sympy.symbols('x')
    f = Taylor(sympy.log(x), 0)
    print(f.evaluate(10, 100))
```

```
C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe
C:/Users/Joe/PycharmProjects/Coursework/maths_scripts/series.py
```

```
nan
```

Process finished with exit code 0

While the function and centre are always to be verified before usage in the main program, in case of incorrect usage, it was decided to include an exception which ends the program if an incorrect centre is entered to prevent an incorrect question result being generated later

```
# Outputs an error if a function is entered with an invalid centre (function is undefined at centre)
if function.subs(x, self.centre) == sympy.zoo:
    raise Exception("Invalid centre")
```

4.7.1.1.2 Final Code

```
# Class to generate a taylor series and approximate values for a given function and
# centre
class Taylor:
    def __init__(self, function: sympy.add.Add, centre: int):
        # Sets up x as a variable in SymPy
        x = sympy.symbols('x')
        # Defines constants containing the function to be expanded and the centre
        self.function = function
        self.centre = centre
        # Outputs an error if a function is entered with an invalid centre (function
        # is undefined at centre)
        if function.subs(x, self.centre) == sympy.zoo:
            raise Exception("Invalid centre")
```

4.7.1.1.3 Testing

4.7.1.1.3.1 With valid data:

e^0 is valid (equal to 1), so expansion is valid as $f(0)$ is defined:

```
if __name__ == '__main__':
    x = sympy.symbols('x')
    f = Taylor(sympy.exp(x), 0)
```

```
C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe
C:/Users/Joe/PycharmProjects/Coursework/math_scripts/series.py
```

Process finished with exit code 0

No error occurs

4.7.1.1.3.2 With Invalid data:

$\log(0)$ is undefined for a log of any base, so expanding the below function about $x=0$ should result in an error as $f(0)$ is undefined:

```
if __name__ == '__main__':
    x = sympy.symbols('x')
    f = Taylor(sympy.log(x), 0)
```

Traceback (most recent call last):

```
File "C:/Users/Joe/PycharmProjects/Coursework/math_scripts/series.py", line 40, in <module>
    f = Taylor(sympy.log(x), 0)

File "C:/Users/Joe/PycharmProjects/Coursework/math_scripts/series.py", line 15, in __init__
    raise Exception("Invalid centre")
```

Exception: Invalid centre

Process finished with exit code 1

Correctly returns an error if an expansion about an invalid centre is attempted (This should really be verified before calling function and should not need to occur in practise)

4.7.1.1.3.3 Evaluation

Correctly initialises class and verifies valid centre is used

4.7.1.2 Function to generate an expansion with a given number of terms

4.7.1.2.1 Final Code

```
def equation_in_x(self, precision: int) -> sympy.add.Add:
    # Sets up x as a variable in SymPy
    x = sympy.symbols('x')
    # Initialises variable to hold final function as nothing
    final_equation: sympy.add.Add = sympy.add.Add(0)
    # Initialises variable to hold the next derivative to be used during expansion
    # as the value of the initial
    # function
    function: sympy.add.Add = self.function
    # Loops through for the number of terms entered as the precision and generates
    # next term in the Taylor series
    # on each loop
    for n in range(0, precision):
        # Calculates the value of the current function at the centre of expansion
        f: float = function.subs(x, self.centre)
        # Uses formula (see design section) to generate the next term in x in the
        # Taylor series and add to end of
        # current full expansion
        final_equation += (f / maths_scripts.integer_factorial(n)) * ((x -
self.centre) ** n)
        # Replaces current function with the derivative of the current function
        function = sympy.diff(function, x)
    # Returns full expansion up to number of terms specified
    return final_equation
```

4.7.1.2.2 Testing

4.7.1.2.2.1 With valid data:

4.7.1.2.2.1.1 e^x

Correct expansion with 4 terms:

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} \dots = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

Program's expansion:

```
if __name__ == '__main__':
    x = sympy.symbols('x')
    f = Taylor(sympy.exp(x), 0)
    print(f.equation_in_x(4))
```

```
C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe
C:/Users/Joe/PycharmProjects/Coursework/maths_scripts/series.py
```

```
x**3/6 + x**2/2 + x + 1
```

Process finished with exit code 0

Expansions successfully match

4.7.1.2.2.1.2 $1/(1-x)$

Correct expansion with 4 terms:

$$\frac{1}{1-x} = 1 + x + x^2 + x^3 \dots = \sum_{n=0}^{\infty} x^n$$

Program's expansion:

```
if __name__ == '__main__':
    x = sympy.symbols('x')
    f = Taylor(sympy.exp(x), 0)
    print(f.equation_in_x(4))
```

C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe
C:/Users/Joe/PycharmProjects/Coursework/maths_scripts/series.py

$x^{**3} + x^{**2} + x + 1$

Process finished with exit code 0

Expansions successfully match

4.7.1.2.2.1.3 $\ln(1+x)$

Correct expansion with 4 terms:

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \dots = \sum_{n=0}^{\infty} \frac{(-1)^n x^{n+1}}{n+1}$$

CalculusWorkshop.com

Program's expansion:

```
if __name__ == '__main__':
    x = sympy.symbols('x')
    f = Taylor(sympy.ln(1+x), 0)
    print(f.equation_in_x(5))
```

C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe
C:/Users/Joe/PycharmProjects/Coursework/maths_scripts/series.py

$-x^{**4}/4 + x^{**3}/3 - x^{**2}/2 + x$

Process finished with exit code 0

Expansions successfully match

4.7.1.2.2.2 *With Invalid data:*

Function and centre are validated during class declaration, before equation can be generated, and so invalid expansion cannot occur

4.7.1.2.2.3 *Evaluation*

Correctly generates the correct Taylor series for a given function and centre of expansion

4.7.1.3 Function to evaluate a function at a given value from its Taylor series

4.7.1.3.1 Final Code

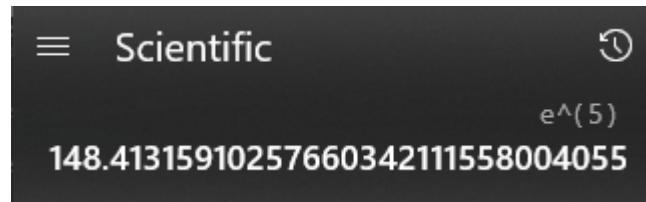
```
def evaluate(self, value: float, precision: int) -> float:  
    # Sets up x as a variable in SymPy  
    x = sympy.symbols('x')  
    # Generates Taylor series to given precision and substitutes value of x with  
    given value to evaluate function  
    function = self.equation_in_x(precision)  
    # Returns final value  
    return function.subs(x, value)
```

4.7.1.3.2 Testing

4.7.1.3.2.1 With valid data:

4.7.1.3.2.1.1 e^x

Correct value of e^5 :



The calculator screen shows the result of calculating e^5 . The display is set to Scientific mode, indicated by the 'Scientific' button. The result is shown as $e^{(5)}$ followed by the numerical value $148.41315910257660342111558004055$.

Program's value (using 500 terms in the expansion):

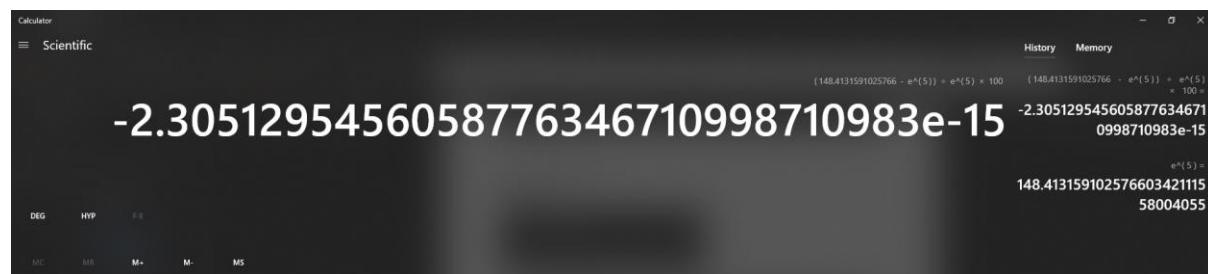
```
if __name__ == '__main__':  
    x = sympy.symbols('x')  
    f = Taylor(sympy.exp(x), 0)  
    print(float(f.evaluate(5, 500)))
```

```
C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe  
C:/Users/Joe/PycharmProjects/Coursework/maths_scripts/series.py
```

148.4131591025766

Process finished with exit code 0

Percentage difference compared to true value:

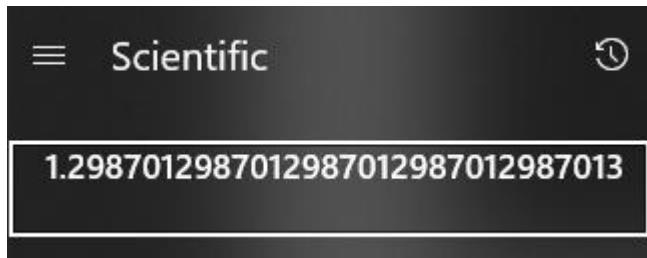


The calculator screen shows the percentage difference between the calculated value and the true value of e^5 . The display shows the formula $(148.4131591025766 - e^5) / e^5 \times 100$, the result $-2.3051295456058776346710998710983e-15$, and the true value $148.41315910257660342111558004055$.

In this case, the generated value is an underestimate within 0.000000000000003 % of the true value (approximation is therefore very accurate)

4.7.1.3.2.1.2 $1/(1-x)$

Correct value of $1 / (1 - 0.23)$:



Program's value (using 500 terms in the expansion):

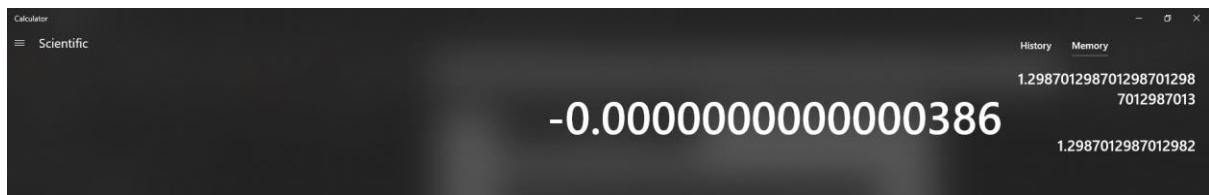
```
if __name__ == '__main__':
    x = sympy.symbols('x')
    f = Taylor(1/(1-x), 0)
    print(float(f.evaluate(0.23, 500)))
```

```
C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe
C:/Users/Joe/PycharmProjects/Coursework/maths_scripts/series.py
```

1.2987012987012982

Process finished with exit code 0

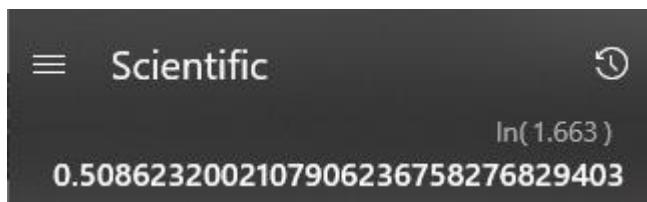
Percentage difference compared to true value:



In this case, the generated value is an underestimate within 0.0000000000004% of the true value (approximation is therefore very accurate)

4.7.1.3.2.1.3 $\ln(1+x)$

Correct value of $\ln(1+0.663)$:



Program's value (using 500 terms in the expansion):

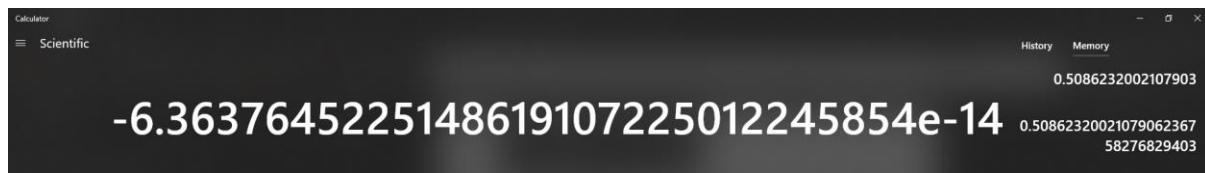
```
if __name__ == '__main__':
    x = sympy.symbols('x')
    f = Taylor(sympy.ln(1 + x), 0)
    print(float(f.evaluate(0.663, 500)))
```

```
C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe  
C:/Users/Joe/PycharmProjects/Coursework/maths_scripts/series.py
```

0.5086232002107903

Process finished with exit code 0

Percentage difference compared to true value:



In this case, the generated value is an underestimate within 0.00000000000007% of the true value (approximation is therefore very accurate)

4.7.1.3.2.2 With Invalid data:

In theory, no values are invalid for usage in the Taylor series, however accuracy will decrease as values move away from the centre of expansion. This can however be addressed by selecting a different centre of expansion when defining the initial Taylor series to one closer to the value to be checked. Validation should therefore be carried out before using function.

4.7.1.3.2.3 Evaluation

Correctly computes value approximations of function to a very high degree of accuracy for values close to the centre of expansion

4.7.2 Maclaurin Series Class

4.7.2.1 Final code

```
class Maclaurin(Taylor):  
    def __init__(self, function: sympy.add.Add):  
        # Inherits all code from Taylor series class while as Maclaurin series is  
        # simply a special case of it with  
        # centre of expansion 0. Centre of expansion is therefore not required to  
        # be passed when declaring instance of  
        # class as it can always be passed to parent class as 0  
        super().__init__(function, 0)
```

4.7.2.2 Testing

Testing not required as Maclaurin series is simply a special case of the already tested Taylor series

4.7.2.3 Evaluation

Maclaurin series works as intended as a special case of the Taylor series

4.7.3 Sine function

4.7.3.1 Issues during development

Initially the following code was implemented:

```

def sin(value: float) -> float:
    # Defines the precision of the function
    precision = 20
    # Sets up x as a variable in SymPy
    x = sympy.symbols('x')
    # Calculates the Maclaurin series for sin x to the required level in a similar
    way to the previously developed
    # function
    final_equation: sympy.add.Add = sympy.add.Add(0)
    for n in range(0, precision):
        final_equation += (-1) ** n / integer_factorial(2 * n + 1) * (x ** (2 * n +
1))
    # Substitutes given value of x and returns the approximate value of sin x
    return final_equation.subs(x, value)

```

This would result in very inaccurate results being calculated for values of x greatly outside the region $0 \leq x < 2\pi$ (as x increases, generated value rapidly tends towards infinity when by the definition of a sine function they should not ever exceed 1 or fall below -1):

```

if __name__ == '__main__':
    print("x=", 999, "Sin(x)=", sin(999))
    print("x=", -999, "Sin(x)=", sin(-999))

```

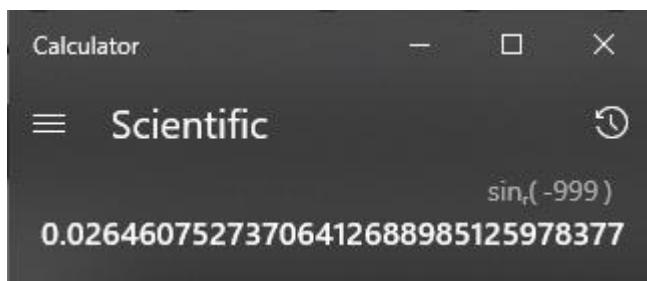
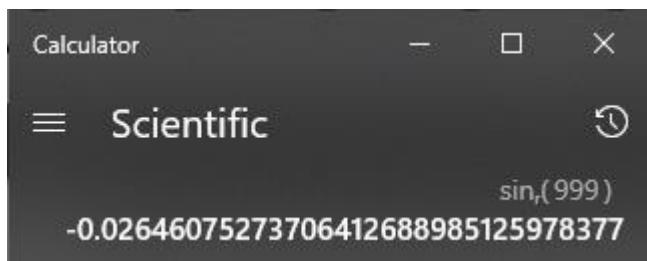
```

C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe
C:/Users/Joe/PycharmProjects/Coursework/maths_scripts/__init__.py

x= 999 Sin(x)= -4.70786967532324e+70
x= -999 Sin(x)= 4.70786967532324e+70

```

Process finished with exit code 0



This is a limitation of the Maclaurin expansion as it requires more and more terms to accurately approximate a function further and further from the centre of expansion. As sine is a repeating function at intervals of 2π however, 2π can simply be taken away from or added to any value of x until it falls within the range $0 \leq x < 2\pi$ which can be approximated accurately with far fewer terms (see final testing), allowing much more accurate approximations to be generated in less time. The following code was implemented:

```

# Brings value within a range of increased accuracy (sine is a repeating function
so all values will have an
# equivalent value within the range 0 to 2pi)
if value > 0:
    value -= 2 * pi * (value // (2 * pi))
else:
    value += 2 * pi * (value // (-2 * pi))

```

This produces vastly more accurate results:

```

C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe
C:/Users/Joe/PycharmProjects/Coursework/maths_scripts/__init__.py

x= 999 Sin(x)= -0.0264607527370743
x= -999 Sin(x)= 0.0264607527370743

```

Process finished with exit code 0

4.7.3.2 Final Code

The code implemented is simply a variation of the code developed for a Maclaurin series. The derivatives of sin follow the following pattern: sin, cos, -sin, -cos, sin. This means the pattern repeats and so as sin and cos of 0 are known to be 0 and 1 respectively, derivatives do not need to be calculated, instead values of 0, 1, 0, -1 repeating can be substituted into consecutive terms of the expansion, meaning only every other term in the expansion needs to be calculated (as 0* anything =0) and all remaining terms can simply alternate between being positive and negative. This can be summarised with the following mathematical expression:

$$\sin(x) = \sum_{i=0}^{\infty} \frac{(-1)^i}{(2i+1)!} x^{2i+1}$$

This allows a series to be calculated more efficiently than if the base Maclaurin series code were used as every second term (each term which will give 0) is ignored and no derivatives are calculated.

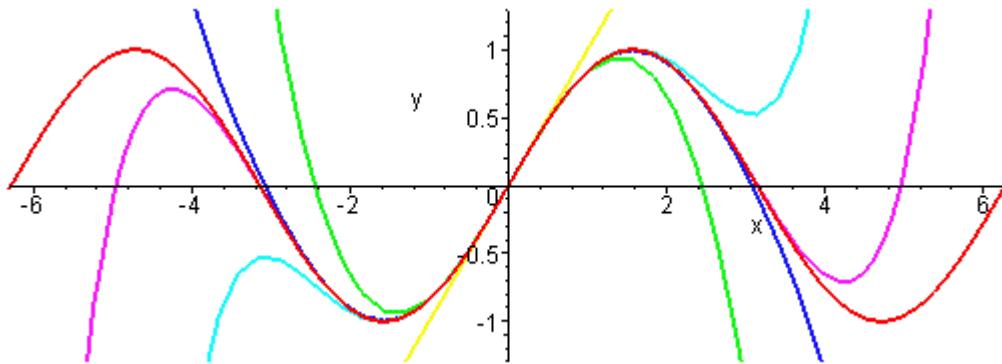
```

def sin(value: float) -> float:
    # Defines the precision of the function
    precision = 20
    # Sets up x as a variable in SymPy
    x = sympy.symbols('x')
    # Brings value within a range of increased accuracy (sine is a repeating function
    # so all values will have an
    # equivalent value within the range 0 to 2pi)
    if value > 0:
        value -= 2 * pi * (value // (2 * pi))
    else:
        value += 2 * pi * (value // (-2 * pi))
    # Calculates the Maclaurin series for sin x to the required level in a similar
    way to the previously developed
    # function
    final_equation: sympy.add.Add = sympy.add.Add(0)
    for n in range(0, precision):
        final_equation += (-1) ** n / integer_factorial(2 * n + 1) * (x ** (2 * n +
1))
    # Substitutes given value of x and returns the approximate value of sin x
    return final_equation.subs(x, value)

```

4.7.3.3 Testing

As all values can be brought within the range 0 to 2 pi of the function and the function is centred around 0, the worst-case scenario of this approximation can be considered to be when the value given is an integer multiple of 2 pi as the approximated polynomial is furthest from the true function:



Therefore, testing of the accuracy of the function needs only to be carried out at $x = 2\pi$ all other values up to this will give a more accurate approximation. The true value was checked for $\sin(2\pi)$ to be 0. The validation to bring a value down into the range $0 \leq x < 2\pi$ was temporarily removed for the sake of this one test as otherwise $\sin(2\pi)$ will be identified to be equal to $\sin(0)$. The following test code was used to allow the precision to be varied and the accuracy of the approximation for given precision analysed:

```
def sin(value: float, precision: int) -> float:
    # Defines the precision of the function
    # precision = 100
    # Sets up x as a variable in SymPy
    x = sympy.symbols('x')
    # Brings value within a range of increased accuracy (sine is a repeating function so all values will have an
    # equivalent value within the range 0 to 2pi)
    if value > 0:
        pass
    else:
        value += 2 * pi * (value // (2 * pi))
    # Calculates the Maclaurin series for sin x to the required level in a similar
    # way to the previously developed
    # function
    final_equation: sympy.add.Add = sympy.add.Add(0)
    for n in range(0, precision):
        final_equation += (-1) ** n / integer_factorial(2 * n + 1) * (x ** (2 * n + 1))
    # Substitutes given value of x and returns the approximate value of sin x
    return final_equation.subs(x, value)

if __name__ == '__main__':
    for n in range(1, 25):
        print(n, "terms gives Sin(2pi)=", sin(2*pi, n))
```

```
C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe  
C:/Users/Joe/PycharmProjects/Coursework/maths_scripts/__init__.py
```

```
1 terms gives Sin(2pi)= 6.28318530717959  
2 terms gives Sin(2pi)= -35.0585169332202  
3 terms gives Sin(2pi)= 46.5467323428549  
4 terms gives Sin(2pi)= -30.1591274102065  
5 terms gives Sin(2pi)= 11.8995665346911  
6 terms gives Sin(2pi)= -3.19507604213184  
7 terms gives Sin(2pi)= 0.624876542716422  
8 terms gives Sin(2pi)= -0.0932457590620785  
9 terms gives Sin(2pi)= 0.0109834031460778  
10 terms gives Sin(2pi)= -0.00104818279604046  
11 terms gives Sin(2pi)= 8.27409521946265e-5  
12 terms gives Sin(2pi)= -5.49438379948697e-6  
13 terms gives Sin(2pi)= 3.11268593122804e-7  
14 terms gives Sin(2pi)= -1.52242416769070e-8  
15 terms gives Sin(2pi)= 6.49425148520743e-10  
16 terms gives Sin(2pi)= -2.43967386552748e-11  
17 terms gives Sin(2pi)= 7.84895900063145e-13  
18 terms gives Sin(2pi)= -5.35445281337729e-14  
19 terms gives Sin(2pi)= -2.51228187033689e-14  
20 terms gives Sin(2pi)= -2.51228187033689e-14  
21 terms gives Sin(2pi)= -2.51228187033689e-14  
22 terms gives Sin(2pi)= -2.51228187033689e-14  
23 terms gives Sin(2pi)= -2.51228187033689e-14  
24 terms gives Sin(2pi)= -2.51228187033689e-14
```

Process finished with exit code 0

All values in the program are outputted to 2 decimal places and as can be seen above, it takes only 10 terms in the series to get a correct value of 0.00 to 2d.p in the worst-case scenario of the algorithm. At 19 terms, any further approximations calculated output to the same value and so it can be assumed that the limitation of approximation using floating point values has been reached. This value is correct to 13 decimal places, more than enough for the purposes of this program. The precision level to be used universally by the function will therefore be set to 20 as it will approximate all values to a maximum level of precision using floating point in python (1 more than the actual limit

of 19 is used to allow for any potential uncertainty in this statement and as the algorithm is a Maclaurin series and completes in O(n) time, the extra processing time is negligible).

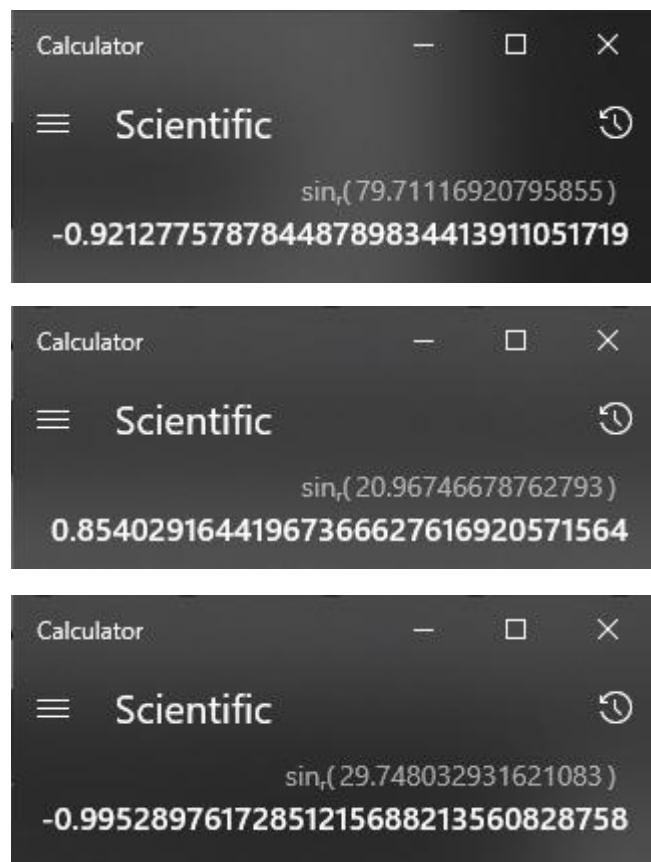
Five further tests will be done using randomly generated floats and checked against a calculator to verify function further:

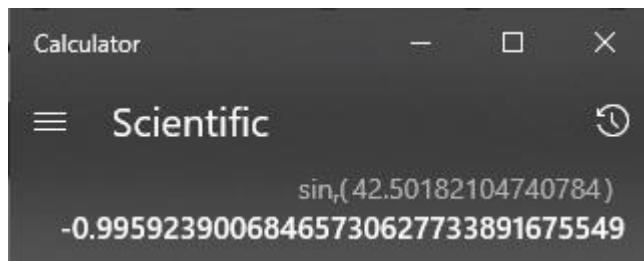
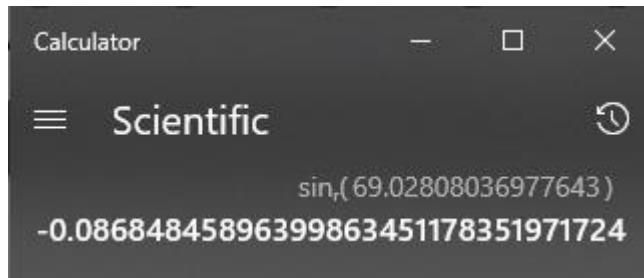
```
if __name__ == '__main__':
    for n in range(5):
        value: float = random.uniform(0, 100)
        print("x=", value, "Sin(x)=", sin(value))
```

```
C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe
C:/Users/Joe/PycharmProjects/Coursework/maths_scripts/__init__.py
```

```
x= 79.71116920795855 Sin(x)= -0.921277578784487
x= 20.96746678762793 Sin(x)= 0.854029164419673
x= 29.748032931621083 Sin(x)= -0.995289761728507
x= 69.02808036977643 Sin(x)= -0.0868484589640188
x= 42.50182104740784 Sin(x)= -0.995923900684654
```

Process finished with exit code 0





As can be seen, all values generated are identical to the true values to the number of decimal places provided by python's floating-point implementation.

4.7.3.4 Evaluation

Function produces values of $\sin(x)$ for a given x to the maximum available level of accuracy

4.7.4 Cosine function

4.7.4.1 Final code

The derivatives of cos follow the following pattern: cos, -sin, -cos, sin, cos. This means that the expansion of cosine works almost identically to that of a sine function except repeating values of 1, 0, -1, 0 are substituted instead of values of 0, 1, 0, -1. This can be summarised mathematically as:

$$\begin{aligned}\cos x &= 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots \\ &= \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!}.\end{aligned}$$

The code for sine was therefore for the most part copied, though with the equation for each term modified to fit the equation for cosine:

```

def cos(value: float) -> float:
    # Defines the precision of the function
    precision = 20
    # Sets up x as a variable in SymPy
    x = sympy.symbols('x')
    # Brings value within a range of increased accuracy (cosine is a repeating
function so all values will have an
    # equivalent value within the range 0 to 2pi)
    if value > 0:
        value -= 2 * pi * (value // (2 * pi))
    else:
        value += 2 * pi * (value // (-2 * pi))
    # Calculates the Maclaurin series for cos x to the required level in a similar
way to the previously developed
    # function
    final_equation: sympy.add.Add = sympy.add.Add(1)
    for n in range(0, precision - 1):
        final_equation += (-1) ** (n + 1) / integer_factorial(2 * (n + 1)) * (x **
(2 * (n + 1)))
    # Substitutes given value of x and returns the approximate value of cos x
    return final_equation.subs(x, value)

```

Precision has been set to equal to that of the sine function and so will be similarly accurate (see sine function testing for further elaboration)

4.7.4.2 Testing

Five tests will be done using randomly generated floats and checked against a calculator to verify function further:

```

if __name__ == '__main__':
    for n in range(5):
        value: float = random.uniform(0, 100)
        print("x=", value, "Cos(x)=", cos(value))

```

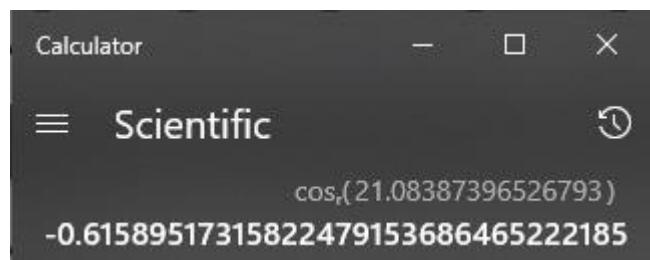
```

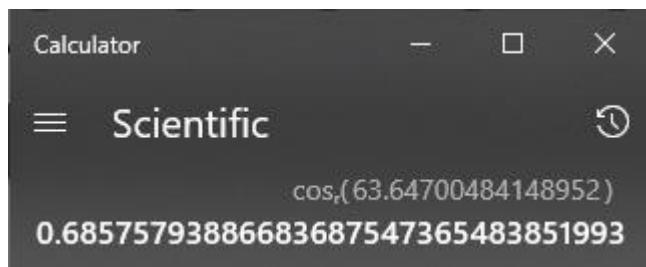
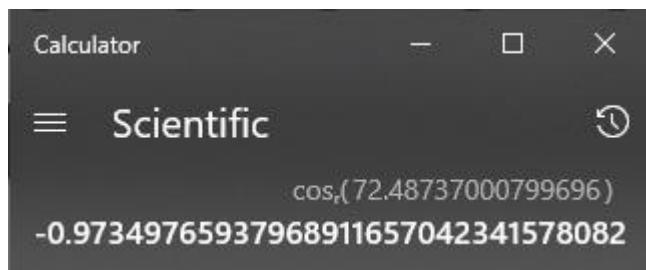
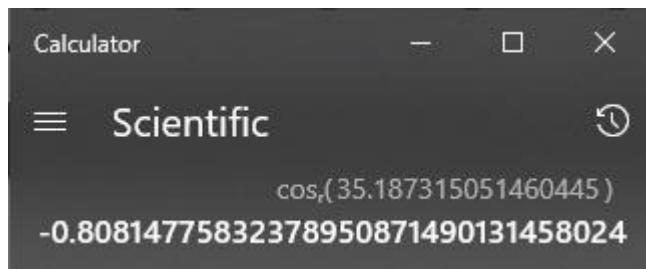
C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe
C:/Users/Joe/PycharmProjects/Coursework/maths_scripts/__init__.py

x= 21.08387396526793 Cos(x)= -0.615895173158226
x= 35.187315051460445 Cos(x)= -0.808147758323790
x= 72.48737000799696 Cos(x)= -0.973497659379688
x= 63.64700484148952 Cos(x)= 0.685757938866836
x= 41.1944300940836 Cos(x)= -0.938088697342334

```

Process finished with exit code 0





As can be seen, all values generated are identical to the true values to the number of decimal places provided by python's floating-point implementation.

4.7.4.3 Evaluation

Function produces values of $\cos(x)$ for a given x to the maximum available level of accuracy

4.8 CALCULUS SCRIPTS (MATHS_SCRIPTS.CALCULUS)

4.8.1 Integral approximation base class

As both Simpson's rule and the trapezium rule both work by splitting an area into a number of strips, a base class can be developed from which both Simpson's rule and trapezium rule classes can inherit and build upon through polymorphism.

4.8.1.1 Function to initialise class

4.8.1.1.1 Final Code

```
class IntegralApproximationBase:  
    def __init__(self, number_of_strips: int, upper_limit: float, lower_limit: float, function):  
        # Stores all constant values related to the range within which the integral  
        # is to be calculated, what function  
        # is being integrated and how many strips are to be used (accuracy level of  
        # approximation)  
        self.number_of_strips: int = number_of_strips  
        self.upper_limit: float = upper_limit  
        self.lower_limit: float = lower_limit  
        self.function = function  
        # Calculates strip width as the range of the integral divided by the de-  
        # sired number of strips  
        self.strip_width: float = (upper_limit - lower_limit) / number_of_strips  
        # Finds and stores the y values at the boundary between each strip  
        self.y_list: list = self.get_y_values()
```

4.8.1.1.2 Testing

Code to test generation of y list will be given after this function in documentation

4.8.1.1.2.1 With valid data:

```
if __name__ == '__main__':  
    x1 = sympy.symbols('x')  
    f = sympy.exp(x1)  
    instance = IntegralApproximationBase(5, 5, 1, f)  
    print(instance.number_of_strips)  
    print(instance.upper_limit)  
    print(instance.lower_limit)  
    print(instance.function)  
    print(instance.strip_width)
```

C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe
C:/Users/Joe/PycharmProjects/Coursework/math斯_scripts/calculus.py

5

5

0

exp(x)

0.8

Process finished with exit code 0

Function correctly stores all attributes and correctly calculates strip width ($(5-1)/5 = 0.8$ as calculated by program)

4.8.1.1.2.2 With Invalid data:

N/A, values used to initialise instance of class should be verified before initialising instance

4.8.1.1.2.3 Evaluation

Class is correctly initialised with all required data stored and strip width correctly calculated

4.8.1.2 Function to generate y values

4.8.1.2.1 Final Code

```
def get_y_values(self) -> list:
    # Variable to keep track of number of y values calculated
    y_counter: int = 0
    # List to store all y values
    y_values: list = []
    # Until each y value has been found, calculates the y for value for each x
    # value from the lower limit to the
    # upper limit at intervals of the strip width and stores within the y_values
    # attribute
    while y_counter <= self.number_of_strips:
        x_value: float = self.lower_limit + (self.strip_width * y_counter)
        x = sympy.symbols('x')
        y_values.append(self.function.subs(x, x_value))
        y_counter += 1
    # Returns list of y values
    return y_values
```

4.8.1.2.2 Testing

4.8.1.2.2.1 With valid data:

```
if __name__ == '__main__':
    x1 = sympy.symbols('x')
    f = sympy.exp(x1)
    instance = IntegralApproximationBase(5, 5, 1, f)
    print(instance.y_list)
```

```
C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe
C:/Users/Joe/PycharmProjects/Coursework/math_scripts/calculus.py
```

```
[2.71828182845905, 6.04964746441295, 13.4637380350017, 29.9641000473970,
66.6863310409252, 148.413159102577]
```

Process finished with exit code 0

Correct values:

1	1.8	2.6	3.4	4.2	5
2.718282	6.049647	13.46374	29.9641	66.68633	148.4132

Values generated match the actual values and there are the correct number of y values generated in the correct order

4.8.1.2.2.2 With Invalid data:

N/A, class properties should be verified when class is initialised

4.8.1.2.2.3 Evaluation

Correctly generates all required y values at the required intervals

4.8.2 Trapezium rule class

4.8.2.1 Function to initialise class

4.8.2.1.1 Final Code

```
class TrapeziumRule(IntegralApproximationBase):
    def __init__(self, number_of_strips: int, upper_limit: float, lower_limit: float, function):
        # Initialises child class with no changes to parent class
        super().__init__(number_of_strips, upper_limit, lower_limit, function)
```

4.8.2.1.2 Testing

Further testing not required as no changes are made to instantiation of parent class

4.8.2.1.3 Evaluation

Correctly initialises class

4.8.2.2 Function to approximate integral

4.8.2.2.1 Final Code

```
def integral(self):
    # Approximates integral using equation defined in design. Y sum stores the sum
    # of all the y values, with all
    # but the first and last being added twice as they are both the end of one tra-
    # pezium and start of the next
    y_sum: float = self.y_list[0] + self.y_list[-1]
    counter: int = 1
    for y in self.y_list[1:-1]:
        y_sum += y * 2
        counter += 1
    # Multiplies y sum by half the strip width to get the approximate integral and
    # returns the value
    return float((self.strip_width / 2) * y_sum)
```

4.8.2.2.2 Testing

4.8.2.2.2.1 With valid data:

The following application of the trapezium rule was found online

(<https://revisionmaths.com/advanced-level-maths-revision/pure-maths/calculus/trapezium-rule>) as an example of how the trapezium rule is applied:

Evaluate

$$\int_0^1 \sqrt{2x+1} dx$$

We'll use values of x at 0, 0.25, 0.5, 0.75 and 1 (I've just chosen regularly spaced values between 0 and 1).

So "h" in this case is 0.25

We need to find the value of $\sqrt{2x+1}$ at these values of x :

x	$\sqrt{2x+1}$
0	1
0.25	1.22
0.5	1.41
0.75	1.58
1	1.73

Now apply the trapezium rule:

$$1/2 \times 0.25 \times [1 + 1.73 + 2(1.22 + 1.41 + 1.58)] \\ = 1.39$$

When the program was run using the same function $(2x+1)^{0.5}$, strip count (4) and boundaries (0 to 1) the following occurred:

```
if name == 'main':
    x1 = sympy.symbols('x')
    f = (2 * x1 + 1) ** 0.5
    instance = TrapeziumRule(4, 1, 0, f)
    print(instance.integral())
```

```
C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe
C:/Users/Joe/PycharmProjects/Coursework/math斯_scripts/calculus.py
```

1.3965306669083282

Process finished with exit code 0

The result generated matches the value calculated in the example to 3 significant figures, however, contains many further digits. This should not be treated as an error however as in the example workings, all y values have been rounded to 2 decimal places, whereas the program has left values unrounded, producing a more accurate result.

4.8.2.2.2 With invalid data

N/A, class properties should be verified when class is initialised

4.8.2.2.3 Evaluation

Correctly finds an approximation to the integral given using the level of precision defined

4.8.3 Simpson's rule class

4.8.3.1 Function to initialise class

4.8.3.1.1 Final Code

```
class SimpsonsRule(IntegralApproximationBase):
    def __init__(self, number_of_strips: int, upper_limit: float, lower_limit: float, function):
        # Verifies that strip count is even as required(see design) and errors if not (validation should occur before
        # using class), then initialises class identically to parent class
        if number_of_strips % 2 != 0:
            raise Exception('Number of strips must be even')
        super().__init__(number_of_strips, upper_limit, lower_limit, function)
```

4.8.3.1.2 Testing

4.8.3.1.2.1 With valid data

```
if __name__ == '__main__':
    x1 = sympy.symbols('x')
    f = (2 * x1 + 1) ** 0.5
    instance = SimpsonsRule(4, 1, 0, f)
```

```
C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe
C:/Users/Joe/PycharmProjects/Coursework/maths_scripts/calculus.py
```

Process finished with exit code 0

No errors occur as strip count is even

4.8.3.1.2.2 With invalid data

```
if __name__ == '__main__':
    x1 = sympy.symbols('x')
    f = (2 * x1 + 1) ** 0.5
    instance = SimpsonsRule(5, 1, 0, f)
```

```
C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe
C:/Users/Joe/PycharmProjects/Coursework/maths_scripts/calculus.py
```

Traceback (most recent call last):

```
File "C:/Users/Joe/PycharmProjects/Coursework/maths_scripts/calculus.py", line 82, in
<module>
    instance = SimpsonsRule(5, 1, 0, f)
File "C:/Users/Joe/PycharmProjects/Coursework/maths_scripts/calculus.py", line 58, in __init__
    raise Exception('Number of strips must be even')

Exception: Number of strips must be even
```

Process finished with exit code 1

Error raised as strip count is not even

4.8.3.1.3 Evaluation

Correctly initialises class after validating that a valid strip count was entered (even integer)

4.8.3.2 Function to approximate integral

4.8.3.2.1 Final Code

```
def integral(self):
    # Approximates integral using equation defined in design. y sum stores the sum
    # of all y values, with the first
    # and last value being added once, and all other values from the second y value
    # through to the penultimate
    # alternating between being added four times and twice respectively
    y_sum: float = self.y_list[0] + self.y_list[-1]
    counter: int = 1
    for y in self.y_list[1:-1]:
        if counter % 2 == 0:
            y_sum += y * 2
        else:
            y_sum += y * 4
        counter += 1
    # Multiplies y sum by 1/3 the strip width to get the approximate integral and
    # returns the value
    return float((self.strip_width / 3) * y_sum)
```

4.8.3.2.2 Testing

4.8.3.2.2.1 With valid data

The following application of Simpson's rule was found online

(<http://tutorial.math.lamar.edu/Classes/CalcII/ApproximatingDefIntegrals.aspx>) as an example of how Simpson's rule is applied:

$$\int_0^2 e^{x^2} dx$$

Simpson's Rule

$$\int_0^2 e^{x^2} dx \approx \frac{1/2}{3} (e^{(0)^2} + 4e^{(0.5)^2} + 2e^{(1)^2} + 4e^{(1.5)^2} + e^{(2)^2}) = 17.35362645$$

When the program was run using the same function $e^{(x^2)}$, strip count (4) and boundaries (0 to 2) the following occurred:

```
if __name__ == '__main__':
    x1 = sympy.symbols('x')
    f = sympy.exp(x1 ** 2)
    instance = SimpsonsRule(4, 2, 0, f)
    print(instance.integral())
```

```
C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe
C:/Users/Joe/PycharmProjects/Coursework/maths_scripts/calculus.py
```

17.353626450374566

Process finished with exit code 0

The result generated matches the value calculated in the example, though includes additional significant figures compared to the example

4.8.3.2.2.2 With invalid data

N/A, class properties should be verified when class is initialised

4.8.3.2.3 Evaluation

Correctly finds an approximation to the integral given using the level of precision defined

4.8.4 Comparison of Simpson's rule to Trapezium rule

The following code was used to calculate the integral of e^x with limits of 0 and 5 with increasing strip counts upon each iteration. The true value was also calculated manually to be 147.41315910257660342111558004055 to 32 significant figures (more than enough for almost any usage. For comparison, NASA requires only 16 decimal places for their value of pi to precisely enough carry out all their work <https://www.jpl.nasa.gov/edu/news/2016/3/16/how-many-decimals-of-pi-do-we-really-need/>).

```
if __name__ == '__main__':
    x1 = sympy.symbols('x')
    f = sympy.exp(x1)
    true_value = 147.41315910257660342111558004055
    upper_limit = 5
    lower_limit = 0
    for n in range(1, 51):
        strip_count = n * 2
        trapezium = TrapeziumRule(strip_count, upper_limit, lower_limit, f)
        simpson = SimpsonsRule(strip_count, upper_limit, lower_limit, f)
        print("Strip count:", strip_count, "Trapezium Rule:", trapezium.integral(),
"Simpson's Rule:",
            simpson.integral())
```

```
C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe  
C:/Users/Joe/PycharmProjects/Coursework/maths_scripts/calculus.py
```

```
Strip count: 2 Trapezium Rule: 217.22268377997943 Simpson's Rule: 165.11927912115877  
Strip count: 4 Trapezium Rule: 166.12562308689547 Simpson's Rule: 149.0932695225342  
Strip count: 6 Trapezium Rule: 155.84688117728047 Simpson's Rule: 147.77767796066036  
Strip count: 8 Trapezium Rule: 152.18081137066676 Simpson's Rule: 147.5325407985905  
Strip count: 10 Trapezium Rule: 150.47154599798284 Simpson's Rule: 147.46285985230713  
Strip count: 12 Trapezium Rule: 149.53972698356162 Simpson's Rule: 147.437342252322  
Strip count: 14 Trapezium Rule: 148.97672978147574 Simpson's Rule: 147.42628336643932  
Strip count: 16 Trapezium Rule: 148.61086243383915 Simpson's Rule: 147.4208794548966  
Strip count: 18 Trapezium Rule: 148.35981504505287 Simpson's Rule: 147.41799056207466  
Strip count: 20 Trapezium Rule: 148.18013739360293 Simpson's Rule: 147.4163345254763  
Strip count: 22 Trapezium Rule: 148.0471398606064 Simpson's Rule: 147.4153307471685  
Strip count: 24 Trapezium Rule: 147.94595219250706 Simpson's Rule: 147.41469392882215  
Strip count: 26 Trapezium Rule: 147.86718516848688 Simpson's Rule: 147.41427427375282  
Strip count: 28 Trapezium Rule: 147.80467396747437 Simpson's Rule: 147.4139886961406  
Strip count: 30 Trapezium Rule: 147.75423539279677 Simpson's Rule: 147.41378893410734  
Strip count: 32 Trapezium Rule: 147.71294997859468 Simpson's Rule: 147.41364582684653  
Strip count: 34 Trapezium Rule: 147.6787301325279 Simpson's Rule: 147.4135411445491  
Strip count: 36 Trapezium Rule: 147.6500511216505 Simpson's Rule: 147.41346314718305  
Strip count: 38 Trapezium Rule: 147.6257783323121 Simpson's Rule: 147.413404073743  
Strip count: 40 Trapezium Rule: 147.60505335327326 Simpson's Rule: 147.4133586731634  
Strip count: 42 Trapezium Rule: 147.58721705684147 Simpson's Rule: 147.41332331815013  
Strip count: 44 Trapezium Rule: 147.57175655709725 Simpson's Rule: 147.41329545592754  
Strip count: 46 Trapezium Rule: 147.55826793585553 Simpson's Rule: 147.41327325941282  
Strip count: 48 Trapezium Rule: 147.54642959869653 Simpson's Rule: 147.41325540075974  
Strip count: 50 Trapezium Rule: 147.53598293265244 Simpson's Rule: 147.4132409013829
```

Process finished with exit code 0

From the values seen above it is clear that Simpson's rule will produce a much more accurate approximation of the integral than the trapezium rule given an equal strip count. The code was altered to better illustrate this by instead outputting the percentage difference between the approximations and the true value, as seen below:

```
if __name__ == '__main__':
    x1 = sympy.symbols('x')
    f = sympy.exp(x1)
    true_value = 147.41315910257660342111558004055
    upper_limit = 5
    lower_limit = 0
    for n in range(1, 26):
        strip_count = n * 2
        trapezium = TrapeziumRule(strip_count, upper_limit, lower_limit, f)
        simpson = SimpsonsRule(strip_count, upper_limit, lower_limit, f)
        print("Strip count:", strip_count, "Trapezium Rule:", 100 * (trapezium.integral() - true_value) / true_value,
              "Simpson's Rule:", 100 * (simpson.integral() - true_value) / true_value)
```

```

C:\Users\Joe\PycharmProjects\Coursework\venv\Scripts\python.exe
C:/Users/Joe/PycharmProjects/Coursework/maths_scripts/calculus.py

Strip count: 2 Trapezium Rule: 47.35637245846301 Simpson's Rule: 12.01122079356665
Strip count: 4 Trapezium Rule: 12.693889811626596 Simpson's Rule: 1.1397289293478237
Strip count: 6 Trapezium Rule: 5.721146013047118 Simpson's Rule: 0.24727701400802837
Strip count: 8 Trapezium Rule: 3.234210769998231 Simpson's Rule: 0.08098442278876343
Strip count: 10 Trapezium Rule: 2.0747041268399133 Simpson's Rule: 0.03371527347565308
Strip count: 12 Trapezium Rule: 1.4425902639433033 Simpson's Rule: 0.016405014242025402
Strip count: 14 Trapezium Rule: 1.0606723907267588 Simpson's Rule: 0.00890304769439811
Strip count: 16 Trapezium Rule: 0.8124806079416086 Simpson's Rule: 0.005237220589401203
Strip count: 18 Trapezium Rule: 0.64217872287612 Simpson's Rule: 0.0032774953928651016
Strip count: 20 Trapezium Rule: 0.5202916046949623 Simpson's Rule: 0.0021540973133055367
Strip count: 22 Trapezium Rule: 0.43007066797112803 Simpson's Rule: 0.0014731687490570407
Strip count: 24 Trapezium Rule: 0.36142844585517886 Simpson's Rule: 0.0010411731590923494
Strip count: 26 Trapezium Rule: 0.30799561496023076 Simpson's Rule: 0.0007564936420959908
Strip count: 28 Trapezium Rule: 0.2655901734154809 Simpson's Rule: 0.0005627676450613749
Strip count: 30 Trapezium Rule: 0.2313743849576119 Simpson's Rule: 0.00042725597536813584
Strip count: 32 Trapezium Rule: 0.20336778469652864 Simpson's Rule: 0.0003301769481750697
Strip count: 34 Trapezium Rule: 0.1801542220301426 Simpson's Rule: 0.0002591640901165444
Strip count: 36 Trapezium Rule: 0.16069937074550186 Simpson's Rule: 0.00020625336862915868
Strip count: 38 Trapezium Rule: 0.14423354809699118 Simpson's Rule: 0.00016617998549193754
Strip count: 40 Trapezium Rule: 0.1301744375229992 Simpson's Rule: 0.00013538179903079816
Strip count: 42 Trapezium Rule: 0.11807490954301457 Simpson's Rule: 0.00011139817810952494
Strip count: 44 Trapezium Rule: 0.10758704004863387 Simpson's Rule: 9.249740780889742e-05
Strip count: 46 Trapezium Rule: 0.09843682488206652 Simpson's Rule: 7.744005821026527e-05
Strip count: 48 Trapezium Rule: 0.09040610548695895 Simpson's Rule: 6.532536425111435e-05
Strip count: 50 Trapezium Rule: 0.08331944775050822 Simpson's Rule: 5.548948736325513e-05

```

Process finished with exit code 0

As can be seen above, with only 2 strips, while the trapezium rule approximation falls within a relatively large 48% of the true value, Simpson's rule is already within 13% of the true value. As more and more strips are used, Simpson's rule also gains more accuracy at a faster rate, with the trapezium rule requiring only 4 more strips (6 in total) to beat the accuracy of Simpson's rule with only 2 strips, but 10 more strips (14 in total) to beat the accuracy of Simpson's rule with 4 strips. The gap only further widens as the number of strips further increases.

This does not yet provide enough information to guarantee that Simpson's rule will produce a more accurate result in an equal amount of time to the Trapezium rule as it is possible that calculating a trapezium rule value with a far greater strip count may be faster than a Simpson's rule value with a smaller strip count. Looking at the algorithms however indicates that this is not the case as both algorithms contain only a single loop proportional in number of iterations to the number of strips being used – 2, so each algorithm has a time complexity of $O(n)$, where n is the number of strips used. While each loop of the Simpson's rule algorithm requires an if statement to be carried out while the Trapezium rule does not, the time this would take is negligible when compared to how many fewer strips are needed to give an approximation of similar accuracy and so Simpson's rule is still more effective.

Due to the above reasoning, it was decided that where definite integrals are required in the program, Simpson's rule can be used as it will be able to very quickly approximate to the 2 decimal places used in the main program and require less processing power and time than if the trapezium rule were used.

5 EVALUATION

5.1 FINAL TESTING

Extensive testing was carried out on the code during development to the point where further direct testing of code and the potential incorrect usage of the program is redundant as it has been covered already. However, all functionality within the user interface itself should be retested now in order to assess usability and ensure all user accessible functions of the program work as intended and are clear how to use. While all tests will be specified by me, each test will be carried out by one of the stakeholders and their feedback analysed to determine success. For the most part, this will simply take the form of teacher and student creating accounts and utilising each feature of the program to set a quiz, attempt a quiz, and view results. The table of test criteria will be revisited throughout this process and will be included in documentation after the testing with it being labelled where criteria have been met, and if they haven't why they haven't.

5.1.1 Testing

5.1.1.1 Robustness of program

5.1.1.1.1 Test plan

Test Number	Test	Test method	Expected outcome
1	Pages change within a reasonable amount of time	All page transition buttons will be rapidly clicked when used and ensured to still complete in a reasonable amount of time to the stakeholder each time	Pages change near instantly any time transitions should occur
2	Deleting database while in use	Delete file using file explorer whilst a student is attempting a quiz	OS will block removal of file while in use

5.1.1.1.2 Pages change within a reasonable amount of time (SUCCESS)

To ensure the act of loading a new page in the program is responsive, an easily accessible page change link will be selected with another page change link in roughly the same position and these will be rapidly clicked. It will then be checked if the number of page transitions matches the number of clicks and that the final page transition occurs within 1 second of the final click. The buttons to be used are the ‘change account’ button on the login page and ‘login’ button on the create account page.

 Maths Quiz - Joe Down

	Username Password <input type="text" value="Username"/> <input type="password" value="Password"/> <input type="button" value="Submit"/>
--	----------------------------------------------------------------------------------------------------------------------------------------------------

[Create Account](#)

Joe Down

 Maths Quiz - Joe Down

	First Name Last Name <input type="text" value="First Name"/> <input type="text" value="Last Name"/> Username: <input type="text" value="Username"/> Password: <input type="password" value="Password"/> Verify Password: <input type="password" value="Verify Password"/> User Type: <input checked="" type="radio"/> Student <input type="radio"/> Teacher <input type="button" value="Submit"/>
--	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

[Login](#)

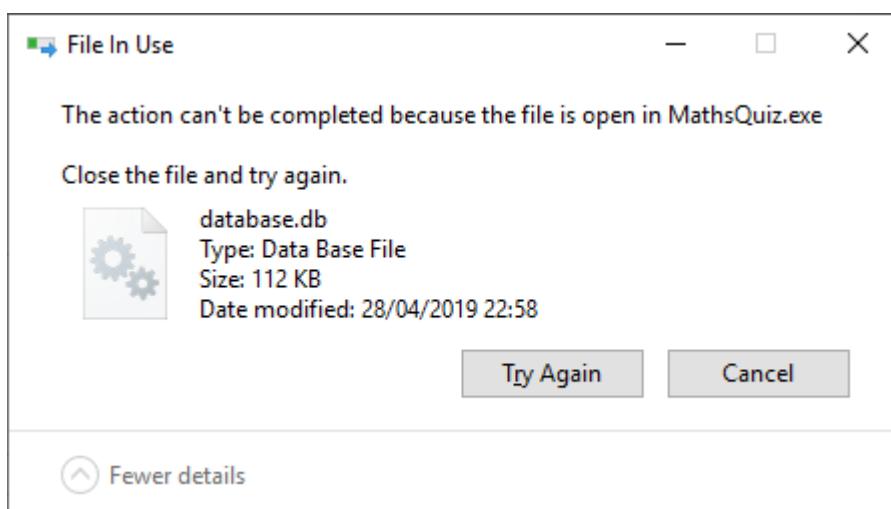
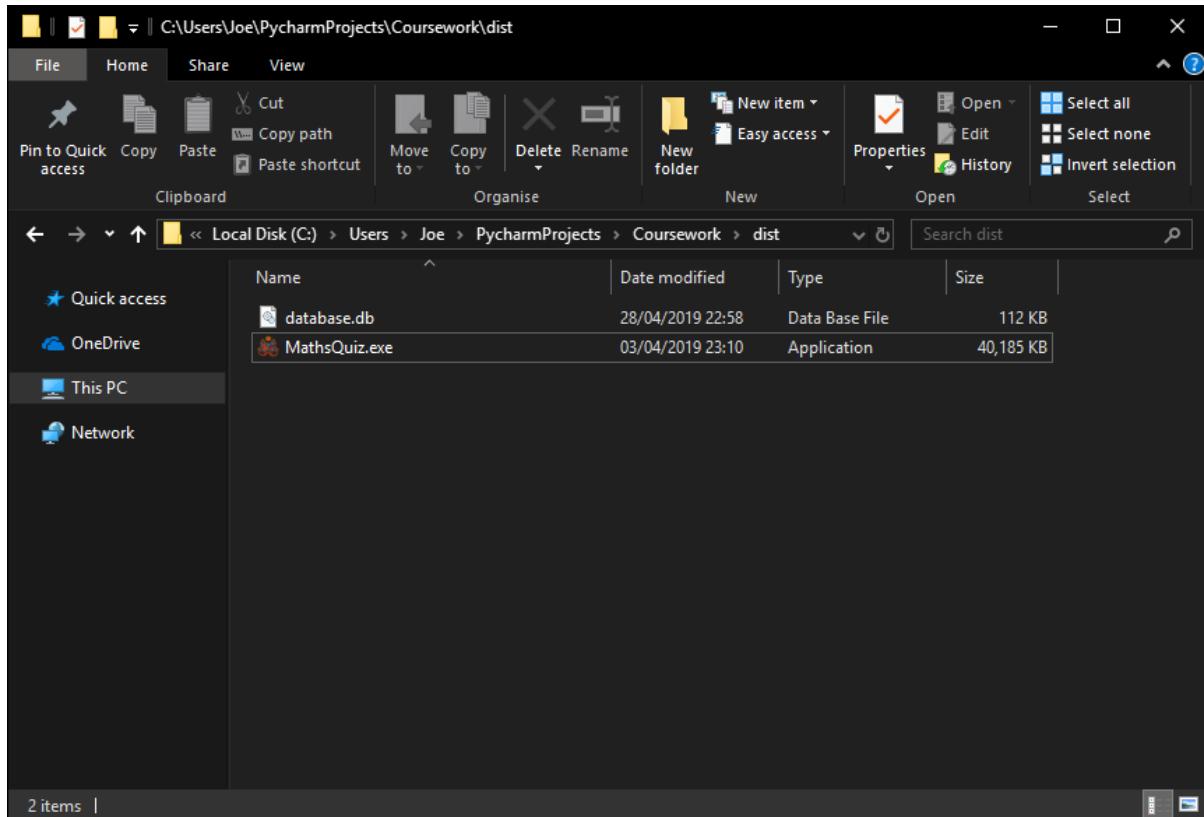
Joe Down

Stakeholder 'Troy Garvin' clicked each of the two buttons 10 times (20 clicks in total) and gave the following feedback:

"I clicked the buttons as quickly as I could while also still being able to even count the number of page changes I saw and counted exactly one change to the correct page for each of my 20 clicks. In terms of timings, all page changes appeared to me to be instant"

The response given by the stakeholder validates all aspects of the given criteria, indicating that page transitions in the program are sufficiently responsive. This will be re-tested on all buttons in the program during further testing and any issues that arise discussed further if/when they arise.

5.1.1.1.3 Deleting database while in use (SUCCESS)



Database file cannot be removed through file explorer (may be possible to intentionally force removal through command line though this is very much not something which could be done unless intentionally trying to cause an error) while in use by the program and is simply rebuilt if not present when program starts (see previous testing).

```

Select Administrator: Command Prompt
Microsoft Windows [Version 10.0.17763.437]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>cd C:\Users\Joe\PycharmProjects\Coursework\dist
C:\Users\Joe\PycharmProjects\Coursework\dist>del /f database.db
C:\Users\Joe\PycharmProjects\Coursework\dist\database.db
The process cannot access the file because it is being used by another process.

C:\Users\Joe\PycharmProjects\Coursework\dist>

```

Further investigation reveals that even using an elevated command prompt and using the `/f` parameter to force deletion, the database cannot be deleted while the program is in use.

5.1.1.2 *Usability*

5.1.1.2.1 Test plan

Test Number	Test	Expected outcome	Stakeholder to perform test	Decomposition of problem area covered
25.	Go to create account screen	Loads create account screen	Either	1d
26.	Create teacher account	Teacher account permanently created	Teacher	6ai, 6aii, 6aiii, 6aiv, 6av
27.	Create student account	Student account permanently created	Student	
28.	Return from account creation to login screen	Loads login screen	Either	6e
29.	Teacher login	Loads teacher main menu as relevant teacher	Teacher	1a, 1b, 1c, 6d
30.	Student login	Loads student main menu as relevant student	Student	3
31.	Logout button	Returns to login screen and unselects current user	Either	2a, 2b, 5a
32.	Go to account management	Loads account management page	Either	7ai

33.	Set new first name, last name and password	Sets new first name and last name and only new password works when attempting to login with the edited account	Either	7ai1, 7ai2, 7ai3, 7ai4, 7ai5
34.	Home button	When clicked returns to main menu	Either	5b
35.	Create class	Enter a class name and new class is created	Teacher	7cii1a, 7cii1b
36.	Add user to class	Enter username and adds given username to class	Teacher	7cii2ai, 7cii2av
37.	Remove user from class	Select student from list and removes user from class	Teacher	7cii2bi, 7cii2bii
38.	Add homework to class	Enter homework name, description and due date and inserts homework into class	Teacher	7cii2ci, 7cii2cii, 7cii2ciii, 7cii2civ, 7cii2cv
39.	Remove homework from class	Select from list of homework in class and remove homework from database and class	Teacher	7cii2di, 7cii2dii, 7cii2diii
40.	Delete class	Select from list of classes and remove from database	Teacher	7cii3a, 7cii3b
41.	Add automatic question to homework	Select class and homework to add question to and question topic and difficulty to add an automatically generated question to the class	Teacher	7ciii1, 7ciii2, 7ciii3a, 7ciii3b, 7ciii3c, 7ciii3d

42.	Add custom question to homework	Select class and homework to add question to and enter question name, text, answer and incorrect answers and add to homework	Teacher	7ciii4a, 7ciii4b, 7ciii4c, 7ciii4d, 7ciii4e
43.	Remove question from homework	Select question to remove from list, show question details and when remove selected remove from database	Teacher	7ciii5a, 7ciii5b, 7ciii5c
44.	Select homework	Select class to do homework from and select homework to do from a list of homework, showing the due date. Go to quiz when selected and show first question	Student	7bii1, 7bii2, 7bii3, 7bii4, 8a
45.	Navigate quiz	Next question loads next question and previous loads previous	Student	8bi, 8bii
46.	Question output	Must show question text and four potential answers, allowing only 1 to be selected, submitted and a response given	Student	8c, 8di, 8dii, 8eii1, 8eii2
47.	View previous homework results	Show list of homework with name, score and date	Student	7bi1a, 7bi1b, 7bi1c
48.	View class scores	Select class view, select class, select homework and output a list of results showing for each	Teacher	7ci1, 7ci2, 7ci3, 7ci4a, 7ci4b, 7ci4c, 7ci4d, 7ci4e

		student: first name, last name, raw score, weighted percentage and number of attempts		
49.	View user score	Select student view, select class, select student and output a list of results, showing for each homework: homework name, score, percentage and number of attempts	Teacher	ADDED SINCE DESIGN

5.1.1.2.2 Go to create account screen (SUCCESS)

5.1.1.2.2.1 Attempt

The screenshot shows a web browser window with the following details:

- Title Bar:** Maths Quiz - Joe Down
- Content Area:**
 - A logo featuring a cartoon character with wings and a crown.
 - Two input fields: "Username" and "Password".
 - A "Submit" button below the password field.
 - A blue horizontal bar at the bottom containing the text "Create Account".
- Bottom Right:** The text "Joe Down" is visible.

The screenshot shows a Windows application window titled "Maths Quiz - Joe Down". The window contains a registration form with the following fields:

- First Name: [Text Box]
- Last Name: [Text Box]
- Username: [Text Box]
- Password: [Text Box]
- Verify Password: [Text Box]
- User Type:
 Student Teacher
- Submit [Button]

At the bottom of the window, there are two links: "Login" and "Joe Down".

5.1.1.2.2.2 Response from tester (teacher)

Page successfully loaded instantly upon pressing the button

5.1.1.2.3 Create teacher account (SUCCESS)

5.1.1.2.3.1 Attempt

The screenshot shows the same Windows application window as the previous one, but with the following data entered into the fields:

- First Name: Troy
- Last Name: Garvin
- Username: troygarvin
- Password: [REDACTED]
- Verify Password: [REDACTED]
- User Type:
 Student Teacher

At the bottom of the window, there are two links: "Login" and "Joe Down".

 Maths Quiz - Joe Down

First Name Last Name

Username:

Password:

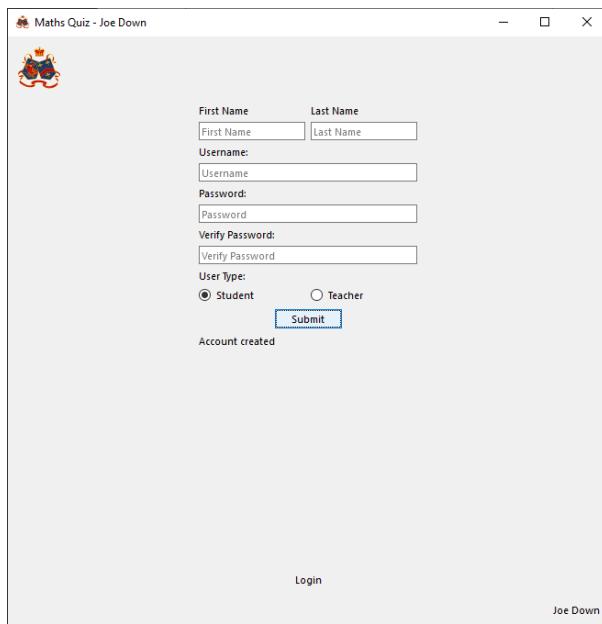
Verify Password:

User Type:
 Student Teacher

Account created

[Login](#)

Joe Down



5.1.1.2.3.2 Response from tester (teacher)

I was able to enter details to create an account and was informed by the program my submission was successful

5.1.1.2.4 Create student account (SUCCESS)

5.1.1.2.4.1 Attempt

 Maths Quiz - Joe Down

First Name Last Name

Username:

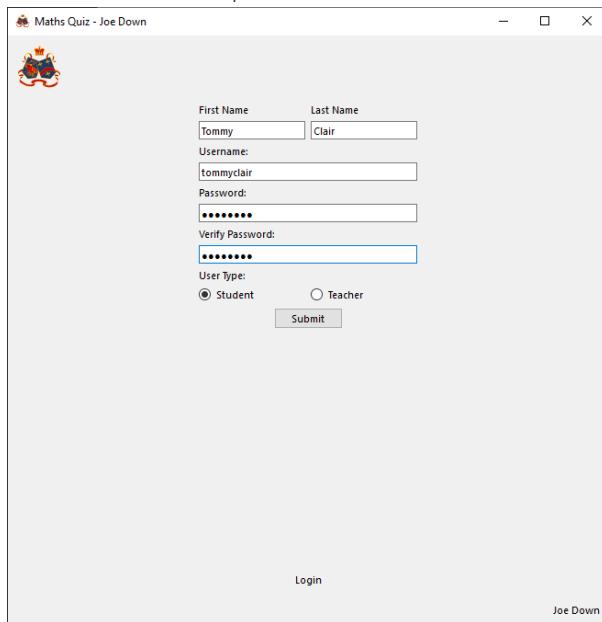
Password:

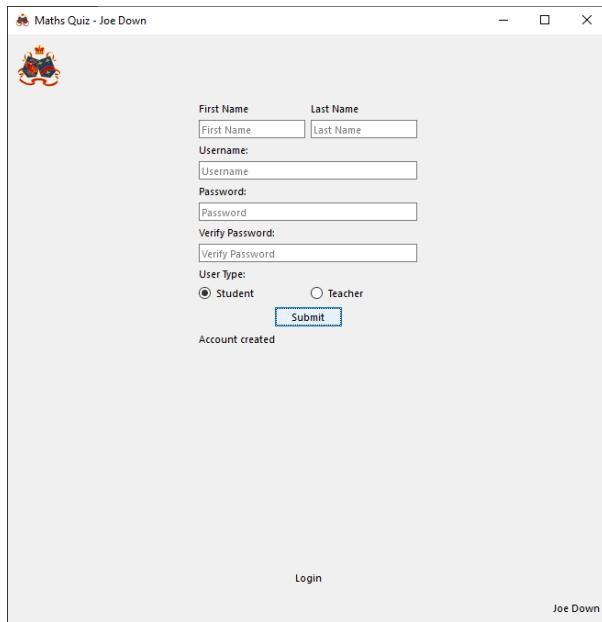
Verify Password:

User Type:
 Student Teacher

[Login](#)

Joe Down



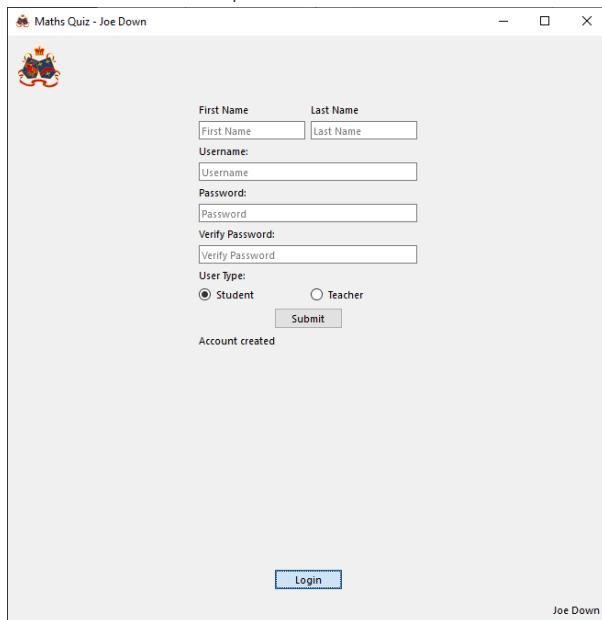


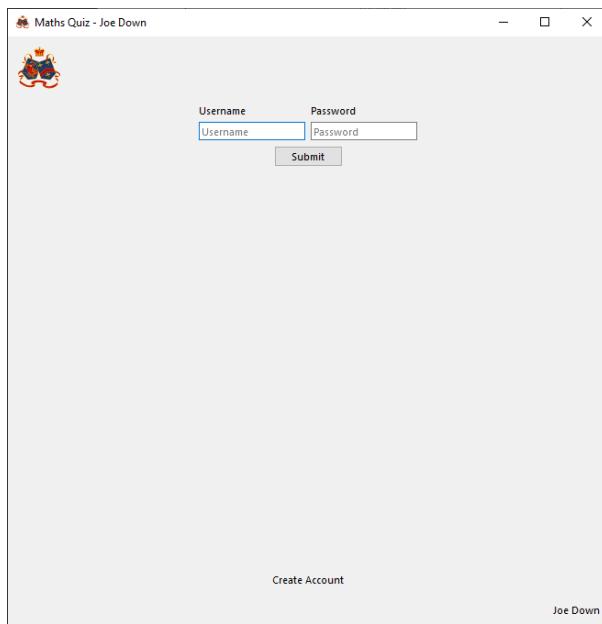
5.1.1.2.4.2 Response from tester (student)

I entered my details and was given a success message

5.1.1.2.5 Go to login screen from create account screen (SUCCESS)

5.1.1.2.5.1 Attempt



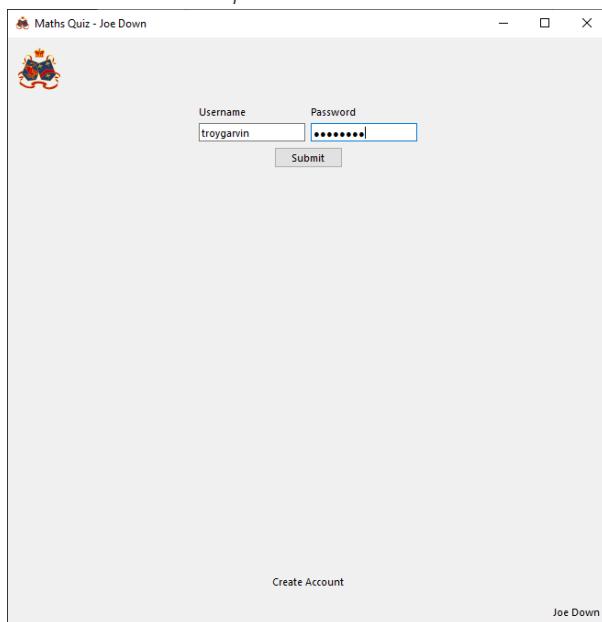


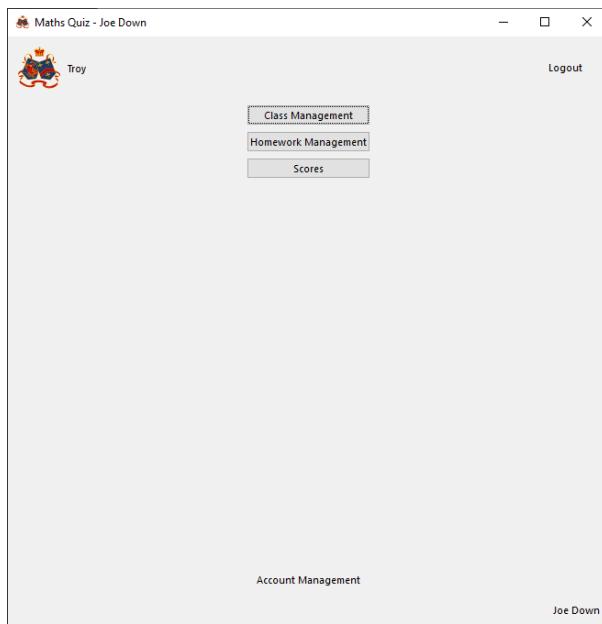
5.1.1.2.5.2 Response from tester (teacher)

The screen transition occurred seemingly instantly

5.1.1.2.6 Teacher login (SUCCESS)

5.1.1.2.6.1 Attempt



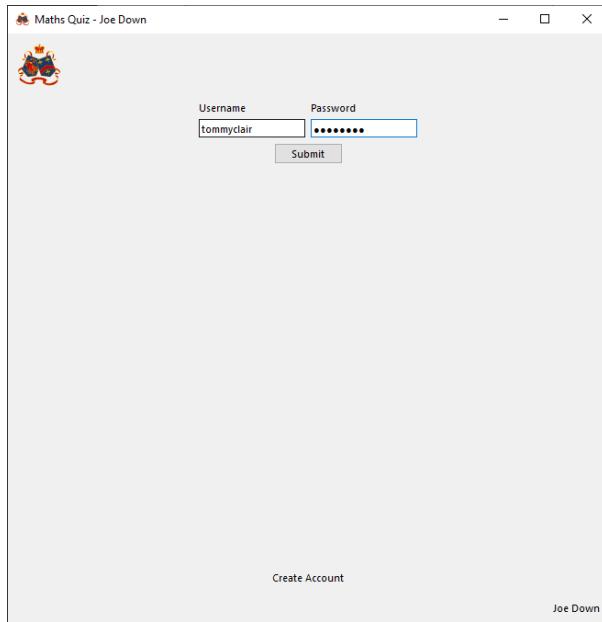


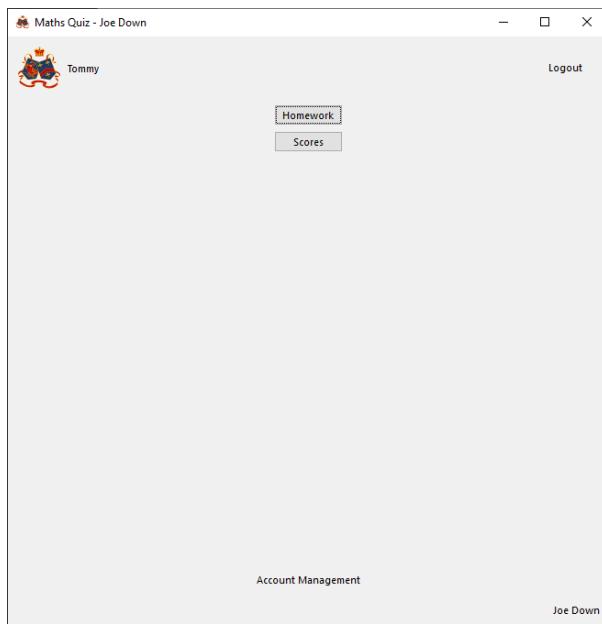
5.1.1.2.6.2 Response from tester (teacher)

I entered my login details and was taken to the teacher main menu page. My own name was in the top left to show I was logged in as myself

5.1.1.2.7 Student login (SUCCESS)

5.1.1.2.7.1 Attempt



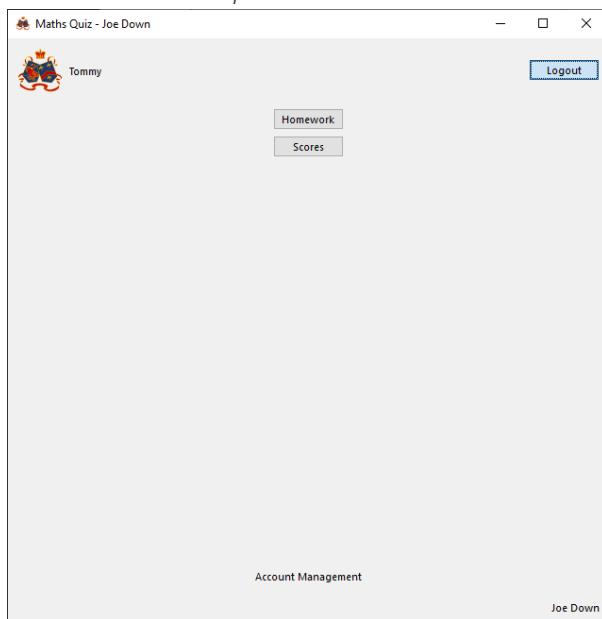


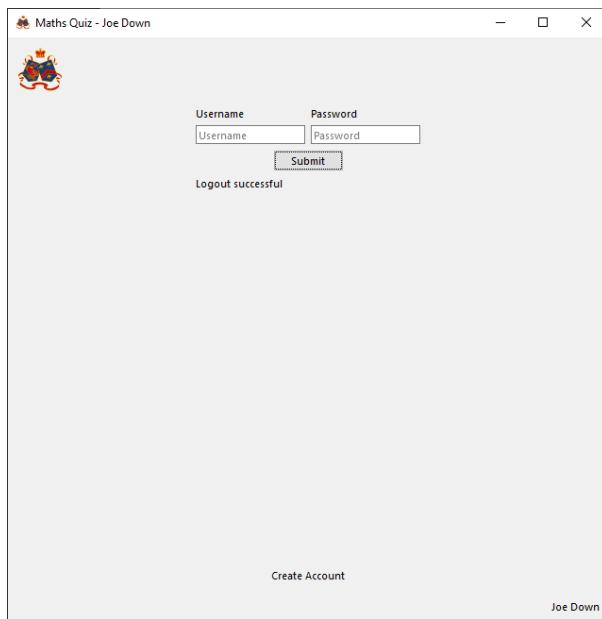
5.1.1.2.7.2 Response from tester (student)

Using my details logged in to the student main menu page as myself

5.1.1.2.8 Logout button (SUCCESS)

5.1.1.2.8.1 Attempt



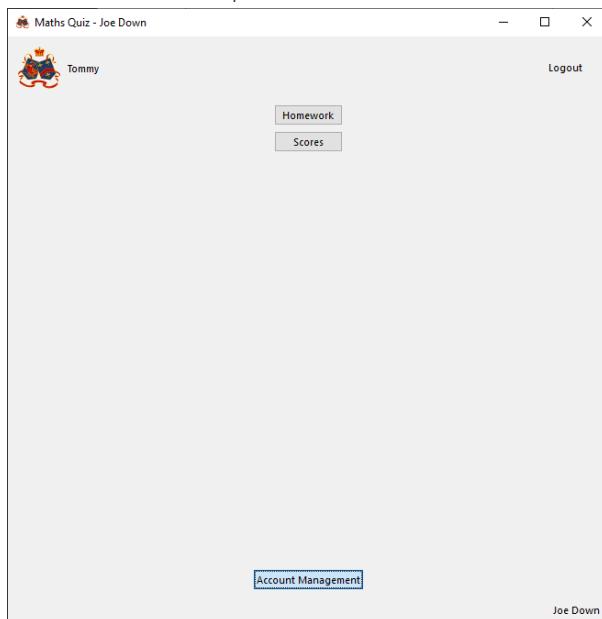


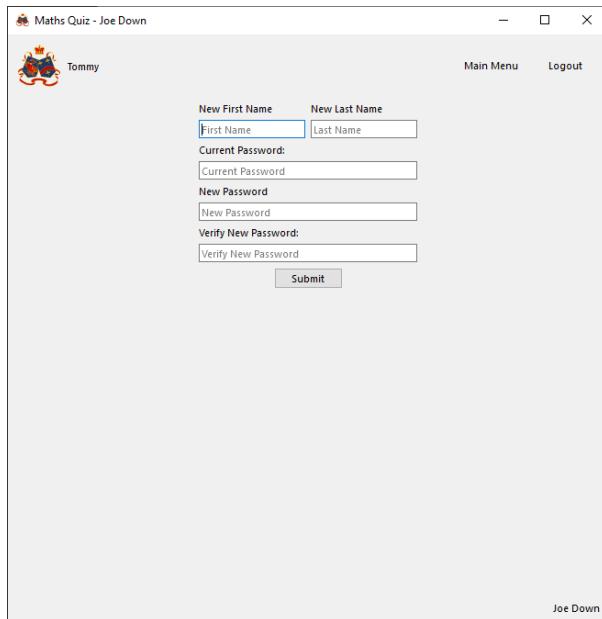
5.1.1.2.8.2 Response from tester (student)

Clicking the logout button returned me to the login screen and gave me a message saying I was logged out. My name disappeared from the top left corner to show I was fully logged out

5.1.1.2.9 Go to account management from main menu (SUCCESS)

5.1.1.2.9.1 Attempt



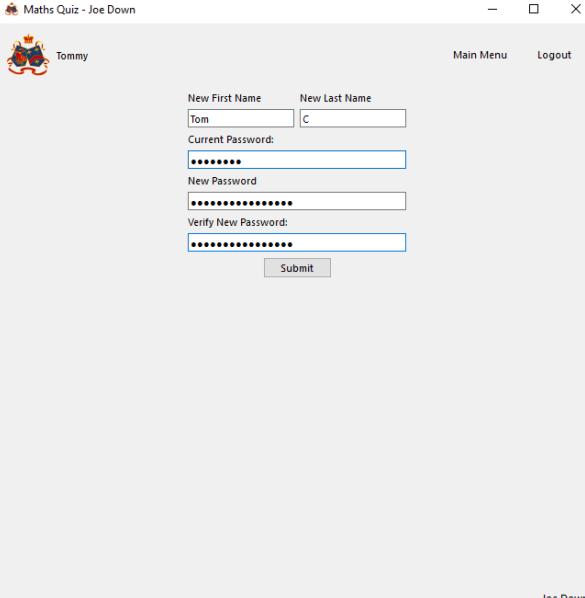


5.1.1.2.9.2 Response from tester (student)

Clicking the button to go to account management instantly loaded the account management page

5.1.1.2.10 Set new account details (SUCCESS)

5.1.1.2.10.1 Attempt



Maths Quiz - Joe Down

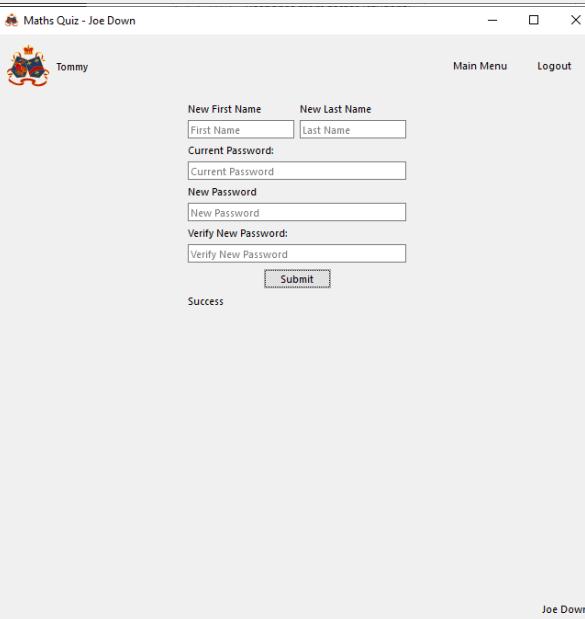
Tommy Main Menu Logout

New First Name: Tom New Last Name: C

Current Password: (Masked)

New Password: (Masked)

Verify New Password: (Masked)



Maths Quiz - Joe Down

Tommy Main Menu Logout

New First Name: First Name New Last Name: Last Name

Current Password: (Masked)

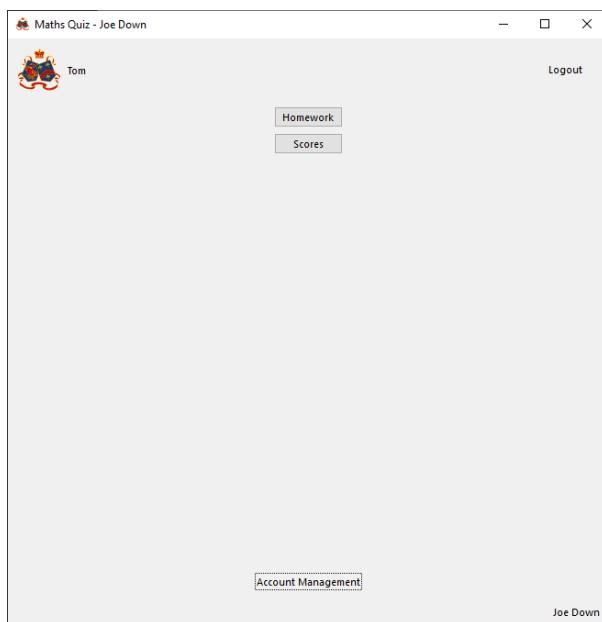
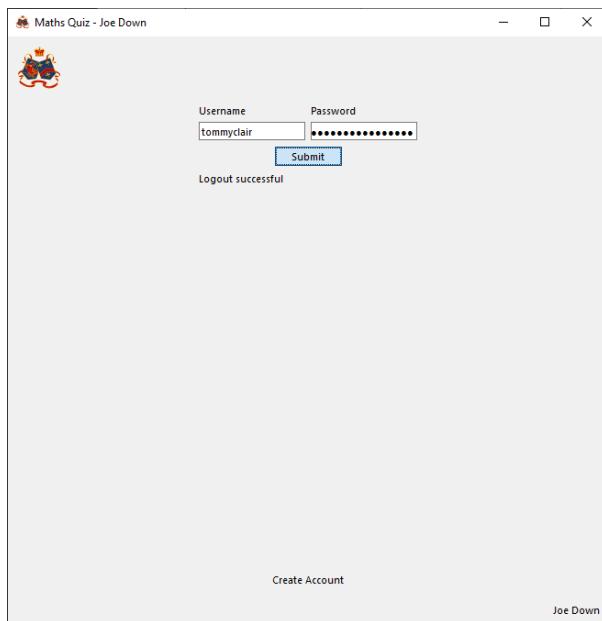
New Password: (Masked)

Verify New Password: (Masked)

Success

Joe Down

Attempting to log in with new password:

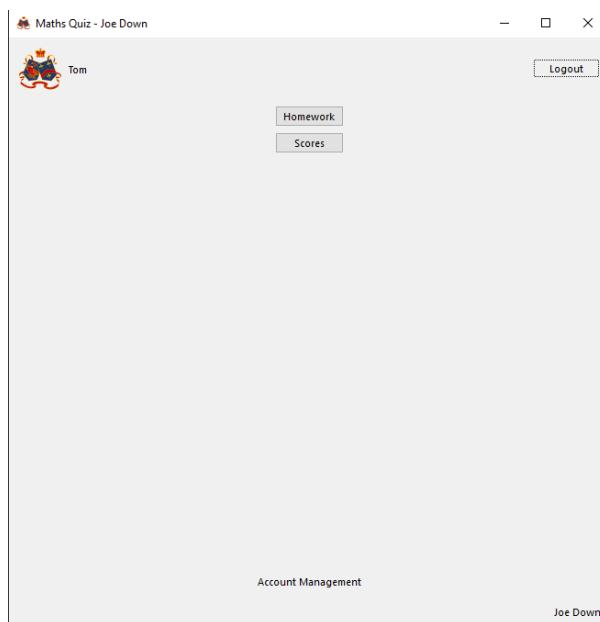
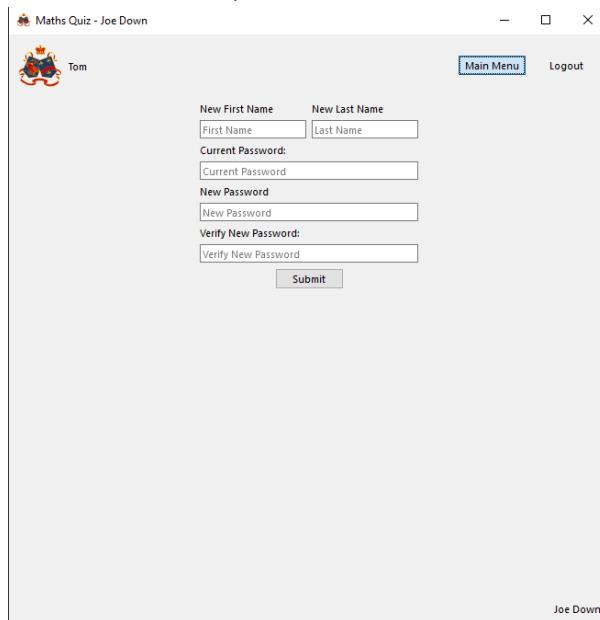


5.1.1.2.10.2 Response from tester (student)

When I entered and submitted a new first name, last name and password I was informed the changes were a success. When I tried to log back into the account, only the new password worked and not the old one and my name now displayed as the new first name, 'Tom'

5.1.1.2.11 Home button (SUCCESS)

5.1.1.2.11.1 Attempt



5.1.1.2.11.2 Response from tester (student)

Whenever I click the home button, whatever page I'm on, the program goes back to the main menu

5.1.1.2.12 Create class (SUCCESS)

5.1.1.2.12.1 Attempt

The screenshot shows the 'Maths Quiz - Joe Down' application window. At the top, there is a user icon labeled 'Troy' and navigation links for 'Main Menu' and 'Logout'. Below this is a 'Create Class:' section with a text input field containing 'My Class' and a 'Submit' button. Underneath is a 'Manage Class:' section with tabs for 'Add User', 'Remove User', 'Add Homework', and 'Remove Homework'. A sub-section for 'Add User' contains a 'Username' input field and a 'Submit' button. At the bottom is a 'Delete Class:' section with a dropdown menu and a 'REMOVE' button. The status bar at the bottom right shows 'Joe Down'.

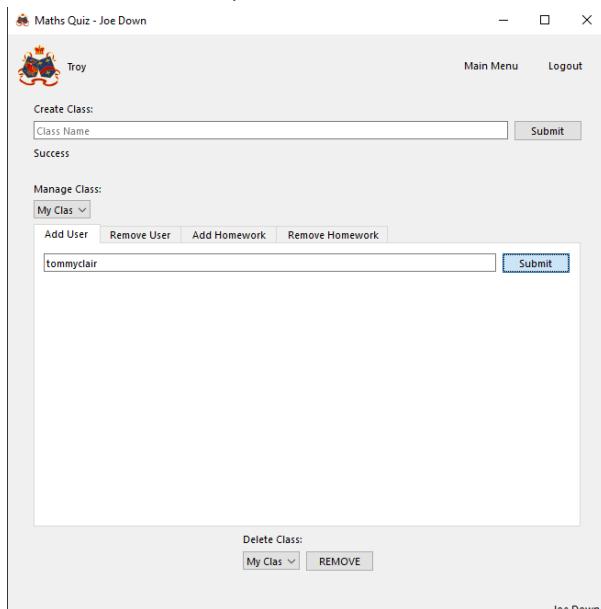
This screenshot is identical to the one above, but it includes a 'Success' message below the 'Create Class:' section, indicating that the class was created successfully. The rest of the interface, including the 'Manage Class:' section and the status bar, remains the same.

5.1.1.2.12.2 Response from tester (teacher)

When I entered a class name and clicked to create the class the program told me 'success' and the class began to show up elsewhere in the UI

5.1.1.2.13 Add user to class (SUCCESS)

5.1.1.2.13.1 Attempt



Maths Quiz - Joe Down

Troy

Main Menu Logout

Create Class:

Class Name Submit

Success

Manage Class:

My Clas

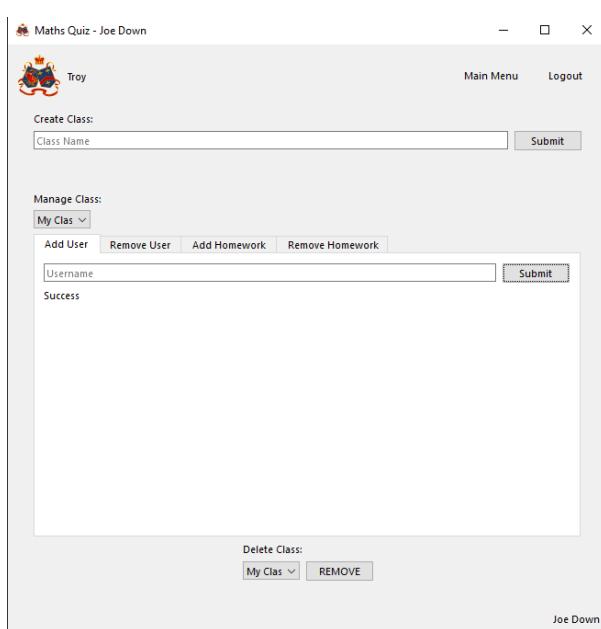
Add User Remove User Add Homework Remove Homework

tommyclair

Delete Class:

My Clas REMOVE

Joe Down



Maths Quiz - Joe Down

Troy

Main Menu Logout

Create Class:

Class Name Submit

Manage Class:

My Clas

Add User Remove User Add Homework Remove Homework

Username

Success

Delete Class:

My Clas REMOVE

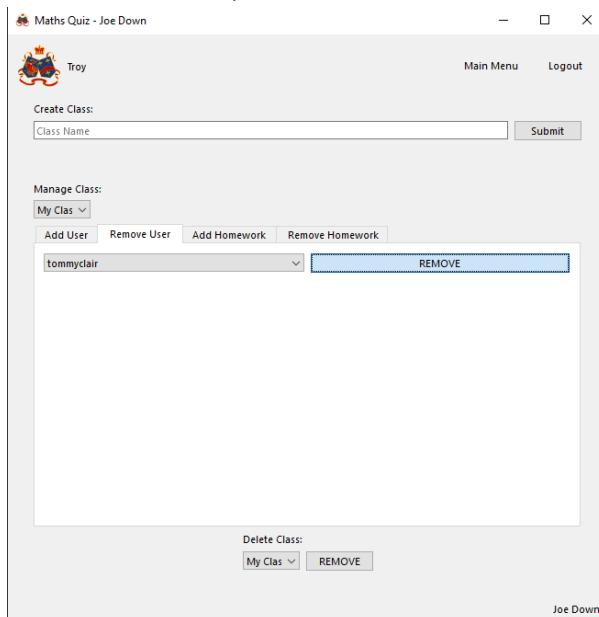
Joe Down

5.1.1.2.13.2 Response from tester (teacher)

When I typed in a student username and clicked submit the program outputted that they had successfully been added to the class and they began to show up as a class member elsewhere in the UI

5.1.1.2.14 Remove user from class (SUCCESS)

5.1.1.2.14.1 Attempt



Maths Quiz - Joe Down

Troy Main Menu Logout

Create Class: Class Name Submit

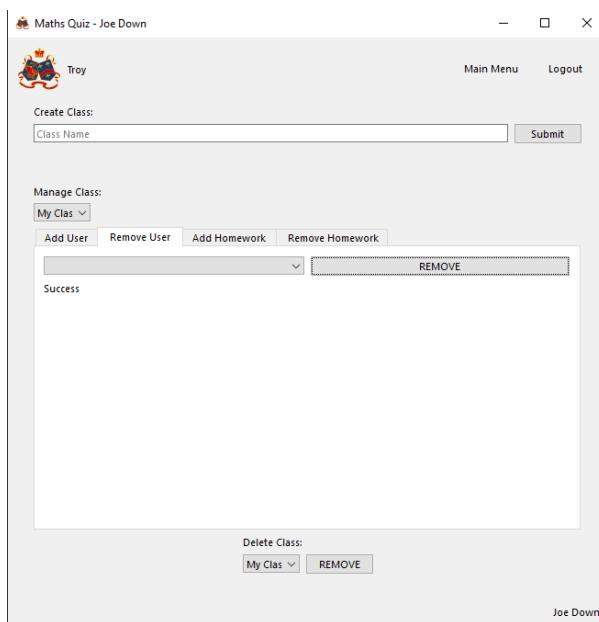
Manage Class: My Class

Add User Remove User Add Homework Remove Homework

tommyclair REMOVE

Delete Class: My Class REMOVE

Joe Down



Maths Quiz - Joe Down

Troy Main Menu Logout

Create Class: Class Name Submit

Manage Class: My Class

Add User Remove User Add Homework Remove Homework

tommyclair REMOVE Success

Delete Class: My Class REMOVE

Joe Down

5.1.1.2.14.2 Response from tester (teacher)

When I selected to remove the user from the class the program told me the removal was a success and the student stopped showing up as part of the class elsewhere in the UI

5.1.1.2.15 Add homework to class (SUCCESS)

5.1.1.2.15.1 Attempt

The screenshot shows the 'Maths Quiz - Joe Down' application window. At the top, there's a logo of a cartoon character named 'Troy'. Below it, the user name 'Troy' is displayed. On the right side of the header are 'Main Menu' and 'Logout' buttons. The main content area is titled 'Create Class:' with a 'Class Name' input field and a 'Submit' button. Below this is a 'Manage Class:' section with a dropdown menu set to 'My Clas'. There are four tabs: 'Add User', 'Remove User', 'Add Homework' (which is selected), and 'Remove Homework'. Under 'Add Homework', there are fields for 'Homework Name' (set to 'Homework') and 'Description' (set to 'Description'). A 'Due Date:' calendar shows the month of August 2019. The date '23' is highlighted in grey, indicating it is selected. At the bottom of this section is an 'Add Homework' button. Below the main form is a 'Delete Class:' section with a dropdown menu and a 'REMOVE' button. The status bar at the bottom right shows 'Joe Down'.

This screenshot shows the same application window as the previous one, but for the month of March 2019. The 'Due Date:' calendar displays the months from February to April. The date '30' in March is highlighted in grey. The rest of the interface is identical to the August screenshot, including the 'Add Homework' button and the 'Homework Added' message at the bottom.

5.1.1.2.15.2 Response from tester (teacher)

When I entered all details and submitted the program told me the homework was successfully added and it began to show up elsewhere in the UI

5.1.1.2.16 Remove homework from class (SUCCESS)

5.1.1.2.16.1 Attempt

The screenshot shows the 'Maths Quiz - Joe Down' application window. At the top, there's a user icon labeled 'Troy' and navigation links for 'Main Menu' and 'Logout'. Below this is a 'Create Class:' section with a 'Class Name' input field and a 'Submit' button. A 'Manage Class:' section follows, with a dropdown menu set to 'My Class'. Underneath are buttons for 'Add User', 'Remove User', 'Add Homework', and 'Remove Homework'. The 'Remove Homework' button is highlighted with a blue border. A dropdown menu under 'Add Homework' is open, showing 'Homework' and a 'REMOVE' button. At the bottom, there's a 'Delete Class:' section with a dropdown menu set to 'My Class' and a 'REMOVE' button.

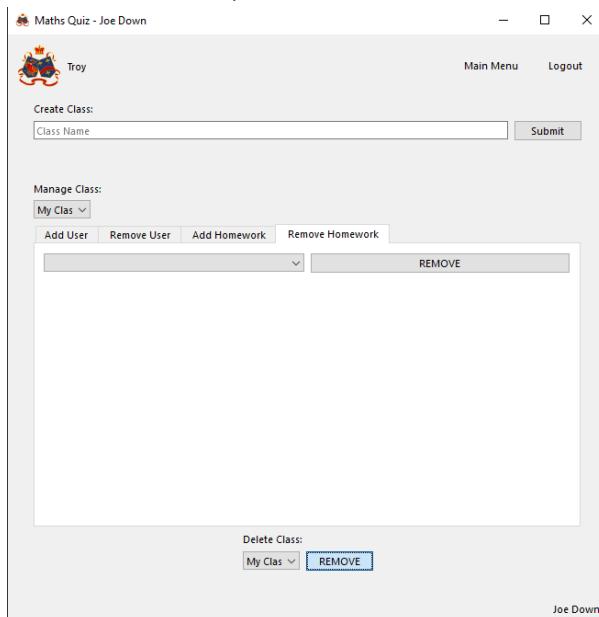
This screenshot shows the same application window after the homework removal. The 'Remove Homework' button is now grayed out, indicating it has been used. A message 'Homework removed' is displayed in the center of the screen. The rest of the interface remains the same as in the previous screenshot.

5.1.1.2.16.2 Response from tester (teacher)

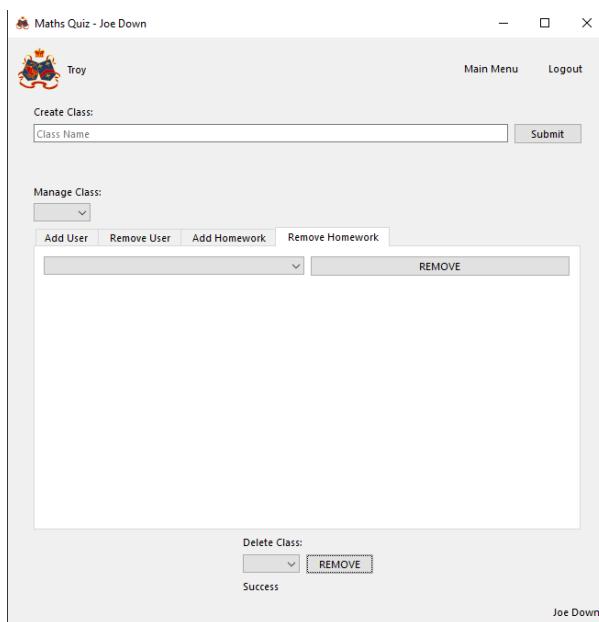
When I selected to remove the homework from the class the program told me the removal was a success and the homework stopped showing up as part of the class elsewhere in the UI

5.1.1.2.17 Delete class (SUCCESS)

5.1.1.2.17.1 Attempt



The screenshot shows the Maths Quiz application interface. At the top, there's a header with the title 'Maths Quiz - Joe Down', a user icon labeled 'Troy', and navigation links 'Main Menu' and 'Logout'. Below the header, there are two main sections: 'Create Class:' and 'Manage Class:'. Under 'Create Class:', there's a 'Class Name' input field and a 'Submit' button. Under 'Manage Class:', there's a dropdown menu set to 'My Class', followed by buttons for 'Add User', 'Remove User', 'Add Homework', and 'Remove Homework'. A large 'REMOVE' button is prominently displayed. At the bottom, there's a 'Delete Class:' section with a dropdown set to 'My Class' and another 'REMOVE' button. The status bar at the bottom right shows the name 'Joe Down'.



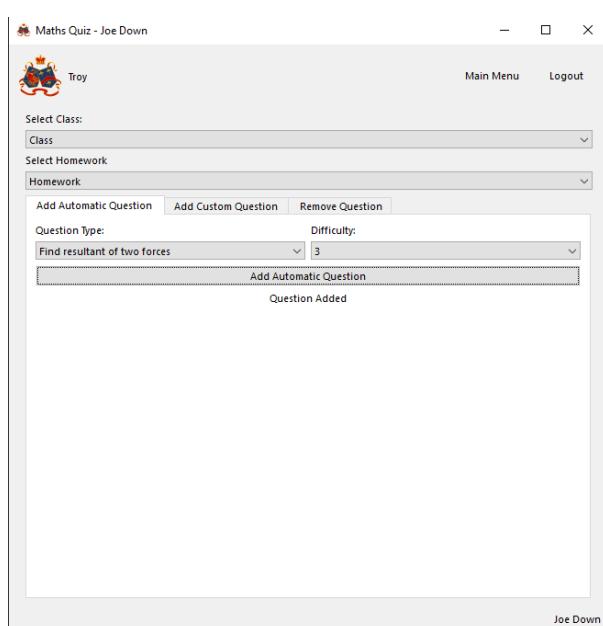
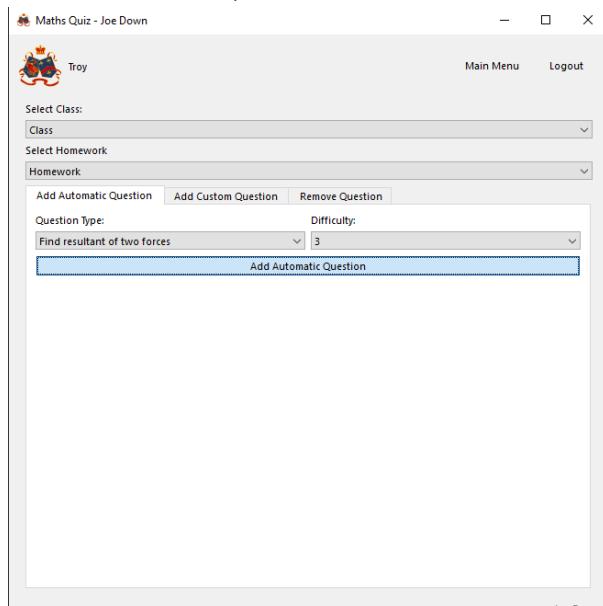
This screenshot is identical to the one above, showing the Maths Quiz application interface. The 'My Class' dropdown is selected, and the 'REMOVE' button is highlighted. However, a 'Success' message is now displayed below the 'Delete Class:' section, indicating that the removal was successful. The status bar at the bottom right shows the name 'Joe Down'.

5.1.1.2.17.2 Response from tester (teacher)

When I selected to remove the class, the program told me the removal was a success and the class stopped showing up elsewhere in the UI

5.1.1.2.18 Add automatic question to homework (SUCCESS)

5.1.1.2.18.1 Attempt



Checking new question is added to the class:

The screenshot shows a Windows application window titled "Maths Quiz - Joe Down". At the top right are "Main Menu" and "Logout" buttons. On the left, there's a user icon labeled "Troy". Below the title bar are dropdown menus for "Select Class" (set to "Class") and "Select Homework" (set to "Homework"). Under these are buttons for "Add Automatic Question", "Add Custom Question", and "Remove Question". A dropdown menu under "Add Custom Question" is open, showing the option "Find resultant of two forces". The main area contains a question text: "A force P acts on an object on a frictionless plane with magnitude 64N in the direction -14° to the x axis and another force, Q acts with magnitude 9N in the direction -126° to the x axis. Find the magnitude of the resultant force." Below the question is the answer "61.2" and a "REMOVE QUESTION" button.

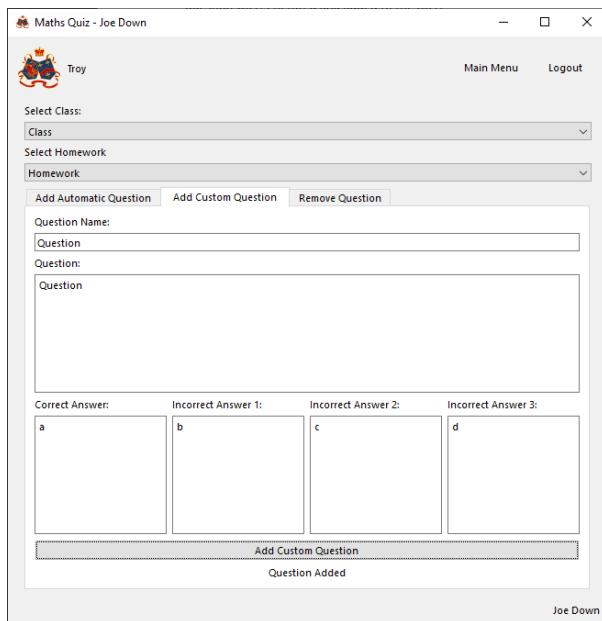
5.1.1.2.18.2 Response from tester (teacher)

After I selected the class and homework then selected a topic and difficulty to add an automatic question, when I clicked submit the program said adding the question was a success and a valid question was generated and added to the class

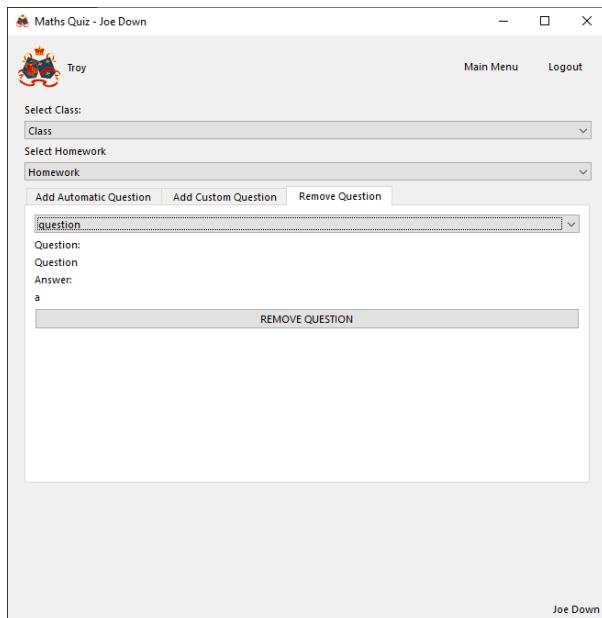
5.1.1.2.19 Add custom question to homework (SUCCESS)

5.1.1.2.19.1 Attempt

This screenshot shows the same application window as the previous one, but the "Add Custom Question" dropdown is now active, displaying the "Question Name:" field with "Question" typed into it. Below the question name is a large text area for the "Question:" which is currently empty. At the bottom of the screen, there are four input fields for "Correct Answer:" and "Incorrect Answer 1:", "Incorrect Answer 2:", and "Incorrect Answer 3:", each containing the letters "a", "b", "c", and "d" respectively. A blue "Add Custom Question" button is visible at the bottom of the screen.



Checking new question is added to the class:



5.1.1.2.19.2 Response from tester (teacher)

After I selected the class and homework then entered all the custom question details, when I clicked submit the program said adding the question was a success and the question was added to the class

5.1.1.2.20 Remove question from homework (SUCCESS)

5.1.1.2.20.1 Attempt

The screenshot shows a Windows application window titled "Maths Quiz - Joe Down". At the top right are "Main Menu" and "Logout" buttons. On the left, there's a user icon labeled "Troy". Below the title bar, a "Select Class:" dropdown is set to "Class". Under "Select Homework", "Homework" is selected. A toolbar at the top has three buttons: "Add Automatic Question", "Add Custom Question", and "Remove Question", with "Remove Question" being the active one. A dropdown menu is open, showing "question" as the current selection. Below it, the word "Question:" is followed by a text input field containing "Question". Under "Answer:", there is a text input field containing "a". At the bottom of the screen, the name "Joe Down" is visible.

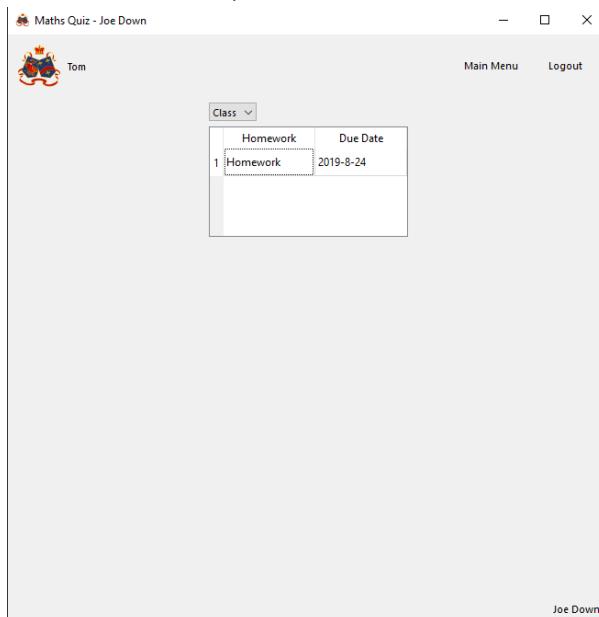
This screenshot shows the same application window after a question has been removed. The "Remove Question" button is still highlighted. The dropdown menu now shows "Find resultant of two forces" as the selected question. The "Question:" field contains the text: "A force P acts on an object on a frictionless plane with magnitude 64N in the direction -14° to the x axis and another force, Q acts with magnitude 9N in the direction -126° to the x axis. Find the magnitude of the resultant force." The "Answer:" field contains "61.2". At the bottom of the screen, the message "Question removed" is displayed.

5.1.1.2.20.2 Response from tester (teacher)

When I clicked to remove the question, the program told me removal was a success and the removed question stop showing up in the UI

5.1.1.2.21 Select homework to do (SUCCESS)

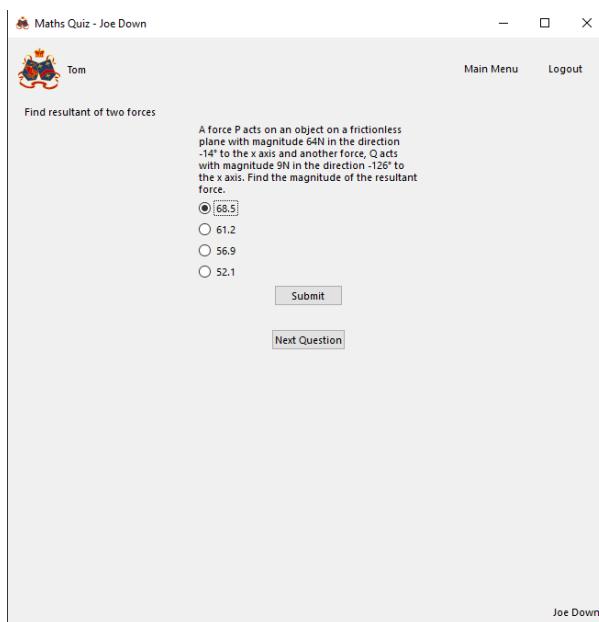
5.1.1.2.21.1 Attempt



The screenshot shows a Windows-style window titled "Maths Quiz - Joe Down". In the top left corner is a cartoon character icon labeled "Tom". The top right has "Main Menu" and "Logout" buttons. A dropdown menu "Class" is open, showing a table with one item:

Homework	Due Date
1 Homework	2019-08-24

The name "Joe Down" is visible at the bottom right of the window.



The screenshot shows a Windows-style window titled "Maths Quiz - Joe Down". In the top left corner is a cartoon character icon labeled "Tom". The top right has "Main Menu" and "Logout" buttons. The text "Find resultant of two forces" is displayed. Below it is a question: "A force P acts on an object on a frictionless plane with magnitude 65N in the direction -14° to the x axis and another force, Q acts with magnitude 9N in the direction -126° to the x axis. Find the magnitude of the resultant force." There are four radio button options: 59.5, 61.2, 56.9, and 52.1. A "Submit" button is at the bottom left, and a "Next Question" button is at the bottom right. The name "Joe Down" is visible at the bottom right of the window.

5.1.1.2.21.2 Response from tester (student)

I could select a class from the dropdown menu and the program showed me a list of programs and by double clicking a homework I was able to load up the test. Before selecting the homework, I was given the homework name and the date it was due on

5.1.1.2.22 Navigate quiz (SUCCESS)

5.1.1.2.22.1 Attempt

Next question clicked:

 Maths Quiz - Joe Down

Main Menu Logout

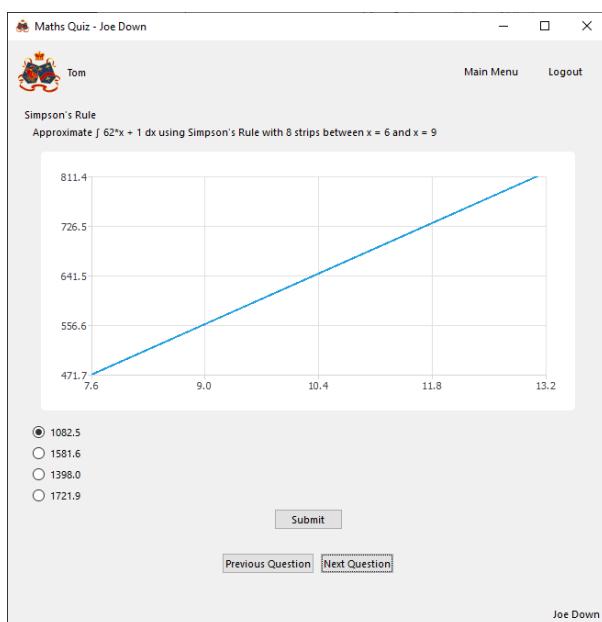
Find resultant of two forces

A force P acts on an object on a frictionless plane with magnitude 64N in the direction 14° to the x axis and another force, Q acts with magnitude 9N in the direction -126° to the x axis. Find the magnitude of the resultant force.

68.5
 61.2
 56.9
 52.1

[Next Question](#)

Joe Down



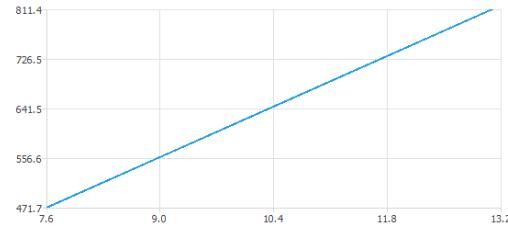
Previous question clicked:

Maths Quiz - Joe Down

 Tom

Main Menu Logout

Simpson's Rule
Approximate $\int 62x^2 + 1 dx$ using Simpson's Rule with 8 strips between $x = 6$ and $x = 9$



811.4
726.5
641.5
556.6
471.7 7.6 9.0 10.4 11.8 13.2

(1082.5) (1581.6) (1398.0) (1721.9)

[Previous Question](#) [Next Question](#)

Joe Down

Maths Quiz - Joe Down

 Tom

Main Menu Logout

Find resultant of two forces

A force P acts on an object on a frictionless plane with magnitude 64N in the direction -14° to the x axis and another force, Q acts with magnitude 9N in the direction -126° to the x axis. Find the magnitude of the resultant force.

(52.1) (68.5) (56.9) (61.2)

[Next Question](#)

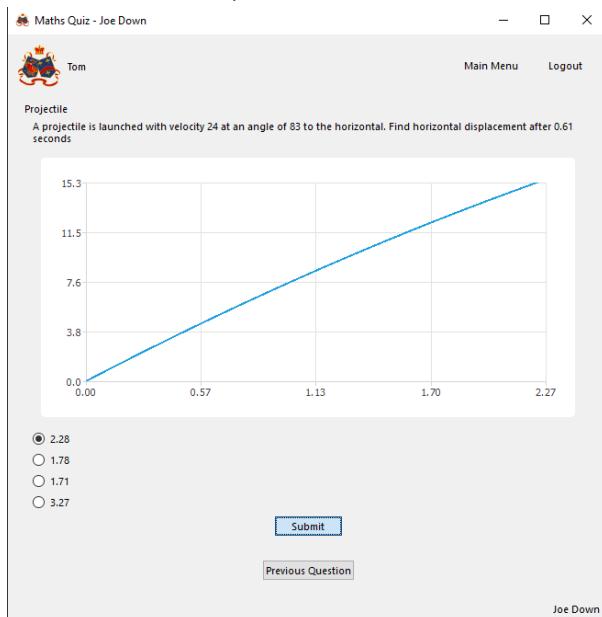
Joe Down

5.1.1.2.22.2 Response from tester (student)

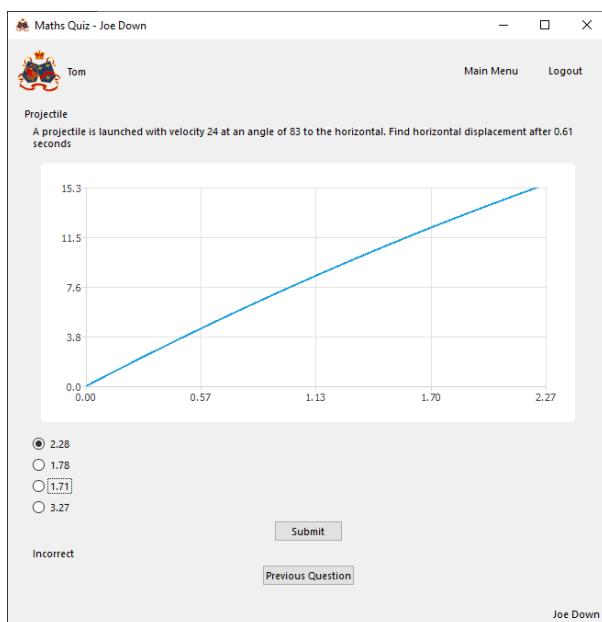
From any question in the quiz, clicking next question would load the next question and clicking previous question would go back to the previous one.

5.1.1.2.23 Question output (SUCCESS)

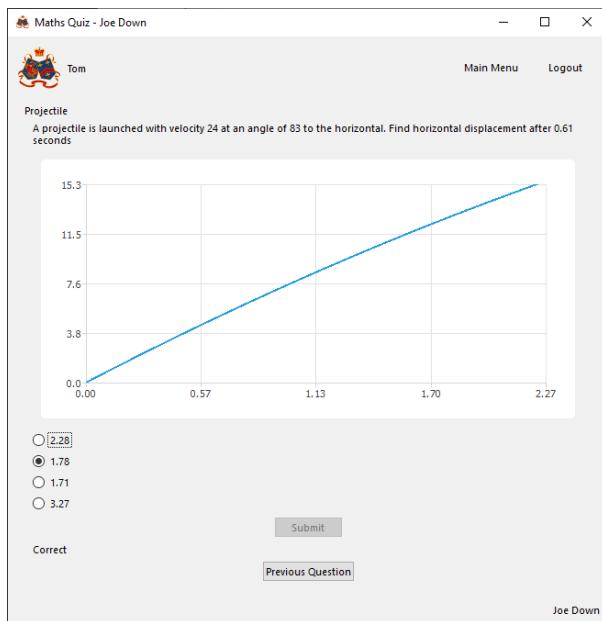
5.1.1.2.23.1 Attempt



Incorrect answer selected:



Correct answer selected:

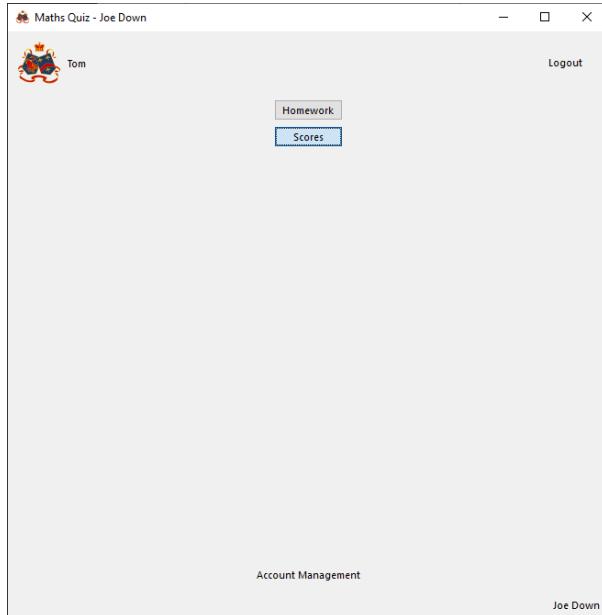


5.1.1.2.23.2 Response from tester (student)

The program showed me the question, four possible answers and in certain cases even a graph! When I selected to submit my response it correctly gave me feedback as to whether my answer was correct and stopped me being able to submit a new answer if I was correct

5.1.1.2.24 View previous homework results as student (SUCCESS)

5.1.1.2.24.1 Attempt



The screenshot shows a window titled "Maths Quiz - Joe Down". In the top left corner is a user icon labeled "Tom". On the right side are "Main Menu" and "Logout" buttons. Below the menu is a dropdown menu labeled "Class" with a dropdown arrow. A table displays one homework entry:

Homework	Score	Date
1 Homework	10%	2019-8-24

At the bottom right of the window is the name "Joe Down".

5.1.1.2.24.2 Response from tester (student)

I was successfully able to select to view results from the main menu and was presented with a screen which allowed me to select a class and have my homework scores displayed to me with the homework name and date

5.1.1.2.25 View class scores as teacher (SUCCESS)

5.1.1.2.25.1 Attempt

The screenshot shows a window titled "Maths Quiz - Joe Down". In the top left corner is a user icon labeled "Troy". On the right side are "Logout" and "Account Management" buttons. Below the menu is a vertical stack of buttons: "Class Management" (highlighted in grey), "Homework Management", and "Scores". At the very bottom of the window is the name "Joe Down".

The screenshot shows a window titled "Maths Quiz - Joe Down". At the top left is a logo of a cartoon character named "Troy". On the right are links for "Main Menu" and "Logout". Below the header, there are dropdown menus for "View Type" (set to "Class View"), "Class" (set to "Class"), and "Homework" (set to "Homework"). A table displays student data:

	First Name	Last Name	Score	Percentage	Attempts
1	tom	c	1	10	2

At the bottom right of the window is the name "Joe Down".

5.1.1.2.25.2 Response from tester (teacher)

I was successfully able to select to view results from the main menu and was presented with a screen which allowed me to select a view type (in this case I selected class view), class and homework and have all student scores for the given homework displayed to me with the student's first name, last name, raw score, percentage and number of attempts

5.1.1.2.26 View class scores as teacher (SUCCESS)

5.1.1.2.26.1 Attempt

The screenshot shows a window titled "Maths Quiz - Joe Down". At the top left is a logo of a cartoon character named "Troy". On the right are links for "Main Menu" and "Logout". Below the header, there are dropdown menus for "View Type" (set to "Student View"), "Class" (set to "Class"), and "Student" (set to "tommyclair"). A table displays student data:

	Homework	Score	Percentage	Attempts
1	Homework	1	10	2

At the bottom right of the window is the name "Joe Down".

5.1.1.2.26.2 Response from tester (teacher)

I was presented with a screen which allowed me to select a view type (in this case I selected student view), class and student and have all homework scores for the given student displayed to me with the homework name, raw score, percentage and number of attempts

5.1.2 Stakeholder response to finished program

Having used the program, the stakeholders were asked to give a final evaluation of the usability and functionality of the program and how useful they believe it is for them

5.1.2.1 *Troy Garvin, teacher of Mathematics*

For me, using the program was very intuitive. Everything was very responsive, and all the features of the program were clearly labelled with what they do. As a teacher it was relatively simple for me to create classes and homework and manage users and homework within the class. I also found the outputs relating to student scores to be enough for my needs as a teacher. The act of creating an account and logging in was also about as intuitive as it could be. A criticism I might make is that the user interface is, while not ugly, quite bland and generic and I think that it might be important for students to have a more attractive user interface to work with, even if functionality wise it is unchanged. The list of automatically generated questions was also quite limited; however, I understand that as a proof of concept which could be further developed upon given more time, the functionality to allow further topics to be integrated is clearly there.

With the addition of some more automatic topics I can very much see this as a tool I would use with my class as it is simple to use and reduces the load on me marking, as well as provides a large number of unique questions to really challenge students as not only are there an unlimited number of possible questions, there is also a scalable difficulty to allow me to tailor homework to the ability of my class in each topic.

In the design section it was stated that buttons with destructive features would be coloured red and this does not appear to be the case. Could this be addressed as I think it would further increase usability.

5.1.2.2 *Tommy Clair, Year 12 student currently studying A Level Maths and Further Maths*

I think this tool has the potential to be very useful. Yes, it doesn't cover quite a lot of the A-Level syllabus with the limited number of question types available, but the questions that are there exceed my expectations from the initial design (I think the inclusion of graphs for many of the mathematical function in questions are useful for me visualising the problem, particularly the one showing the path of a projectile). I found it particularly helpful that for each topic the actual content of the question would change, not just the values used, meaning the question I practised were less monotonous than I thought they could be as I wasn't always able to use the same method repeatedly.

Another thing I found useful was how easy it was to know how I'm doing. While doing quizzes I was immediately told whether I was right or not without having to check a mark scheme and see the real answer, so I was able to try again without the correct answer being spoiled and stopping me from working efficiently. After doing a homework it was also straightforward to navigate to a screen which would show me a very easily understandable breakdown of my scores in every homework I had done for each class.

While it's not a vital feature, and the program was very intuitive to use, I think it could have done with having a bit more of an aesthetic design as I think it would better entice students to want to use the program beyond obligation to do homework and make it stand out more from other options.

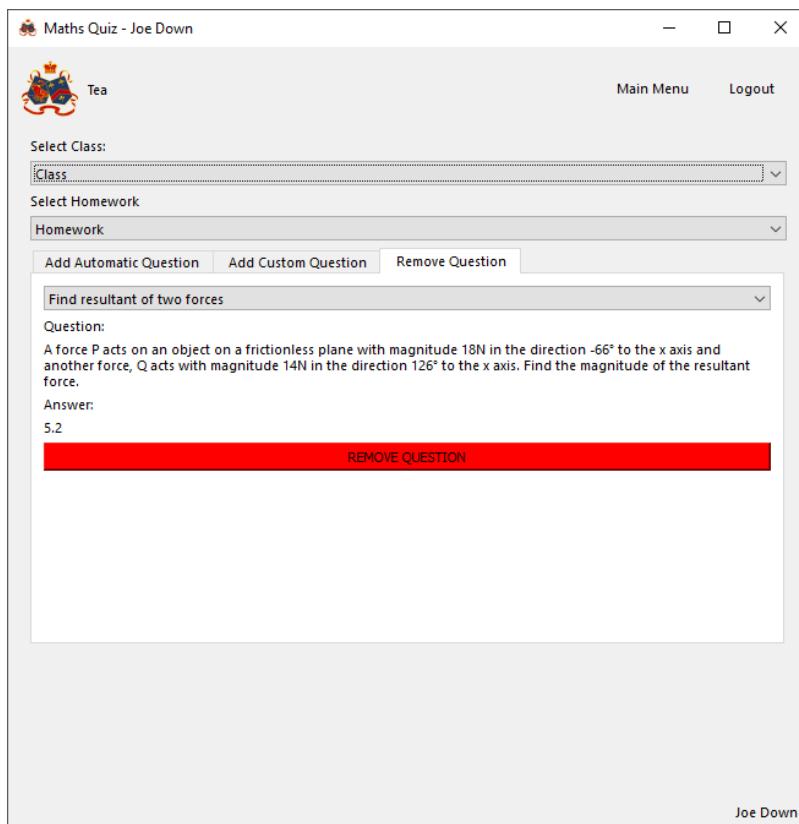
5.1.2.3 *Analysis of responses*

From the stakeholder responses to the program the program clearly largely satisfies and, in some areas, even exceeds expectations. This is with regards to the ease of navigation, taking quizzes,

viewing scores and receiving any feedback related to any program functions. The addition to include graphs with questions seems to be a particularly strong adaptation from the initial design from the student side of the program as it helps give a better understanding of what a question is asking for and how the situation, such as the motion of a projectile, works. It is also apparent that the program could do with being able to generate a greater range of question types and this will be discussed further in the further development section.

Stakeholder responses will be used below, along with my own testing, to decide to what extent success criteria have been met below.

With regards to the lack of red buttons from the design in the final program have now been addressed by simply changing the colour property of the relevant buttons:



Maths Quiz - Joe Down

 Tea Main Menu Logout

Create Class:

Manage Class:
Class

Delete Class:
Class

Joe Down

Maths Quiz - Joe Down

 Tea Main Menu Logout

Create Class:

Manage Class:
Class

Delete Class:
Class

Joe Down

5.2 LEVEL OF SUCCESS

5.2.1 Success Criteria Summary

Success criteria will be checked against stakeholder responses to the program and final testing done above and labelled with whether they have been achieved. Where not achieved or only partially achieved, further detail will be given.

5.2.1.1 General

Criteria	Evidence of success	Criteria met?
User can create account	User details (username, password, name etc.) are correctly and permanently stored in a database	Yes
User can login with account details	User login details can be correctly compared to those in database and matching user access given if login details are matching	Yes
User can logout	Logout button can be pressed and return to login screen with access to the rest of the program revoked	Yes
User can update account details	Non-identifying user details (i.e. All details except for username and id) stored in database can be changed and amended values permanently stored without confliction	Yes
User details are store securely	Password is stored in a correctly salted and hashed form	Yes
Enough usability features are in place	Features are sufficiently labelled and features such as buttons or dropdown menus etc. are used as appropriate. Destructive features such as deleting information is made clear to not take lightly (e.g. buttons made red to alert user)	Yes
Navigation features	Ability to navigate to key pages such as the main menu	Yes

All features contained within a graphical user interface	All features of the program can be accessed through windows with user typing kept to a minimum and at no point requiring interaction with a terminal	Yes
Database integration to permanently store data	All data is retained when the program is restarted and structured in such a way that data can be accessed, and new data added as needed	Yes
Features should be all accessible as required from a main menu	Main menu through which all pages are reachable by navigation though a minimal number of other pages	Yes
Permission system	Access to data should be restricted to only those it relates to (i.e. teachers can only see their details and student details in their classes and students should be unable to access teacher or other student data). Program features should be limited to only the type of user they apply to (only teachers can make and manage classes and only students can be set homework)	Yes

5.2.1.2 Teacher

Criteria	Evidence of success	Criteria met?
Teacher can create class	Class details (Name, teacher, students, homework etc.) are correctly and permanently stored in a database	Yes
Teacher can add student to class	Class student relationship is correctly and permanently stored in database	Yes

Teacher can remove student from class	Class student relationship is correctly and permanently removed from database	Yes
Teacher can add homework to class	Class homework relationship is correctly and permanently stored in database with relevant homework details (name, questions etc.)	Yes
Teacher can remove homework from class	Class homework relationship is correctly and permanently removed from database	Yes
Teacher can delete class	Class details are correctly and permanently removed from database	Yes
Teacher can add custom question to homework	Question homework relationship is correctly and permanently stored in database with relevant question details (name, question text, correct answer, incorrect answers etc.)	Yes
Teacher can add automatically generated question to homework	Question homework relationship is correctly and permanently stored in database with relevant question details (name, question text, correct answer, incorrect answers etc.)	Yes
Teacher can select question difficulty and question adjusts as needed	Question difficult is correctly stored with question in database and question generation criteria are satisfied (see below)	Partially While all question types adjust depending on the difficulty selected to produce harder questions, in some cases the scope of this is limited to the same question type simply using larger numbers or increasing the number of terms in mathematical function rather than the question itself asking for something more

		complex to be found compared to lower difficulties
Teacher can remove question from homework	Question homework relationship is correctly and permanently removed from database with relevant question details	Yes
Teacher can view results for each student for a given homework	For each student, question results for a given homework and student are retrieved from the database and a score calculated from the number of correct responses	Yes
Teacher can view results for each homework for a given student	Question results for each homework in the class and a given student are retrieved from the database and a score calculated from the number of correct responses	Yes
Scores correctly calculated	Score calculated as the sum of each correct response divided by the number of times that question was attempted, all divided by the number of questions and multiplied by 100 to give a percentage	Yes
Database usability	Database should be structured in such a way that it is fast to insert and find relevant data without the need for superfluous extra processing Database relations should allow for the introduction of new tables or introduction of new types of relationship between tables to allow for potential future maintainability and	Partially Database is well structured and easily usable within the program's code. However, it would have been better to anticipate the inclusion of graphs with questions when first creating the database a new graphs table would not have needed to be created

	updates without extensive restructuring	
Validation	All inputs should be validated to be of a valid form and usable for their intended purposes	Yes

5.2.1.3 Student

Criteria	Evidence of success	Criteria met?
Student can access homework set for a given class	Program can output a list of homework from the database for the given class and launches the quiz if a homework is selected	Yes
Student can only do homework before due date	Program excludes student from accessing quiz past stored due date	Yes
Quiz gives question and allows choice and submission of multiple-choice response	Question correct status is correctly changed to True in database for given user and question only if the correct response was selected and submitted	Yes
Student can see results for each homework in a class by class basis	Question results for each homework in the class and the current student are retrieved from the database and a score calculated from the number of correct responses	Yes
Score correctly calculated	Score calculated as the sum of each correct response divided by the number of times that question was attempted, all divided by the number of questions and multiplied by 100 to give a percentage	Yes

5.2.1.4 Questions

Criteria	Evidence of success	Criteria met?
Enough topics covered	Topics that can be generated cover multiple different topics	Partially Topics covered are a mixture of both pure

		and mechanics, with some from Further Maths and others Maths. However, much of the syllabus is not accounted for in any way other than if a teacher creates a custom question and so given more development time more question generation frameworks would be written
Generated correct answers are correct	When manually calculating answers to generated questions, the answer found matches the one generated	Yes
Generated incorrect answers are unique and incorrect	When manually calculating answers to generated questions, the answer found does not match any of the ones generated and each incorrect answer is different	Yes
Questions are relevant to the mathematics specification	Topics covered are all within the Edexcel mathematics specification	Yes

5.2.2 Evaluation of success criteria

Overall most success criteria have been successfully accomplished, with many more features added on top. Areas of only partial completion can all be attributed to time constraints and have still been demonstrated in some capacity in the program, even if not in all instances where they could be demonstrated (while not all question types can be generated the framework to integrate new types of question is very strong). As no success criteria have been completely failed to be met it can therefore be concluded that the program development has been a success. In addition to this success in terms of functionality, usability and robustness testing carried out and the stakeholder responses indicate that all features have been implemented efficiently and intuitively, further indicating a useful and usable final product.

5.3 MAINTENANCE AND FURTHER DEVELOPMENT

5.3.1 Current program limitations

Limitation	Consequence
Database is stored locally	Teachers and students must use the same installation of the

	software on the same system or manually transfer updated versions of the database, meaning students cannot easily take work set away from school and only one student can work at a time.
Not all topics on syllabus covered	Program is only useful in its current state for revision of a very narrow selection of topics spread out between Maths and Further Maths syllabuses
Platform availability	Executable can only run on windows and so users of Linux or MacOS machines cannot use the program in its current state

5.3.2 Summary of further development and maintenance

5.3.2.1 *Range of questions*

The main limitation of the program is clearly that there are many topics in the A-Level syllabus which there was not time to develop question generators for. However, the relatively little code required to integrate these generators into the main code of the quiz program itself indicate that a framework has been developed of sufficient quality that, given further development time, it would not be infeasible to code these functions and implement them into the quiz program as it currently stands without impacting any of the other functions of the quiz. This is because all question generation scripts are kept in a library separate from the main program, with all final generated questions conforming to the Question class before being passed to the main program, ensuring they will always be handled the same way from this point, regardless of topic.

5.3.2.2 *User interface*

Further developments could also be made regarding the UI. While functionality wise everything works as intended it is still quite visually bland and so time could be spent implementing backgrounds and/or a colour scheme to the program, as well as manipulating the properties of each widget such as buttons to give the program a more unique visual identity.

5.3.2.3 *Usability*

Alongside visual improvements to the UI, changes could also be made to further increase usability. Firstly, simply making the UI more aesthetically pleasing as described above should make it easier for a user to use the program as a clear and consistent design language is memorable and helps indicate what various interactions better from the context of other actions in the program. Further features for usability in further development could also be the implementation of features to accommodate for disabilities, such as a high contrast mode which makes darker colours darker and lighter colours lighter or a large text mode to accommodate for users with visual impairments. Meta tags could also be included with each element on a page in order to allow a screen reader to inform a user what is on the page and what they have selected by audio.

In its current state the program is also limited only to the English language, so will be unusable in many places around the world. This could be improved by working with a speaker of both English and another language to implement translations of all text in the program. A menu could then be

implemented to set which language the program should be displayed in and methods added which redefine what is placed in every text output in the program to match this. Language selection could be made clear by placing a flag next to each language to indicate what it is. No significant changes would need to be made to the workings of the program as maths is a near universal language, with only certain small regional changes being required for notation, such as substituting a . for a , when defining a decimal for mainland European users.

5.3.2.4 Database structure

Another aspect that could be reworked is the database structure now that it is clearer exactly how the program is structured and where further development would be headed. The most notable change would be integrated the data stored in the graphs table into the questions table to reduce the amount of data searching required. This would however potentially require a large amount of reworking and retesting any code which utilises either of these two tables.

5.3.2.5 Other platforms

PyQt is also supported on MacOS and Linux. This means that, given access to the platforms (which I do not currently have) a bundled program of the correct type (the generated windows exe is incompatible with either of these OSes) could be generated from my code and made to run on that platform. This would increase accessibility of the program for those using other platforms, though further testing would need to be done to check for platform specific errors.

5.3.2.6 Database accessibility

To address the limitation that the database is stored locally, and teachers and students must therefore use the same machine, work could be done to have the database stored on an external, internet accessible, server to which the program could be adapted to connect to as a client from any internet connected system at any location. This would however require the cost of setting up the server, as well as extensive changes to the code used to read and write to the database as direct database access will not be available to the program, only the ability to request data from the server or for the server to change data it has stored. Relevant and extensive permission systems would need to be put in place server side.

5.3.2.7 Commenting

Finally, the coded solution has been heavily commented and meaningful naming used in order to ensure that the way code works and what sections of code do can be more easily understood if development were to recommence in the future. This means development of new features could begin more quickly and existing functions could more easily be utilised.

5.3.2.8 Version control usage

Additionally, git has been used for version control of the program, allowing the easy tracking of changes made to the program and allowing changes to be reverted if errors are created in later development. This further increases maintainability as it is made easier to identify when errors were introduced. It also allows for the branching in development to prototype new features which can be later merged back into the master branch only once fully tested, without interfering with further development of the main branch itself until ready. Changes could also be pushed and pulled from GitHub by other developers if permission were granted, allowing the potential to increase the scope of further development by allowing additional developers to work on different parts of the program and easily share changes.