QUIZO

AN EDUCATIONAL TRIVIA APP

MATTHEW WOODING

CANDIDATE NUMBER: 2487

CENTRE NUMBER: 52423

CONTENTS

ANALYSIS

THE PROBLEM

As we progress through the technology era, teaching is evolving, and iPads and laptops are making their way into more and more classrooms. However, as fast as the penetration of hardware into education is, there is still the necessity for more universal software to aid teachers in the teaching of students. Certain applications such as *Showbie* can be useful for teachers to interact with students, but the need for tutorials on YouTube show the high level of inaccessibility the program has. Other applications such as *MyMaths* are simple and easy to use, but are restricted to just one subject. The best way to implement an educational program into schools, that is appealing and useful for both students and teachers, is one which has a user-friendly interface, and has the scalability to be expanded across multiple subjects. This is the problem which my education trivia application, *Quizo*, aims to fix.

INITIAL DESCRIPTION

Quizo is an educational application for teachers and students of all stages of education, where students can compete against themselves or friends to reach high scores in sets of questions from a wide range of topics. These questions can either come from the pre-made sets, or teachers can login to the program to add their own question sets to the program for students to use.

The game has 2 play settings; The first mode is multiple choice, where each question will have 4 different options with only one being correct. The other mode will have no guidance for users, and they will have to type their answers in. In both modes, points will be awarded for speed and accuracy: the faster you answer the more points you get, but if you get a wrong answer, your points will

be taken away from your score rather than added. At the end of a set of questions, the users score will be displayed.

WHY COMPUTATIONAL METHODS WILL SOLVE THE PROBLEM

As stated in the problem, the program I am creating needs to be a tool for teachers and students, where they can access a wide database of questions spanning all topics. Using SQL, a large database can be built and added to over time, providing users with all the questions they could ever want. Computational methods such as being able to search through all the data easily also makes this problem suitable to a computational solution as it saves time compared to having to sift through the internet and textbooks to find good questions and revision tools.

Additionally, when taking tests and learning new topics, being able to track your progress over time is a very important part of the educational process, and with a program that can record all your scores and show you graphs and data based on results, this is a more visually appealing and accessible way to show progress. Having to make students take tests in class, having them each marked by the teacher, and then keeping a log of all the results would take more time and effort while providing the same results as using a computer system, making this part of the problem suitable to computational methods.

Having a login system, providing each user with their own account is helpful for allowing teachers to track their students' progress as they will be able to view progress for each student sorted by a certain test, and could use these progress reports to help in areas students are weaker in. Being a computer program, it will be easier for teachers to adapt quizzes to help students in the weaker areas, rather than having to write up whole new tests each time there is new content to add.

The main use of this program is to provide students an environment where they can revise topics and improve their knowledge, while also providing teachers with the tools to track how their students are progressing, and be able to help them where necessary. The program will need to output tracking sheets, which can be created using lists of a user's previous scores, and allow the teachers the

ability to view them, by being able to group students that a teacher teaches. More complex output like graphs and leaderboards can be created using graphics libraries, which can encourage competition between students to reach the higher scores and improve their knowledge.

Any more in-depth output such as a grading system or being able to see every question a student has gotten wrong, would not give nearly the same desire to improve their knowledge as a simple graph, and so is unnecessary to create. While it is useful to create the program using computational methods and take advantage of this fact, adding small unnecessary details such as grouping students by year group and form, would take development time away from the more important parts of the program that will make it a useful tool for schools.

The program that I will be creating is clearly suited to a computational solution, and even though not being the only way to solve this problem, is a far more efficient and intuitive way to do so when compared to setting tests during lessons and marking each individually, taking up a lot of time, and not providing the immediate feedback that my program will be able to do.

I plan to develop this program as a prototype, fleshing out the basis of my initial idea and building what I believe to be a working solution by the end of my development. Seeing as my target audience for this application is students and teachers of schools, I hope to take my prototype and demonstrate its functionality to local schools and see what they think of the idea, taking on board any changes they feel could be made to improve the solution, and make it a better product for their students. I have chosen to develop the program as a prototype for this reason, because being able to follow a spiral model allows for changes to be constantly made according to feedback from my stakeholders and incorporated into newer iterations of the program.

THE END-USERS

As stated above, the target audience for my program is teachers and students of schools. Although I have decided to use the spiral model for developing prototypes of my program, I decided that I would need an initial stakeholder in order to point me in the right direction for areas where I had less than ideal knowledge of how a school revision application should be run.

After reaching out to a few schools in my local area, I got a response from the head of Computer Science at the Royal Grammar School, High Wycombe.

Established in 1562 and granted a royal charter upon its inception, the Royal Grammar School has a long history of providing its' students with an outstanding education. It is this education, paired with excellent examination results that makes the school one of the top sixty state schools in the country. Since its humble beginnings in the sixteenth century, the school is now home to more than 1300 students and over 100 members of staff.

Having heard back from the head of Computer Science, Mr James Hamm, he proceeded to talk to me about my idea for how I could help improve his student's revision techniques. After a lengthy conversation about the prospect of my program, he agreed to let me interview him as well as a few of his students to see if we could make my program a possible application used by the students of the school.

Below are the initial interviews I carried out with Mr Hamm and his students.

FINLAY FRANKS (STUDENT OF THE RGS) – FIRST INTERVIEW:

Q: What stage of your education are you currently in?

A: I'm currently in my final year of sixth form, studying Maths, Further Maths, and Physics.

Q: Have you ever used an educational application to aid your learning, and if so what form did it take?

A: Throughout my school life, I have taken advantage of many applications such as *MyMaths* and *Code Academy*, which both have lessons and

quizzes to improve my knowledge, which I found were very helpful.

Q: When taking tests, do you find that being multiple choice makes it easier or harder, or does it not make a difference to you?

A: Personally I find that multiple choice is slightly easier than having to come up with answers on the spot, because if I see the answer written out amongst incorrect choices, I can normally recall what the correct choice is.

Q: Would having some form of physical printout feel rewarding to you when completing quizzes, or just an unnecessary addition?

A: Seeing as everything is digital now, having a printout doesn't seem too appealing to me, however some way of tracking my progress would be something I'm interested in.

Q: Would competition against your class mates within the program encourage you to try harder in quizzes?

A: Definitely. Any chance for me to compete against my friends will make me want to try harder as being at a school like RGS encourages me to be the best I can be.

Q: Would being able to stop a quiz midway so you can continue it later be something that would appeal to you, or do you prefer to just take the quiz all at once?

A: It would be nice to be able to pause midway in case I'm needed somewhere else or if I don't have enough time to finish the whole quiz in one sitting, however there would need to be limitations to prevent cheating, like maybe only being able to pause a few times before an anti-cheat system is used.

Q: Are there any specific features you would be looking for in an educational trivia application?

A: Having a program that is applicable to all subjects would be great as it would encourage a wider user-base resulting in a larger question-base so that I never exhaust questions related to topics that I want to improve in.

Q: What stage of your education are you currently in?

A: I'm currently in my first year of sixth form, studying Maths, Physics, and Biology.

Q: Have you ever used an educational application to aid your learning, and if so what form did it take?

A: I've used an application called *Memrise* before, which I used for my GCSEs like Spanish, but it got boring quite quickly because there weren't, many courses on it for me to take.

Q: When taking tests, do you find that being multiple choice makes it easier or harder, or does it not make a difference to you?

A: It depends on the topic of the questions, and if I'm comfortable with the topic I can normally just remember answers, however if I'm learning new material, I find multiple choice to be easier.

Q: Would having some form of physical printout feel rewarding to you when completing quizzes, or just an unnecessary addition?

A: No I wouldn't because it's a waste of paper, and I'd end up just losing it.

Q: Would competition against your class mates within the program encourage you to try harder in quizzes?

A: Yes it would make me try harder because I want to win and prove that I'm the best.

Q: Would being able to stop a quiz midway so you can continue it later be something that would appeal to you, or do you prefer to just take the quiz all at once?

A: If I'm in a rush then being able to pause it would be beneficial to me, but normally I'd prefer to just get it all done at once.

Q: Are there any specific features you would be looking for in an educational trivia application?

A: The application needs to look appealing so that it can be more

engaging and will make me more inclined to do it; It would make it feel less like work and would encourage me to use it more.

JAMES HAMM (HEAD OF COMPUTER SCIENCE AT THE RGS) – FIRST INTERVIEW:

Q: Have you ever used an educational application to aid your teaching, and if so what form did it take?

A: Currently we use our VLE where we can upload homework assignments for students, which is useful for lengthy pieces of work, but for tests it's quite a bloated system.

Q: When setting tests, do you find that being multiple choice makes it easier or harder for your students, or does it not make a difference?

A: I tend to find that when I set tests for my students, multiple choice questions are more easily answered than others.

Q: Would having a way of tracking each of your students be useful in terms of being able to give additional help?

A: Being able to see which of my students may need additional help with certain topics would be very useful to me so that we can work on those areas more. Having a way of seeing how students are doing in relation to each other, like a leaderboard would also be useful and probably prompt a bit of competition and encourage trying to do as many tests as possible as well as they can, and being able to print something like that and sticking it on a board in the classroom could help.

Q: As a teacher, would you also like to be able to attempt your quizzes, or is that functionality not necessary for you?

A: I think being able to attempt my quizzes to make sure they are as difficult or easy as I need them to be for my students is a very necessary feature.

Q: Would you like some way of checking whether your students have taken a certain test, or will you use the application as a type of additional work to homework?

A: I would use this program more for additional work for my students as opposed to homework, so I feel like I wouldn't need to check whether all my students have attempted it, but rather just know how my students who have attempted it are coping.

Q: Are there any specific features you would be looking for in an educational trivia application?

A: Students forgetting their passwords seems to be a big issue with our school computer system, so in lieu of that I think it would be useful for there to be a way to reset passwords in case students forget them. I also feel that being able to edit my tests after having written them would be useful in case I've written something incorrectly or if it's too difficult.

RESULTS OF INTERVIEWS

From my interviews I have found some new information that I will use for the development of my program:

- Multiple choice questions are in general seen to be easier than others, therefore I will call the multiple choice section "Easy Mode" and the typed in section "Hard Mode"
- Having printouts of results isn't useful for my stakeholders, but having printouts of leaderboards could be useful
- Competition is important, so all scores will be added to a leader board for all users to view
- Being able to take breaks is a good idea, but will need an anti-cheat system to prevent exploitation
- The program will need a large number of question sets for an initial install base, in order to encourage teachers to write more questions
- Teachers also need the ability to attempt quizzes
- Being able to reset passwords for users is a very necessary function
- Quizzes need to be able to be updated easily by teachers

Having now conducted some initial interviews with some members of the school, I will need to do a bit more research into how I could tackle the program before I can expect an agreement to be made.

RESEARCHING EXISTING APPLICATIONS

So that I can develop my program to be as fully featured as I can, I will research existing applications similar to my idea so that I can look at features that may or may not be useful for my application. I will be researching *Showbie*, and *Sporcle*.

SHOWBIE

Showbie is an application used by schools to create, collect, and mark students' homework online. It enables students to access portals to each of their classes so that they can do their work from their laptop or iPad. This application is already in use by some of the teachers at the RGS, however is more of a homework related approach to digital education than my end-users are after. However, it does have some functions that could be beneficial for my program which I have outlined below.

Feature	How I can adapt it for my program
Easily search for and track progress of students	I can use similar tracking to create visual representations for progress (graphs, scorecards, etc)
Uploading homework	The system for uploading homework seems fairly simple for this program, and is something I could adapt for uploading questions to my program's database to make it user friendly
Registering Students	In Showbie, to register new students they require a "Class Code", which would make it easy for students to add themselves to classes in my program too. And then if they are in more classes, teachers can enrol pre-existing students to their class

Showbie is an online program that can be accessed from multiple devices, however as I didn't ask about hardware requirements in my initial interviews, I will take this concept back to the head of computer science at RGS and see how he responds.

SPORCLE

Sporcle is an online quiz application, where anyone can sign up with an account and start creating quizzes for anyone to try. The website is free for anyone to use, however is not limited to "educational" material, as anyone can make a quiz about anything. Some features that I found that could be adapted for my program are listed below.

Feature	How I can adapt it for my program
Scoring System	*Sporcle* uses both a score and a time for its leader boards; I can adapt this to form a combination of the two for my program
Grouping & Searching	I can use similar grouping and searching functions from *Sporcle* in my program, like having question sets be categorised by topic or subject, and then having a search bar for if people are looking for something specific
Random Quiz	The application has a random quiz button, which is possibly a bit too broad for my application, but perhaps having a randomizer within subjects may be an interesting feature for my program

This application is also an online one that can be accessed from multiple devices, however it also enables anyone to create quizzes, rather than a specific group, such as teachers. I will need to bring this up in my next interviews to see whether my end-users want to let both students and teachers write questions.

RESULTS OF RESEARCH

From my research of both *Showbie* and *Sporcle*, I have found some new features that I may want to adapt for my program, which I will need to ask my end-users about before I begin to write up my objectives to see exactly how they would like the program to function. I will be going back to my original 3 interviewees for my follow-up questions.

FOLLOW-UP INTERVIEWS

FINLAY FRANKS (STUDENT OF THE RGS) – SECOND INTERVIEW:

Q: Having researched some pre-existing trivia applications since our last talk, I have found some features that I wanted to talk through with you and get your thoughts on. You've already told me that you would like some way of tracking your progress; would visual representations of this such as graphs be useful to you?

A: Some sort of graph could be really helpful for me to be able to track my progress. Perhaps something like a line graph which shows how I've improved over time would be possible, because then I could show my parents that I'm improving without having to go through all my marks. A simple and quick representation of my progress would be really helpful.

Q: When first using a program for school, would you rather have everything registered for you, ready to use on first launch, or would you rather be able to register yourself?

A: I think having everything registered ready for me to go would be better for students, as there are probably some people that if they had to register themselves to take the quizzes, might be put off by the program. However, I would be concerned about my security if a password is generated for me. Maybe if there was a way for me to change my password the first time I logon so I won't forget it.

Q: When you're taking a test, do you find the best way to be scored is by correct answers, or time, or a mixture of the two?

A: I think the main focus of a test is to get the answers correct, so if I had to pick between the two it would be that. On the other hand I think it's good practise to get quick when answering questions so having a scoring system that accounts for both would be good, but perhaps with more emphasis of actually getting the answers correct.

Q: Would being able to just take a randomly selected quiz be something you're interested in, or is that too broad of a function for it to be of any use to you?

A: I think that selecting a random quiz from all of them would be a bit useless compared to doing the quizzes normally, as the odds of me getting a quiz on a topic I actually know something about would get smaller and smaller as more quizzes enter the system.

Q: Is making quizzes yourself something you would want to do, or would you rather just keep to doing the ones made by your teachers?

A: I think there is probably too much room for exploitation if you let students make tests as well, it's probably for the best if only the teachers can write them.

SEB PERRY (STUDENT OF THE RGS) – SECOND INTERVIEW:

Q: You mentioned that you wanted to use a program that wasn't boring and that looked interesting; would having visual representations of your results such as graphs be an interesting feature to you?

A: Yes, especially a line graph or a pie chart as they're just easy to read data from, and it would add an interesting visual element to the program that makes it more interactive.

Q: When first using a program for school, would you rather have everything registered for you, ready to use on first launch, or would you rather be able to register yourself?

A: Personally I would rather be able to register myself into the system, only so that I'd be able to set my own password as I feel my data would be more secure than if I was given a password from my teachers.

Q: When you're taking a test, do you find the best way to be scored is by correct answers, or time, or a mixture of the two?

A: I think that getting the answers correct is more important, but being able to judge people based on time would be good as well so that there is more competition between me and my friends, because otherwise we could all just get it all correct, but then there would be no way to improve on that.

Q: Would being able to just take a randomly selected quiz be something you're interested in, or is that too broad of a function for it to be of any use to you?

A: I think I would definitely use that function because it would be fun to try new topics and trying to learn new material on my own.

Q: Is making quizzes yourself something you would want to do, or would you rather just keep to doing the ones made by your teachers?

A: I only want to do the ones that the teachers set because I know that all the tests will be of good quality, whereas if students started making them I feel like they wouldn't be able to be vetted to see if they were useful to other students.

JAMES HAMM (HEAD OF COMPUTER SCIENCE AT THE RGS) – SECOND INTERVIEW:

Q: I looked at a few online trivia applications for research, so I was wondering whether the program that you want me to develop for you needs to be an online solution, and what sort of devices I need to optimize the program for.

A: My vision for the program is that it will be used on desktop computers and laptops. Although we do use iPads in school now, I think it would be wise to create a PC solution before looking into any other form of device. In terms of networking, we want students and teachers to be able to use the program wherever they are, be that at school or home. We don't want to have to restrict anyone as we want as many people to use the program as possible.

Q: For an initial prototype, would you be happy to accept a program that is local only so that it can be tested on your school's local network before including cloud functionality for students that wish to use their laptops or desktops at home?

A: I think if removing cloud functionality from the program for the time being makes the turnaround time of the project shorter, and allow us to test the system thoroughly first, then I am more than happy to wait on cloud functionality until a later date.

Q: So as a response to that, would having the program pull the database from

the cloud when it's required be a suitable solution in your opinion?

A: I think at the moment that seems like a reasonable idea, having the program be the client and have the database hosted in the cloud. I'll leave the details of how that will work up to you, but as long as the data is secure, i.e. the information of students and teachers and their passwords.

Q: Would you rather have all your students be registered by their respective teachers, or just have students sign up when they want to use the program?

A: I think we are more likely to see a greater number of students use the program if they are able to just login and use the program without the hassle of registering. As long as it is a simple process for the teachers, I see no reason why having them register their students should be a problem.

RESULTS OF INTERVIEWS

From these follow-up interviews I can see that the students will all have their own preferences and that I will need to try be as accommodating as possible, for instance the random quiz button. I'll add it as a function for my program for anyone that wants to use it, but if people don't want to, they just don't have to touch it.

I have also found from my interviews that the students would like both answers correct and time to be included in the program's scoring system, so I will create a function that decides points based on how fast you are to answer, and if you get it correct you gain the points, and if you get it wrong, you lose them.

From talking to the head of computer science, Mr Hamm, we have decided that only teachers will have the ability to add students to the program, so the program must have two levels of access: admins for staff, and regular users for students.

Additionally, for the time being, we are not concerned about the cloud functionality of the program, however it is something they would be looking to implement in the future.

AGREEMENT WITH THE SCHOOL

Having now spoken to Mr Hamm on multiple occasions about my program, we have come to an agreement that I can develop my first prototype of my program for him to view upon completion. When this time comes, he has said that he will be more than happy to start using the program on the school's computer system and allow students and teachers to start being registered onto the program. However, this will only be the case should he be satisfied with my prototype, and that I be open to any changes that he sees will aid the school.

He seems very enthusiastic about the prospect of my program, and hopefully I am able to create the program that he has envisioned after our conversations.

LIMITATIONS

As this is my first large scale project for a stakeholder, the writing of my program will likely be a long and difficult task. In addition to this, given the language I will be using and my current knowledge of python, there are some functions that at the current moment I do not see myself as being able to complete in the current timeframe.

The main limitation that I have already discussed with the school is the cloud aspect of the program. In an ideal world, the program would launch with full cloud functionality, where the user is able to run the application from their laptop or desktop at home, and still have access to the database of questions, and be able to attempt quizzes from home. Given that I have learnt how to use SQLite instead of MySQL for providing the database, and I have little knowledge of how networking works in python, I have explained to the school that it is very unlikely that cloud functionality will be completed by the first prototype, however could be implemented in future prototypes when I have learned the skills required.

Like the networking aspect of the program, my first prototype will be limited to working on Windows computers, due to my lack of knowledge of porting python executable code to other formats such as android, mac or IOS. Although not a large issue due to the school's computers running Windows, it reduces the number of possible students that could utilise the program which is not ideal.

An original idea I had thought of when figuring out what the program would eventually be was a way of competing directly against friends or other people online. Whereas a leader board just lets users see how well they're doing against other users, the idea was to allow multiple users to answer quizzes against each other in real time. Although this idea in concept sounds like a fun way of revising with friends, the network infrastructure required for it to be implemented in a way that doesn't retract from the user's experience would be so immense, and may even require dedicated servers, which would cost a fair amount of money.

Although all the ideas I have discussed above would make for a better user experience with the program, it is due to their complexity, cost, and my lack of knowledge in some areas, which have lead me to leave them as ideas rather than try and incorporate them into my program for the time being.

HARDWARE & SOFTWARE REQUIREMENTS

I want my program to be well optimised so that it can be run on as many computers and laptops as possible, so that as many students and teachers can use it. Due to this I have decided on the minimum hardware and software requirements for my program to run well:
OS: Windows 7 – 32 bit
Memory: 1 GB DDR2
Processor: Intel Celeron Dual Core G1840 2.8GHz or equivalent
Hard Drive Space: 1GB
Graphics: Intel Integrated Graphics
Internet Connection: 512kb/s

OBJECTIVE LIST

Below is a list of my initial objectives for my program, as I progress through the development of my program, it is possible that while conducting interviews with my end-users, more objectives will arise. However, for now these are my objectives:

- Setup/General
    1. Load PyQt UI files from a folder
    2. Load SQLite database from a file
    3. Generate the GUI from PyQt files
    4. Have an easy-to-use and intuitive design throughout
    5. Load graphic resources from a file
    6. Import necessary modules into python (e.g. time, hashlib)
    7. Use object oriented programming to create a class for each window in the program
    8. Swap between windows seamlessly when a function is called to move between them
    9. Allow user to input data into text boxes
    10. Recognize double clicking a row of a table
    11. Working search bars to search through tables
    12. Validation for entry boxes throughout

- Login Screen
    13. When inputting a password, only display bullet points for security reasons
    14. Fetch usernames and passwords from the database when attempting to login
    15. Hash passwords for security reasons
    16. Be able to differentiate between a student and teacher account, moving the user to the according home screen
    17. Check if the user's password has been reset and if so, allow them to change it
    18. When changing password, check that the new password is confirmed before editing the database records
    19. Provide an output when an account hasn't been found

20. Provide an output when an entered password is incorrect
21. Create a button to allow new teachers to be registered
22. When registering a new teacher ensure:
    - No field is left empty
    - Teacher code is correct so only teachers can create new teacher accounts
    - The entered username does not already exist
23. Provide an output when the account has been successfully created
24. Add new teacher accounts to the database

- Teachers/Students Home Screen
    25. Allow access to viewing all student accounts (Teachers Only)
    26. Allow access to viewing all question sets (Teachers Only)
    27. Allow access to playing a round
    28. Allow access to view own results (Students Only)
    29. Allow user to logout

- Viewing Students Screen
    30. Fetch all students' data from the database
    31. Generate a table of all students, displaying name and username
    32. Be able to filter by students that the teacher teaches
    33. Allow teachers to add/remove students to/from their class
    34. Reset a student's password to a default one
    35. Ability to add new students to the database so long as a unique username is chosen
    36. Show results of any students from a teacher's class

- Viewing Results Screen
    37. Display a student's highest score in each of their attempted tests
    38. Allow access to in-depth analysis of progress in a given test
    39. Show a list of all scores ever received in a set, ordered by date
    40. Generate a line graph showing scores over time
    41. Scale the graph depending on how many results the student has and the range of the scores
    42. Show a specific selected score on the graph

- Viewing Question Sets Screen
    43. Fetch all question sets and their authors, and display in a table
    44. Allow access to adding new sets
    45. Have a browser for selecting a file from storage
    46. Import contents of file into database
    47. Allow access to editing sets
    48. Allow user to upload file with updated/edited questions
    49. View a leaderboard of everyone who's attempted a quiz
    50. Allow leaderboard to be printed

- Playing A Round Screen
    51. Fetch all question sets and their authors
    52. Create a table displaying all sets and authors
    53. Create a checkbox for whether a user wants to play multiple choice
    54. Allow playing a selected quiz by pressing 'Play'

- Gameplay Screen
    55. Fetch all questions from database for the given set
    56. Randomly select a question from the pool and then remove it from the pool
    57. Display the question in a text box
    58. If in hard mode:
        - Generate an input box
        - Allow entry of answer by pressing enter or pressing 'Play'
    59. If in easy mode:
        - Shuffle the 4 possible answers
        - Allocate each answer to a button
        - Question moves on when a button is pressed
    60. When a question starts, start a timer
    61. When a question is answered, stop the timer
    62. Generate a number of points based on elapsed time (kn where k is a constant and n is elapsed time)
    63. If answer is correct, add score to total
    64. If answer is incorrect, deduct score from total
    65. When there are no more questions left, go to the results screen

- Results Screen
    66. Display total score on the screen
    67. Display all incorrectly answered questions in a table along with the user's attempt and the correct answer
    68. If not all text fits in a row, allow user to double click to view in a separate box
    69. Fetch the top 10 unique scores for the quiz
    70. Display scores on a leaderboard
    71. Allow user to review their progress of the set if they've attempted it more than once
    72. Create a button to let the user return to the play screen
    73. Write new scores to the database

DESIGN

DECOMPOSING THE PROBLEM

In order to create a thorough design of my program, I must first take the problem and break it down into sections. Having already undertaken some interviews in my analysis section, I have used the responses to better decompose the problem.

Login System – All users of the program require a unique username instead of logging in using their names due to the possibility of two users having the same name. The program also needs a way of distinguishing teachers from students to allow them to utilise different parts of the program.

Registration of Teachers – New teachers will need to be added to the program, however it should be possible to do this without requiring the administrator, so there will need to be a way of only allowing teachers to register from the start screen of the program.

Registration of Students – To prevent exploitation of the program's functionality, the registration of students will need to be only accessible to teachers. From the interviews, this should encourage students to use the program more.

Adding/Editing Question Sets – Being able to add new question sets to the program needs to be intuitive and easy for teachers to encourage more sets to be added. Being able to go back and edit sets after creation will also be necessary in case of mistakes or changes to syllabi.

Student Organisation – Teachers need a way of knowing which students are in their classes and being able to move them in or out according to their timetables. They also need to be able to reset students' passwords in case they forget them.

Intuitive Gameplay – The game needs to be both fun and educational to hold the attention of students. The interface needs to be clean and easy to read to prevent confusion when answering questions.

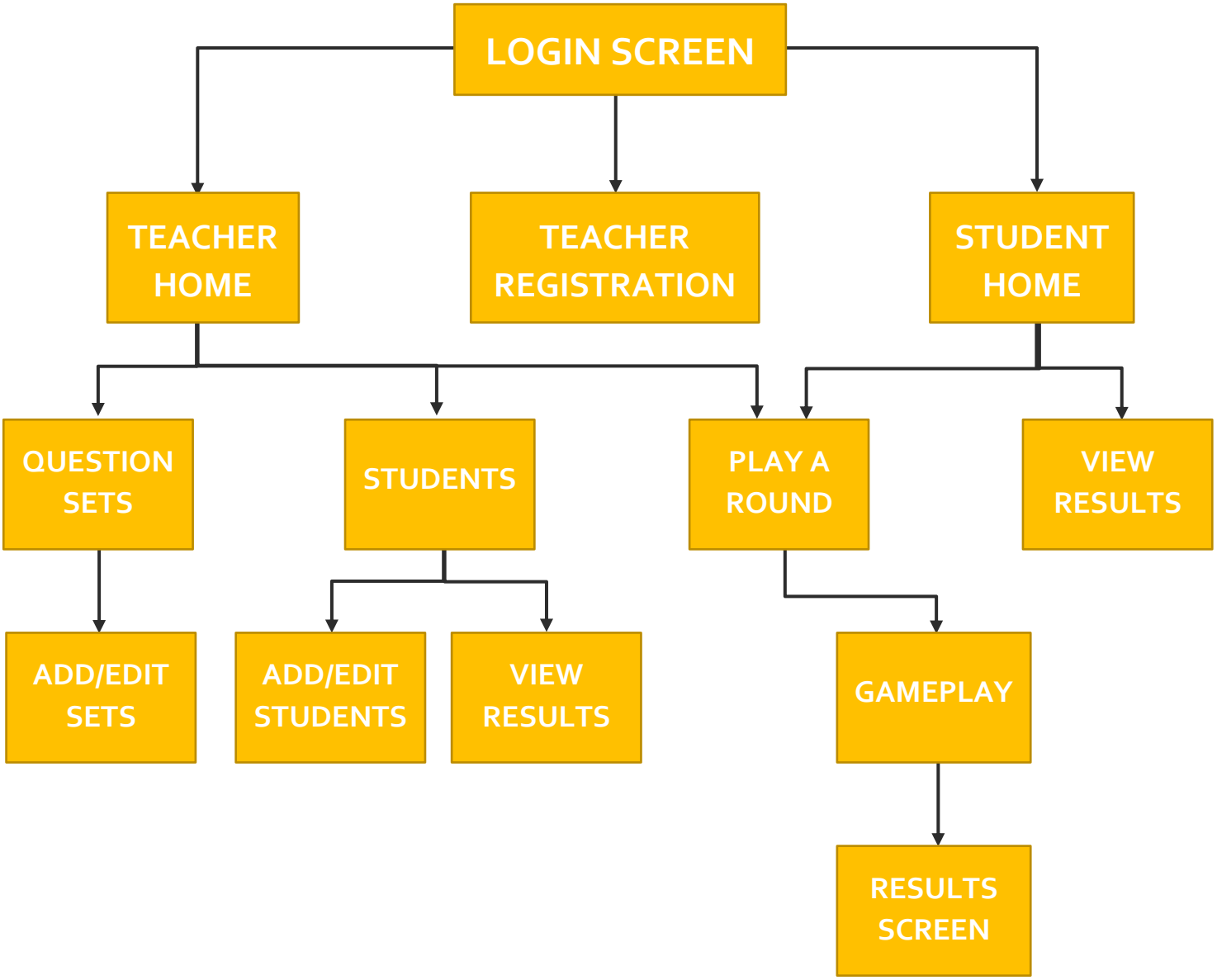
Difficulty Scaling – The game needs to be able to adjust its difficulty for students that are finding it too hard or easy; this could be achieved in multiple ways, including multiple choice answering.

Viewing Results – Students – Students will need a way to track their progress in the program, and using statistics and graphs would be a visually appealing way to achieve this.

Viewing Results – Teachers – Teachers will also need a way to track the progress of their students, both as individuals and as classes in order to assess their work. Having a way of ordering students in a leader board could promote competition.

STRUCTURE OF THE SOLUTION

In order to structure my program, I have designed a site-map of sorts to show how each of the sub-sections of my program will piece together.



Each of my sub-programs will be linked together as shown above, and each one will have its own PyQt UI file that will be loaded upon starting the program. Below I will list all the sub-programs and the components that will be within them.

Login Screen – Upon start-up all users will be met with this initial screen containing two boxes for users to enter their username and password, and then buttons for logging in or registering a new teaching account. There will be outputs for if the user incorrectly enters their username or password in some form.

Teacher Registration – If the user wants to register a new teacher, this screen will allow them to enter a new username, password, name and title. Additionally, to prevent students from registering themselves as teachers, there will be a box for entering a "Teacher Code" that will be exclusively given to teachers so that they can register. If someone tries to register without the code, they will be told they are unable to add a new account.

Teacher Home Screen – If the account logging in is a teaching account, they will be directed to this screen which will have buttons for viewing question sets, students, playing a game themselves, and logging out.

Question Sets Screen – This screen will have a list of all the created question sets, which will be sortable by author. For any sets that that user has created, they will have the option of being able to edit the questions. There will also be a button for going to the creation screen, where they can add more sets to the database.

Question Set Creation/Editing – This screen will let the user edit the title and description of a question set, and also add their own questions to it. They can either type them straight into the program or they will be able to import them from a formatted text document. This will also be where they can edit existing questions from their sets.

Viewing Students Screen – All the students stored in the database will be viewable from here; the teacher will be able to tab between all students and just the students that they teach. They will be able to add and remove students from their class, and there will be buttons for resetting their passwords if they forget them, viewing their progress, or editing their details.

Student Creation/Editing – When viewing a students' details, the teacher will be able to edit their name, reset their password, and reset their progress. They will also be able to add new students to the program with a default password that they can change on their first login.

Viewing Results – For both students and teachers, they will be able to view a graph outlining their progress sorted by single sets or overall. They will also be able to view leader boards to track how the student is doing compared to the year.

Play Game Menu – This screen will show a list of all the question sets within the database, and will allow the user to search by name, author, and topic. After selecting a set, they will be able to choose difficulty modifiers such as multiple choice and extended timers, and then when they click play they will start the game.

Gameplay – During the game, questions will be randomly selected from the set and displayed depending on the difficulty chosen. If multiple choice, the question and all possible answers will be displayed, and the answers will be buttons that the user can press to choose their answer. If not multiple choice, the question will be at the top, and there will be a box for the user to answer. There will also be a timer in the corner to let the user know how much time they have left to answer. At the end of time, or if they answer the question they will be shown which answer is correct, and highlight the incorrect ones, and points will be awarded or taken away accordingly.

Results Screen – At the end of a game, the user will be shown their score, along with a leader board showing other users who have attempted the set. The user will also be able to review their answers and see which ones they got incorrect. They will also be able to attempt the questions again, or go back to try another one.

INPUTS, OUTPUTS, PROCESSES & STORAGE

Input/output	**Purpose**
Buttons	Moving between windows, and selecting answers within the game.
Entry Boxes	Entering details for registration, logging in, searching, and answering questions.
Drop Down Menu	Used for selecting titles when registering teachers.
Table Views	Used for viewing data from the SQLite database.
List Views	Used for the leader boards of the program.
Message Boxes	Used to show user when there is an error.
Printing	Used for printing leader boards.
Labels	Used to display results for students.
Check Box	Used to filter tables

Processes	**Purpose**
Retrieving Data from Database	Load database from a SQLite file and select data.
Writing Data to Database	Write data to an SQLite file.
Generating Hashes	Generate hashes for passwords to ensure security.
Sort Data by Column	Question sets must be sortable to easily find relevant quizzes.
Calculate Score Based on Time	Score for each question must be calculated based on time taken to answer, sign of score is dependent on correctness.
Search by Name, Author, and Topic	Use a search bar to make searching for quizzes faster.

Generate Graphs based on Previous Results	Create line graphs to represent progress over time per quiz or overall.
Generate Scoreboards	Display the top 10 students and a user's position in a given quiz.
Reset User Passwords	Change user's password to a default by a teacher if user forgets their password.
Randomize Question Order and Answers	Questions from a set must be randomly selected to avoid pattern matching or cheating.
Import Questions from Text File	Make it easier to add questions by allowing teachers to import questions from a text file.
Load Graphical Interfaces	Load all PyQt files for the program's interface to run.

Storage	**Purpose**
SQLite Database	The database must be stored alongside the program so that it can be accessed for writing and reading while running the program.
PyQt GUI	All GUI files created from PyQt must be stored with the program so that they can be loaded when running the program to display the User Interface.
Python Modules	Additional python modules will be used for the program to run such as OS, Random, and Sys, and these must be imported into the program.

DATA STRUCTURES

Below are tables displaying how each of the tables within the database will be laid out, and how they will be connected together.

QSLinks Table

Table Column	Purpose	Validation
SID	The ID Number for the set used in the link	Must be an integer Must be unique in sets table
QID	The ID Number for the question used in the link	Must be an integer Must be unique in questions table

This table is used for linking questions and sets, allowing the program to easily fetch all the questions from a given set.

Questions Table

Table Column	Purpose	Validation
QID	The ID Number for the question	Must be an integer Must be unique in questions table
Question	The text of the actual question	Must be a string Cannot contain apostrophes or quote marks
Correct	The correct answer for the question	Must be a string Cannot contain apostrophes or quote marks
Wrong1	One of the incorrect answers used when playing multiple choice	Must be a string Cannot contain apostrophes or quote marks
Wrong2	One of the incorrect answers used when playing multiple choice	Must be a string Cannot contain apostrophes or quote marks
Wrong3	One of the incorrect answers used when playing multiple choice	Must be a string Cannot contain apostrophes or quote marks

This table holds all of the question within the database, along with their answers and an ID number for identifying them.

STLinks Table

Table Column	Purpose	Validation
TUser	The username of the teacher in the link	Must be a string Less than 15 characters Only alphanumerical characters
SUser	The username of the student in the link	Must be a string Less than 15 characters Only alphanumerical characters

This table is used for linking students and teachers, allowing the program to easily fetch all of a given teachers' students.

Scores Table

Table Column	Purpose	Validation
Username	The username of the user that received the score	Must be a string Less than 15 characters Only alphanumerical characters
SID	The set that the new score was set in	Must be an integer Must be unique in sets table
Score	The actual score of the user	Must be an integer
Date	The date and time that the score was recorded	Must be in date-time format

This table holds all of the scores of each of the users that get scores when using the program.

Sets Table

Table Column	Purpose	Validation
SID	The ID number of the set	Must be an integer Must be unique in this table
SetName	The name of the created set	Must be a string Only alphanumerical characters
Author	The username of the teacher that created the set	Must be a string Less than 15 characters Only alphanumerical characters

This table holds the details for all the created sets.

Students Table

Table Column	Purpose	Validation
Username	The username of the added student	Must be a string Less than 15 characters Only alphanumerical characters
Password	The hashed password for the given user	Must be a string Should be hashed
First Name	The first name of the user	Must be a string Made up of only letters
Surname	The surname of the user	Must be a string Can contain one dash for double barrelled names Made up of only letters besides the dash

This table holds all of the students in the database, along with their secured, hashed password, their username, and their name.

Teachers Table

Table Column	Purpose	Validation
Username	The username of the added teacher	Must be a string Less than 15 characters Only alphanumerical characters
Password	The hashed password for the given teacher	Must be a string Should be hashed
Title	The title chosen by the given teacher	Must be a string
Surname	The surname of the given user	Must be a string Can contain one dash for double barrelled names Made up of only letters besides the dash

This table holds all of the teachers in the database, along with their secured, hashed password, their username, title, and surname.

ENTITY RELATIONSHIP DIAGRAM

In order to show how the tables within the database will be connected together I have drawn an entity relationship diagram, and have identified any primary, secondary, and foreign keys within in each table.

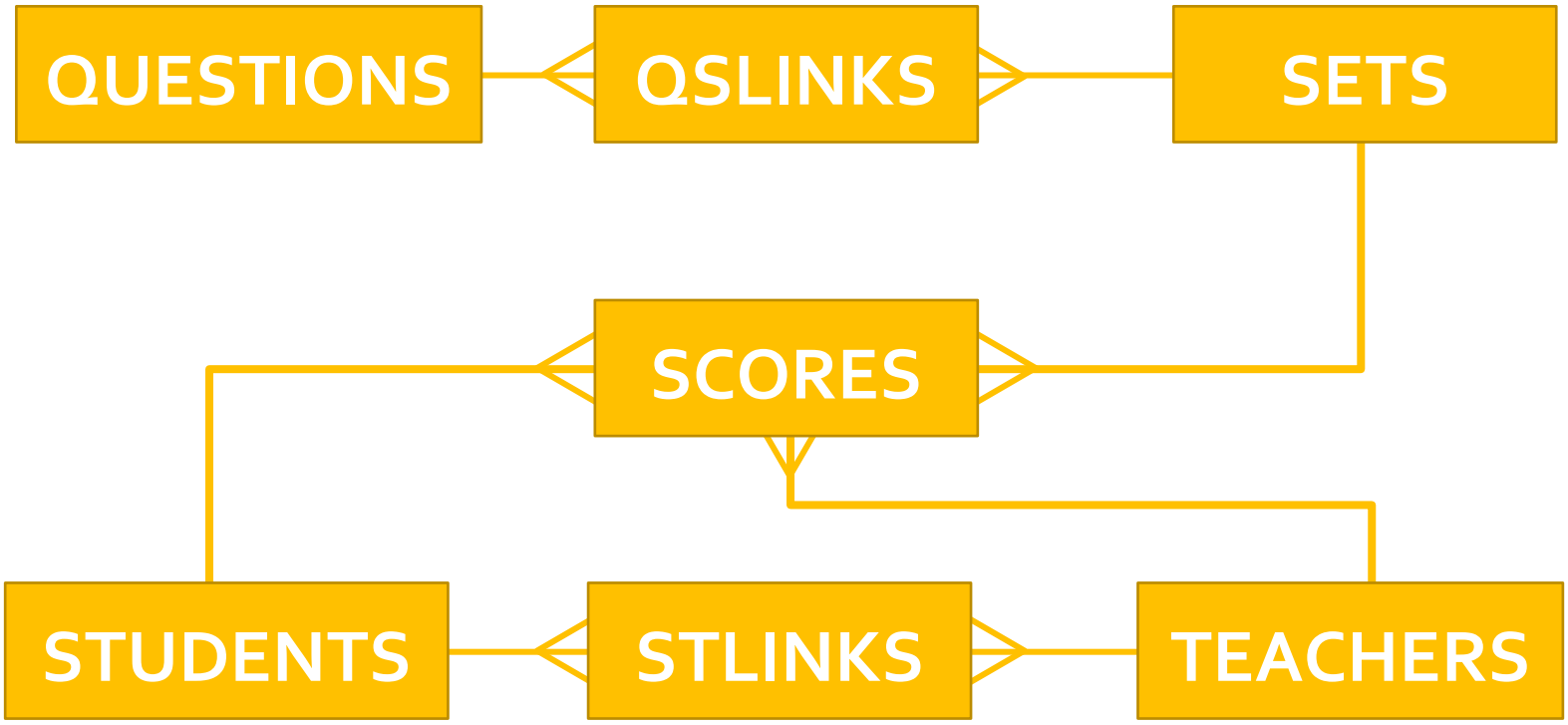


Table Name	Primary Key	Secondary Key	Foreign Key
Questions	QID	Question	QID
Sets	SID	SetName	SID
QSLinks	QID AND SID	QID, SID	None
Students	Username	Firstname, Surname	Username
Teachers	Username	Surname	Username
STLinks	TUser AND SUser	TUser,SUser	None
Scores	Date	SID,Username	None

ALGORITHMS AND UML DIAGRAMS

In order to begin thinking about how I am going to write my program, I have created UML diagrams for each of the classes which will be used in my program; for the most part, each of the windows in my program will be written as a class. Having written these UML diagrams, I have then used them to create algorithms for how the majority of the functions will work within the program.

LOGIN SCREEN

```
class Login()

    public function FetchData(table)
        Database = openRead(ProgramDatabase)
        Records = Database.read(table)
        Database.close()
        Return Records
    end function

    private procedure LoginAttempt()
        Username = input("Enter username") // Global variable
        Password = input("Enter password")
        AllStudents = FetchData("Students")
        AllTeachers = FetchData("Teachers")
        Access = "" // Global variable
        For i=0 to length(AllStudents)
            If Username == AllStudents[i][0] Then
                Password = Hash.Hash(Password)
                If Password == AllStudents[i][1] Then
                    If Password == Hash.Hash("Default") Then
                        print("Password has been reset")
                        new ResetPassword()
                    Else
                        print("Successful Login")
                        Access = "Student"
                        new StudentHome()
                    EndIf
                Else
                    print("Incorrect password")
                EndIf
                Break
            EndIf
        Next i
        If Access == "" Then
            For i=0 to length(AllTeachers)
                If Username == AllTeachers[i][0] Then
                    Password = Hash.Hash(Password)
                    If Password == AllTeachers[i][1] Then
                        print("Successful Login")
                        Access = "Teacher"
                        new TeacherHome()
                    Else
                        print("Incorrect password")
                    EndIf
                    Break
                EndIf
            Next i
        EndIf
        If Access == "" Then
            print("Account not found")
        EndIf
    end procedure

    If LoginButton Pressed Then
        LoginAttempt()
    EndIf

    If RegisterButton Pressed Then
        new TeacherRegistration()
    EndIf
end class
```

Login
Database: DB Records: LIST Username: STRING Password: STRING AllStudents: LIST AllTeachers: LIST Access: STRING
Constructor() LoginAttempt()

TEACHER REGISTRATION

TeacherRegistration
ContainsNumbers: BOOLEAN ContainsLetters: BOOLEAN IsAlphanumerical: BOOLEAN Lowercase: INTEGER Uppercase: INTEGER Username: STRING Password: STRING TeacherCode: STRING Title: STRING Surname: STRING UsernameResults: LIST PasswordResults: LIST CharacterResults: STRING Dashes: INTEGER AllStudents: LIST AllTeachers: LIST UsernameTaken: BOOLEAN Errors: LIST
Constructor() StringValidation() AddTeacher()

```
class TeacherRegistration()
    public function StringValidation(Word)
        ContainsNumbers = False
        ContainsLetters = False
        IsAlphanumerical = True
        Lowercase = 0
        Uppercase = 0
        for i=0 to Length(Word)
            If ASCII(Word[i]) > 48 AND ASCII(Word[i]) < 58 Then
                ContainsNumbers = True
            Else If ASCII(Word[i]) > 64 AND ASCII(Word[i]) < 90 Then
                ContainsLetters = True
                Uppercase = Uppercase + 1
            Else If ASCII(Word[i]) > 96 AND ASCII(Word[i]) < 123 Then
                ContainsLetters = True
                Lowercase = Lowercase + 1
            Else
                IsAlphanumerical = False
            End If
        Next i
        Return ContainsNumbers, ContainsLetters, IsAlphanumerical, Lowercase, Uppercase
    end function
    private procedure AddTeacher()
        Username = input("Enter username")
        Password = input("Enter password")
        TeacherCode = input("Enter teacher code")
        Title = input("Enter title")
        Surname = input("Enter surname")
        If length(Username) > 15 Then
            Add "Username is too long" to Errors
        EndIf
        UsernameResults = StringValidation(Username)
        If UsernameResults[2] == False Then
            Add "Only alphanumerical values are allowed in usernames" to Errors
        End If
        If length(Password) > 6 Then
            PasswordResults = StringValidation(Password)
            If PasswordResults[2] == False Then
                Add "Only alphanumerical values are allowed in passwords" to Errors
            End If
            If PasswordResults[3] == 0 Or PasswordResults[4] == 0 Then
                Add "Password must contain at least 1 uppercase and 1 lowercase letter" to Errors
            EndIf
        Else
            Add "Password must be longer than 6 characters" to Errors
        EndIf
        If TeacherCode != "ABCDE" Then
            Add "Teacher code must be correct to register new teachers" to Errors
        EndIf
        If Title == "Select Title Here" Then
            Add "You must select a title" to Errors
        EndIf
        Dashes = 0
        For i=0 to length(Surname)
            CharacterResults = StringValidation(Surname[i])
            If Surname[i] == "-" Then
                Dashes = Dashes + 1
            Else If CharacterResults[1] == False Then
                Add "Surnames are made up of only letters" to Errors
                Break
            EndIf
        Next i
        If Dashes > 1 Then
            Add "Only one dash is allowed in a surname" to Errors
        EndIf
        AllStudents = FetchData("Students")
        AllTeachers = FetchData("Teachers")
        UsernameTaken = False
        For i=0 to length(AllStudents)
            If Username == AllStudents[i][0] Then
                Add "Username Taken" to Errors
                UsernameTaken = True
                Break
            EndIf
        Next i
        If UsernameTaken == False Then
            For i=0 to length(AllTeachers)
                If Username == AllTeachers[i][0] Then
                    Add "Username Taken" to Errors
                    UsernameTaken = True
                    Break
                EndIf
            Next i
        EndIf
        If UsernameTaken == False Then
            If length(Errors) == 0 Then
                Password = Hash.Hash(Password)
                Database = openRead(ProgramDatabase)
                AllTeachers = Database.read("Teachers")
                AllTeachers.append(Username,Password,Surname,Title)
                Database.save()
                Database.close()
            Else
                print(Errors)
            EndIf
        EndIf
    end procedure
    If RegisterButton Pressed Then
        AddTeacher()
    EndIf
    If BackButton Pressed Then
        new Login()
    EndIf
end class
```

TEACHER HOME

```
class TeacherHome()
        If LogoutButton Pressed Then
                new Login()
        EndIf

        If PlayButton Pressed Then
                new PlayGame()
        EndIf

        If ViewSetsButton Pressed Then
                new ViewSets()
        EndIf

        If ViewStudentsButton Pressed Then
                new ViewStudents()
        EndIf
end class
```

TeacherHome
Constructor()
Logout()
Play()
ViewSets()
ViewStudents()

STUDENT HOME

```
class StdentHome()
        If LogoutButton Pressed Then
                new Login()
        EndIf

        If PlayButton Pressed Then
                new PlayGame()
        EndIf

        If ViewResultsButton Pressed Then
                new ViewResults(Username)
        EndIf
end class
```

StudentHome
Constructor()
Logout()
Play()
ViewResults()

VIEW QUESTION SETS

ViewSets
AllSets: LIST Table: LIST SearchWord: STRING Contains: BOOLEAN SetID: INTEGER AllLinks: LIST QuestionIDs: LIST AllQuestions: LIST Questions: LIST SelectedRow: INTEGER SelectedSet: LIST SetAuthor: STRING
Constructor() FetchQuestions() Search() AddSet() EditSet() Back()

```
class ViewSets()
    AllSets = FetchData("Sets")
    Table = []
    For i=0 to length(AllSets)
        Table.append(AllSets[i])
    Next i

    private procedure Search()
        SearchWord = input("Enter Search Term")
        AllSets = FetchData("Sets")
        Table = []
        For i=0 to Length(AllSets)
            Contains = False
            For j=0 to Length(AllSets[i][1])
                For k=0 to Length(SearchWord)
                    If j+k < Length(AllSets[i][1]) Then
                        If SearchWord[k] == AllSets[i][1][j+k] Then
                            If k == Length(SearchWord)-1 Then
                                Contains = True
                            EndIf
                        Else
                            Break
                        EndIf
                    EndIf
                Next k
            Next j
            If Contains == True Then
                Table.append(AllSets[i][1],AllSets[i][2])
            EndIf
        Next i
    end procedure

    private function FetchQuestions(Set)
        AllSets = FetchData("Sets")
        For i=0 to Length(AllSets)
            If AllSets[i][1] == Set[0] And AllSets[i][1] == Set[1] Then
                SetID = AllSets[i][0]
                Break
            EndIf
        Next i
        AllLinks = FetchData("SQLinks")
        QuestionIDs = []
        For i=0 to Length(AllLinks)
            If AllLinks[i][1] == SetID Then
                QuestionIDs.append(AllLinks[i][0])
            EndIf
        Next i
        AllQuestions = FetchData("Questions")
        Questions = []
        For i=0 to Length(QuestionIDs)
            For j=0 to Length(AllQuestions)
                If AllQuestions[j][0] == QuestionIDs[i] Then
                    Questions.append(AllQuestions[i])
                    Break
                EndIf
            Next j
        Next i
        Return Questions
    end function

    If AddSetButton Pressed Then
        new ImportQuestions()
    EndIf

    If EditSetButton Pressed Then
        SelectedRow = input("Enter selected row's number")
        SelectedSet = Table[SelectedRow]
        SetAuthor = SelectedSet[1]
        If Username == SetAuthor Then
            new AddEditQuestions(FetchQuestions(SelectedSet))
        Else
            print("You can only edit sets that you have created")
        EndIf
    EndIf

    If SearchButton Pressed Then
        Search()
    EndIf

    If BackButton Pressed Then
        new TeacherHome()
    EndIf
end class
```

IMPORT QUESTIONS

ImportQuestions
FileLocation: STRING FileContents: STRING ImportedQuestions: LIST
Constructor() ImportFile() CreateNewSet() Back()

```
class ImportQuestions()

        If BrowseFilesButton Pressed Then
                ImportedQuestions = []
                FileLocation = input("Enter File Location")
                FileContents = openRead(FileLocation)
                while Not FileContents.endOfFile()
                        Question = FileContents.readLine()
                        Add Question to ImportedQuestions
                endwhile
                FileContents.close()
                new AddEditQuestions(ImportedQuestions)
        EndIf

        If NewButton Pressed Then
                new AddEditQuestions([])
        EndIf

        If BackButton Pressed Then
                new ViewSets()
        EndIf
end class
```

ADD/EDIT QUESTIONS

```
class AddEditQuestions(Questions)

        Table = []
        For i=0 to length(Questions)
                Table.Append(Questions[i])
        Next i

        If AddQuestionButton Pressed Then
                Table.append([])
        EndIf

        If SaveChangesButton Pressed Then
                Title = input("Enter Set Title")
                AllSets = FetchData("Sets")
                Taken = False
                For i=0 to Length(AllSets)
                        If Title == AllSets[1] Then
                                Taken = True
                                Break
                        EndIf
                Next i
                If Taken == False Then
                        Questions = []
                        For i=0 to Length(Table)
                                Questions.append(Table[i])
                        Next i
                        Database = openRead(ProgramDatabase)
                        AllQuestions = Database.read("Questions")
                        AllLinks = Database.read("QSLinks")
                        AllSets = Database.read("Sets")

                        AllSets.append([Length(AllSets),Title,Username])

                        For i=0 to Length(Questions)
                                AllQuestions.append([Length(AllQuestions)+1, Questions[i][0],
                                                Questions[i][1], Questions[i][2], Questions[i][3],
                                                Questions[i][4]])
                                AllLinks.append([Length(AllSets),Length(AllQuestions)+1])
                        Next i
                        Database.save()
                        Database.close()
                Else
                        print("This question set name has already been taken")
                EndIf
        EndIf

        If DeleteQuestionsButton Pressed Then
                SelectedRow = input("Enter selected row's number")
                If SelectedRow >= 0 And Selected Row < Length(Table) Then
                        Table.delete(Table[SelectedRow])
                Else
                        print("Must select an existing row to delete")
                EndIf
        EndIf

        If BackButton Pressed Then
                new ViewSets()
        EndIf

end class
```

AddEditQuestions
Questions: LIST Username: STRING Table: LIST Title: STRING AllSets: LIST Taken: BOOLEAN Database: DB AllQuestions: LIST AllLinks: LIST SelectedRow: INTEGER
Constructor() AddQuestion() DeleteQuestion() Finish() Back()

VIEW STUDENTS

ViewStudents
AllStudents: LIST Table: LIST MyStudents: LIST AllLinks: LIST Filtered: LIST Contains: BOOLEAN SelectedRow: INTEGER SelectedStudent: STRING Database: DB TemporaryPassword: STRING Found: BOOLEAN
Constructor() ViewAll() ViewMy() ResetPassword() AddToClass() RemoveFromClass() ViewStudentResults() Back()

```
class ViewStudents()

private procedure ViewAll()
        AllStudents = FetchData("Students")
        Table = []
        For i=0 to length(AllStudents)
                Table.append(AllStudents[i][0],AllStudents[i][2],AllStudents[i][3])
        Next i
end procedure

private procedure ViewMy()
        Table = []
        MyStudents = []
        AllLinks = FetchData("STLinks")
        For i=0 to Length(AllLinks)
                If AllLinks[i][1] == Username Then
                        MyStudents.append(AllLinks[i][0])
                EndIf
        Next i

        For i=0 to Length(AllStudents)
                For j=0 to Length(MyStudents)
                        If AllStudents[i][0] == MyStudents[j] Then
                                Table.append(AllStudents[i][0], AllStudents[i][2],
                                        AllStudents[i][3])
                                Break
                        EndIf
                Next j
        Next i
end procedure

private procedure Search()
        SearchWord = input("Enter Search Term")
        Filtered = []
        For i=0 to Length(Table)
                Contains = False
                For j=0 to Length(Table[i][0])
                        For k=0 to Length(SearchWord)
                                If j+k < Length(Table[i][0]) Then
                                        If SearchWord[k] == Table[i][0][j+k] Then
                                                If k == Length(SearchWord)-1 Then
                                                        Contains = True
                                                EndIf
                                        Else
                                                Break
                                        EndIf
                                EndIf
                        Next k
                Next j
                If Contains == True Then
                        Filtered.append(Table[i])
                EndIf
        Next i

        Table = []
        For i=0 to Length(Filtered)
                Table.append(Filtered[i])
        Next i

end procedure

If ResetPasswordButton Pressed Then
        SelectedRow = input("Enter selected row's number")
        If SelectedRow > 0 And SelectedRow < Length(Table) Then
                SelectedStudent = Table[SelectedRow][0]
                Database = openRead(ProgramDatabase)
                AllStudents = Database.read("Students")
                For i=0 to Length(AllStudents)
                        If SelectedStudent == AllStudents[i][0] Then
                                TemporaryPassword = Hash.Hash("Default")
                                AllStudents[i][1] = TemporaryPassword
                                Database.save()
                                Database.close()
                                Break
                        EndIf
                Next i
        Else
                print("Must select an existing student to reset their password")
        EndIf
EndIf
```

Pseudocode continued on next page

```
    If ViewMyStudentsButton Pressed Then
            ViewMy()
    EndIf

    If ViewAllStudentsButton Pressed Then
            ViewAll()
    EndIf

    If SearchButton Pressed Then
            Search()
    EndIf

    If RegisterStudentButton Pressed Then
            new StudentRegistration()
    EndIf

    If AddToClassButton Pressed Then
            SelectedRow = input("Enter selected row's number")
            If SelectedRow > 0 And SelectedRow < Length(Table) Then
                    SelectedStudent = Table[SelectedRow][0]
                    Database = openRead(ProgramDatabase)
                    AllLinks = Database.read("STLinks")
                    Found = False
                    For i=0 to Length(AllLinks)
                            If AllLinks[i] == [SelectedStudent,Username] Then
                                    print("Student already in class")
                                    Found = True
                                    Break
                            EndIf
                    Next i
                    If Found == False Then
                            AllLinks.append([SelectedStudent,Username])
                            Database.save()
                    EndIf
                    Database.close()
            Else
                    print("Must select an existing student")
            EndIf
    EndIf

    If RemoveFromClass Button Pressed Then
            SelectedRow = input("Enter selected row's number")
            If SelectedRow > 0 And SelectedRow < Length(Table) Then
                    SelectedStudent = Table[SelectedRow][0]
                    Database = openRead(ProgramDatabase)
                    AllLinks = Database.read("STLinks")
                    For i=0 to Length(AllLinks)
                            If AllLinks[i] == [SelectedStudent,Username] Then
                                    AllLinks.delete(AllLinks[i])
                                    Database.save()
                                    Database.close()
                                    Break
                            EndIf
                    Next i
            Else
                    print("Must select an existing student")
            EndIf
    EndIf

    If ViewStudentResultsButton Pressed Then
            SelectedRow = input("Enter selected row's number")
            If SelectedRow > 0 And SelectedRow < Length(Table) Then
                    SelectedStudent = Table[SelectedRow][0]
                    new ViewResults(SelectedStudent)
            Else
                    print("Must select an existing student to view")
            EndIf
    EndIf

    If BackButton Pressed Then
            new TeacherHome()
    EndIf

    ViewAll()
end class
```

ADD STUDENTS

AddStudents
Student_Username: STRING Password: STRING Firstname: STRING Surname: STRING Errors: LIST UsernameResults: LIST PasswordResults: LIST Dashes: INTEGER CharacterResults: LIST AllStudents: LIST AllTeachers: LIST UsernameTaken: BOOLEAN Database: DB AllLinks: LIST
Constructor() AddStudent() Back()

```
class StudentRegistration()

    private procedure AddStudent()
        Student_Username = input("Enter username")
        Password = input("Enter password")
        Firstname = input("Enter student's first name")
        Surname = input("Enter student's surname")
        Errors = []

        If length(Student_Username) > 15 Then
            Add "Username is too long" to Errors
        EndIf

        UsernameResults = StringValidation(Username)
        If UsernameResults[2] == False Then
            Add "Only alphanumerical values are allowed in usernames" to Errors
        End If

        If length(Password) > 6 Then
            PasswordResults = StringValidation(Password)
            If PasswordResults[2] == False Then
                Add "Only alphanumerical values are allowed in passwords" to Errors
            End If
            If PasswordResults[3] == 0 Or PasswordResults[4] == 0 Then
                Add "Password must contain at least 1 uppercase and 1 lowercase letter" to Errors
            EndIf
        Else
            Add "Password must be longer than 6 characters" to Errors
        EndIf

        Dashes = 0
        For i=0 to length(Surname)
            CharacterResults = StringValidation(Surname[i])
            If Surname[i] == "-" Then
                Dashes = Dashes + 1
            Else If CharacterResults[1] == False Then
                Add "Surnames are made up of only letters" to Errors
                Break
            EndIf
        Next i
        If Dashes > 1 Then
            Add "Only one dash is allowed in a surname" to Errors
        EndIf

        AllStudents = FetchData("Students")
        AllTeachers = FetchData("Teachers")
        UsernameTaken = False
        For i=0 to length(AllStudents)
            If Student_Username == AllStudents[i][0] Then
                Add "Username Taken" to Errors
                UsernameTaken = True
                Break
            EndIf
        Next i
        If UsernameTaken == False Then
            For i=0 to length(AllTeachers)
                If Student_Username == AllTeachers[i][0] Then
                    Add "Username Taken" to Errors
                    UsernameTaken = True
                    Break
                EndIf
            Next i
        EndIf
        If UsernameTaken == False Then
            If length(Errors) == 0 Then
                Password = Hash.Hash(Password)
                Database = openRead(ProgramDatabase)
                AllStudents = Database.read("Students")
                AllStudents.append(Student_Username,Password,Firstname,Surname)
                AllLinks = Database.read("STLinks")
                AllLinks.append(Student_Username,Username)
                Database.save()
                Database.close()
            Else
                print(Errors)
            EndIf
        EndIf
    end procedure

    If RegisterButton Pressed Then
        AddStudent()
    EndIf

    If BackButton Pressed Then
        new ViewStudents()
    EndIf
end class
```

VIEW RESULTS

ViewResults
Table: LIST AllScores: LIST PrevFound: BOOLEAN AllSets: LIST SearchWord: STRING Filtered: LIST Contains: BOOLEAN SelectedRow: INTEGER SelectedSetID: INTEGER
Constructor() DisplayAll() Search() Analysis() Back()

```
class ViewResults(ViewingUser)

    private procedure DisplayAll()
        Table = []
        AllScores = FetchData("Scores")
        For i=0 to Length(AllScores)
            If AllScores[i][0] == ViewingUser Then
                PrevFound = False
                For j=0 to Length(Table) Then
                    If AllScores[i][1] == Table[j][0] Then
                        PrevFound = True
                        If AllScores[i][2] > Table[j][2] Then
                            Table.delete(Table[j])
                            Table.append(AllScores[i][1],"",
                                AllScores[i][2])
                        EndIf
                    EndIf
                If PrevFound == False Then
                    Table.append(AllScores[i][1],"",AllScores[i][2])
                EndIf
                Next j
            EndIf
        Next i
        AllSets = FetchData("Sets")
        For i=0 to Length(Table)
            For j=0 to Length(AllSets)
                If Table[i][0] == AllSets[j][0] Then
                    Table[i][1] = AllSets[j][1]
                    Break
                EndIf
            Next j
        Next i
    end procedure

    private procedure Search()
        SearchWord = input("Enter Search Term")
        Filtered = []
        For i=0 to Length(Table)
            Contains = False
            For j=0 to Length(Table[i][1])
                For k=0 to Length(SearchWord)
                    If j+k < Length(Table[i][1]) Then
                        If SearchWord[k] == Table[i][1][j+k] Then
                            If k == Length(SearchWord)-1 Then
                                Contains = True
                            EndIf
                        Else
                            Break
                        EndIf
                    EndIf
                Next k
            Next j
            If Contains == True Then
                Filtered.append(Table[i])
            EndIf
        Next i
        Table = []
        For i=0 to Length(Filtered)
            Table.append(Filtered[i])
        Next i
    end procedure

    If AnalysisButton Pressed Then
        SelectedRow = input("Enter selected row's number")
        If SelectedRow > 0 And SelectedRow < Length(Table) Then
            SelectedSetID = Table[SelectedRow][0]
            new Analysis(SelectedSet,ViewingUser)
        Else
            print("Must select an existing set to analyse")
        EndIf
    EndIf

    If SearchButton Pressed Then
        DisplayAll()
        Search()
    EndIf

    If BackButton Pressed Then
        If Access == "Student" Then
            new StudentHome()
        Else
            new ViewStudents()
        EndIf
    EndIf

    DisplayAll()
end class
```

```
class Analysis(CurrentSetID,CurrentUser,GameResultsSettings)
        private function FetchAllResults(SetID,User)
                AllScores = FetchData("Scores")
                UserScores = []
                For i=0 to Length(AllScores)
                        If AllScores[i][0] == User And AllScores[i][1] == SetID Then
                                UserScores.append(AllScores[i][2])
                        EndIf
                Next i
                Return UserScores
        end function

        private function Largest(List)
                LARGEST = -999999999999999
                For i=0 to Length(List)
                        If List[i] > LARGEST Then
                                LARGEST = List[i]
                        EndIf
                Next i
                Return LARGEST
        end function

        private function Smallest(List)
                SMALLEST = 999999999999999
                For i=0 to Length(List)
                        If List[i] < SMALLEST Then
                                SMALLEST = List[i]
                        EndIf
                Next i
                Return SMALLEST
        end function

        private procedure DisplayResults()
                Scores = FetchAllResults(CurrentSetID,CurrentUser)
                Largest = Largest(Scores)
                Smallest = Smallest(Scores)
                GRAPH_HEIGHT = 100
                GRAPH_WIDTH = 100
                X_Interval = INT(GRAPH_WIDTH/Length(Scores))
                X_Value = 0
                GraphPoints = []
                For i=0 to Length(Scores)
                        Y_Value = INT((GRAPH_HEIGHT*(Scores[i]-Smallest))/(Largest-Smallest))
                        GraphPoints.append(X_Value,Y_Value)
                        X_Value = X_Value+X_Interval
                Next i
                For i=0 to Length(GraphPoints)
                        Plot GraphPoints[i] on Graph // Code for plotting the co-ordinates onto a graph in the lanugage I
                                                        choose would go here
                Next i
                ZeroLine_Y_Value = INT((GRAPH_HEIGHT*(0-Smallest))/(Largest-Smallest))
                Draw Horizontal Line on graph at y=ZeroLine_Y_Value // Code for drawing a straight line at this value would go
                                                                        here
        end procedure

        If BackButton Pressed Then
                If CurrentUser == Username Then
                        new GameResults(GameResultsSettings)
                Else
                        new ViewResults(CurrentUser)
                EndIf
        EndIf

        DisplayResults()
end class
```

Analysis
AllScores: LIST UserScores: LIST LARGEST: INTEGER (CONSTANT) SMALLEST: INTEGER (CONSTANT) Scores: LIST Largest: INTEGER Smallest: INTEGER GRAPH_HEIGHT: INTEGER (CONSTANT) GRAPH_WIDTH: INTEGER (CONTSTANT) X_Interval: INTEGER X_Value: INTEGER Y_Value: INTEGER GraphPoints: LIST ZeroLine_Y_Value: INTEGER
Constructor() FetchAllResults() Largest() Smallest() DisplayResults() Back()

PLAY GAME

PlayGame
Table: LIST AllSets: LIST SearchWord: STRING Contains: BOOLEAN RowNumber: INTEGER SelectedSet: LIST MultipleChoice: BOOLEAN SelectedRow: INTEGER
Constructor() Search() RandomQuiz() Play() Back()

```
class PlayGame()
    AllSets = FetchData("Sets")
    Table = []
    For i=0 to length(AllSets)
        Table.append(AllSets[i])
    Next i

    private procedure Search()
        SearchWord = input("Enter Search Term")
        AllSets = FetchData("Sets")
        Table = []
        For i=0 to Length(AllSets)
            Contains = False
            For j=0 to Length(AllSets[i][1])
                For k=0 to Length(SearchWord)
                    If j+k < Length(AllSets[i][1]) Then
                        If SearchWord[k] == AllSets[i][1][j+k] Then
                            If k == Length(SearchWord)-1 Then
                                Contains = True
                            EndIf
                        Else
                            Break
                        EndIf
                    EndIf
                Next k
            Next j
            If Contains == True Then
                Table.append(AllSets[i])
            EndIf
        Next i
    end procedure

    If RandomButton Pressed Then
        RowNumber = Random.RandomNumber(1,Length(Table)) // Will use an external module for
                                                         selecting random numbers
        SelectedSet = Table[RowNumber]
        MultipleChoice = input("Would you like to play multiple choice? Y/N")
        If MultipleChoice == "Y" Then
            new EasyMode(SelectedSet)
        Else If MultipleChoice == "N" Then
            new HardMode(SelectedSet)
        Else
            print("Must select an option")
        EndIf
    EndIf

    If PlayButton Pressed Then
        SelectedRow = input("Enter selected row's number")
        If SelectedRow > 0 And SelectedRow < Length(Table) Then
            SelectedSet = Table[SelectedRow]
            MultipleChoice = input("Would you like to play multiple choice? Y/N")
            If MultipleChoice == "Y" Then
                new EasyMode(SelectedSet)
            Else If MultipleChoice == "N" Then
                new HardMode(SelectedSet)
            Else
                print("Must select an option")
            EndIf
        Else
            print("Must select an existing set to play")
        EndIf
    EndIf

    If SearchButton Pressed Then
        Search()
    EndIf

    If BackButton Pressed Then
        If Access == "Student" Then
            new StudentHome()
        Else
            new TeacherHome()
        EndIf
    EndIf
end class
```

EASY MODE (MULTIPLE CHOICE)

EasyMode
SetID: INTEGER AllLinks: LIST QuestionIDs: LIST Questions: LIST ShuffledList: LIST Index: INTEGER SetQuestions: LIST IncorrectQuestions: LIST Total: INTEGER CurrentQuestion: STRING Answers: LIST StartTime: DATE/TIME Choice: INTEGER EndTime: DATE/TIME ElapsedTime: INTEGER
Constructor() FetchQuestions() ShuffleList() AskQuestion()

```
class EasyMode(Set)
    private function FetchQuestions(Set)
        SetID = Set[0]

        AllLinks = FetchData("SQLinks")
        QuestionIDs = []
        For i=0 to Length(AllLinks)
            If AllLinks[i][1] == SetID Then
                QuestionIDs.append(AllLinks[i][0])
            EndIf
        Next i
        AllQuestions = FetchData("Questions")
        Questions = []
        For i=0 to Length(QuestionIDs)
            For j=0 to Length(AllQuestions)
                If AllQuestions[j][0] == QuestionIDs[i] Then
                    Questions.append(AllQuestions[i])
                    Break
                EndIf
            Next j
        Next i
        Return Questions
    end function

    private function ShuffleList(List)
        ShuffledList = []
        For i=0 to Length(List)
            Index = Random.RandomNumber(0,Length(List))
            ShuffledList.append(List[Index])
            List.delete(List[Index])
        Next i
        Return ShuffledList
    end function

    SetQuestions = ShuffleList(FetchQuestions(Set))
    IncorrectQuestions = []
    Total = 0

    private procedure AskQuestion()
        CurrentQuestion = SetQuestions[0][1]
        Answers = [SetQuestions[0][2],SetQuestions[0][3],SetQuestions[0][4],
            SetQuestions[0][5]]
        Answers = ShuffleList(Answers)
        StartTime = System.Time // Will use a module to get the current time
        print(CurrentQuestion)
        print(Answers)
        Choice = input("Enter your chosen answer (1/2/3/4)")
        If Choice > 0 And Choice < 5 Then
            EndTime = System.Time // Will use a module to get the current time
            ElapsedTime = EndTime - StartTime
            If Answers[Choice-1] == SetQuestions[0][2] Then
                print("Correct Answer!")
                Total = Total + ElapsedTime
            Else
                print("Incorrect Answer")
                Total = Total - ElapsedTime
                IncorrectQuestions.append(CurrentQuestion,
                                          SetQuestions[0][2],
                                          Answers[Choice-1])
            EndIf
            SetQuestions.delete(SetQuestions[0])
            If Length(SetQuestions) > 0 Then
                AskQuestion()
            Else
                new GameResults(Set,IncorrectQuestions,Total)
            EndIf

        Else
            print("Must choose a given option")
        EndIf
    end procedure

    AskQuestion()
end class
```

HARD MODE (NON-MULTIPLE CHOICE)

HardMode
SetID: INTEGER AllLinks: LIST QuestionIDs: LIST Questions: LIST ShuffledList: LIST Index: INTEGER SetQuestions: LIST IncorrectQuestions: LIST Total: INTEGER CurrentQuestion: STRING Answer: STRING StartTime: DATE/TIME EndTime: DATE/TIME ElapsedTime: INTEGER
Constructor() FetchQuestions() ShuffleList() AskQuestion()

```
class HardMode(Set)
    private function FetchQuestions(Set)
        SetID = Set[0]

        AllLinks = FetchData("SQLinks")
        QuestionIDs = []
        For i=0 to Length(AllLinks)
            If AllLinks[i][1] == SetID Then
                QuestionIDs.append(AllLinks[i][0])
            EndIf
        Next i

        AllQuestions = FetchData("Questions")
        Questions = []
        For i=0 to Length(QuestionIDs)
            For j=0 to Length(AllQuestions)
                If AllQuestions[j][0] == QuestionIDs[i] Then
                    Questions.append(AllQuestions[i])
                    Break
                EndIf
            Next j
        Next i
        Return Questions
    end function

    private function ShuffleList(List)
        ShuffledList = []
        For i=0 to Length(List)
            Index = Random.RandomNumber(0,Length(List))
            ShuffledList.append(List[Index])
            List.delete(List[Index])
        Next i
        Return ShuffledList
    end function

    SetQuestions = ShuffleList(FetchQuestions(Set))
    IncorrectQuestions = []
    Total = 0

    private procedure AskQuestion()
        CurrentQuestion = SetQuestions[0][1]
        StartTime = System.Time // Will use a module to get the current time
        print(CurrentQuestion)
        Answer = input("Enter your answer")
        EndTime = System.Time // Will use a module to get the current time
        ElapsedTime = EndTime – StartTime

        If Answer == SetQuestions[0][2] Then
            print("Correct Answer!")
            Total = Total + ElapsedTime
        Else
            print("Incorrect Answer")
            Total = Total - ElapsedTime
            IncorrectQuestions.append(CurrentQuestion,SetQuestions[0][2],Answer)
        EndIf
        SetQuestions.delete(SetQuestions[0])

        If Length(SetQuestions) > 0 Then
            AskQuestion()
        Else
            new GameResults(Set,IncorrectQuestions,Total)
        EndIf
    end procedure

    AskQuestion()
end class
```

GAME RESULTS

```
class GameResults(Set,IncorrectQuestions,Total)

        print(Total)
        Table = IncorrectQuestions

        If DetailsButton Pressed Then
                SelectedRow = input("Enter the row number you want details for")
                If SelectedRow > 0 And SelectedRow < Length(Table) Then
                        SelectedQuestion = Table[SelectedRow]
                        print(SelectedQuestion)
                Else
                        print("Must select an existing question")
                EndIf
        EndIf

        If AnalyseButton Pressed Then
                new Analysis(Set[0],Username,[Set,IncorrectQuestions,Total])
        EndIf


        If LeaderboardButton Pressed Then
                new Leaderboard(Set[0],[Set,IncorrectQuestions,Total])
        EndIf

        If HomeButton Pressed Then
                If Access == "Student" Then
                        new StudentHome()
                Else
                        new TeacherHome()
                EndIf
        EndIf

        If NewSetButton Pressed Then
                new PlayGame()
        EndIf

end class
```

GameResults
Table: LIST SelectedRow: INTEGER SelectedQuestion: LIST
Constructor() Details() Analysis() Leaderboard() Home() NewSet()

LEADERBOARD

```
class  Leaderboard(SetID,GameResultsSettings)

        AllScores = FetchData("Scores")
        SetScores = []
        For i=0 to Length(AllScores)
                If AllScores[i][1] == SetID Then
                        PrevFound = False
                        For j=0 to Length(SetScores)
                                If AllScores[i][0] == SetScores[j][0] Then
                                        PrevFound = True
                                        If AllScores[i][2] > SetScores[j][1] Then
                                                SetScores.delete(SetScores[j])
                                                SetScores.append([AllScores[i][0],AllScores[i][2]])
                                        EndIf
                                EndIf

                        If PrevFound == False Then
                                SetScores.append([AllScores[i][0],AllScores[i][2]])
                        EndIf

                        Next j
                EndIf
        Next i

        private procedure BubbleSort()
                For x=1 to Length(SetScores)
                        For i=0 to Length(SetScores)-x
                                If SetScores[i][1] < SetScores[i+1][1] Then
                                        Temp = SetScores[i+1][1]
                                        SetScores[i+1] = SetScores[i]
                                        SetScores[i] = Temp
                                EndIf
                        Next i
                Next x
        end procedure

        BubbleSort()

        Table = []
        If Length(SetScores) > 10 Then
                For i=0 to 9
                        Table.append(SetScores[i])
                Next i
        Else
                For i=0 to Length(SetScores)
                        Table.append(SetScores[i])
                Next i
        EndIf

        If BackButton Pressed Then
                new GameResults(GameResultsSettings)
        EndIf

end class
```

Leaderboard
AllScores: LIST SetScores: LIST PrevFound: BOOLEAN Temp: STRING Table: LIST
Constructor() BubbleSort() Back()

RESET PASSWORD

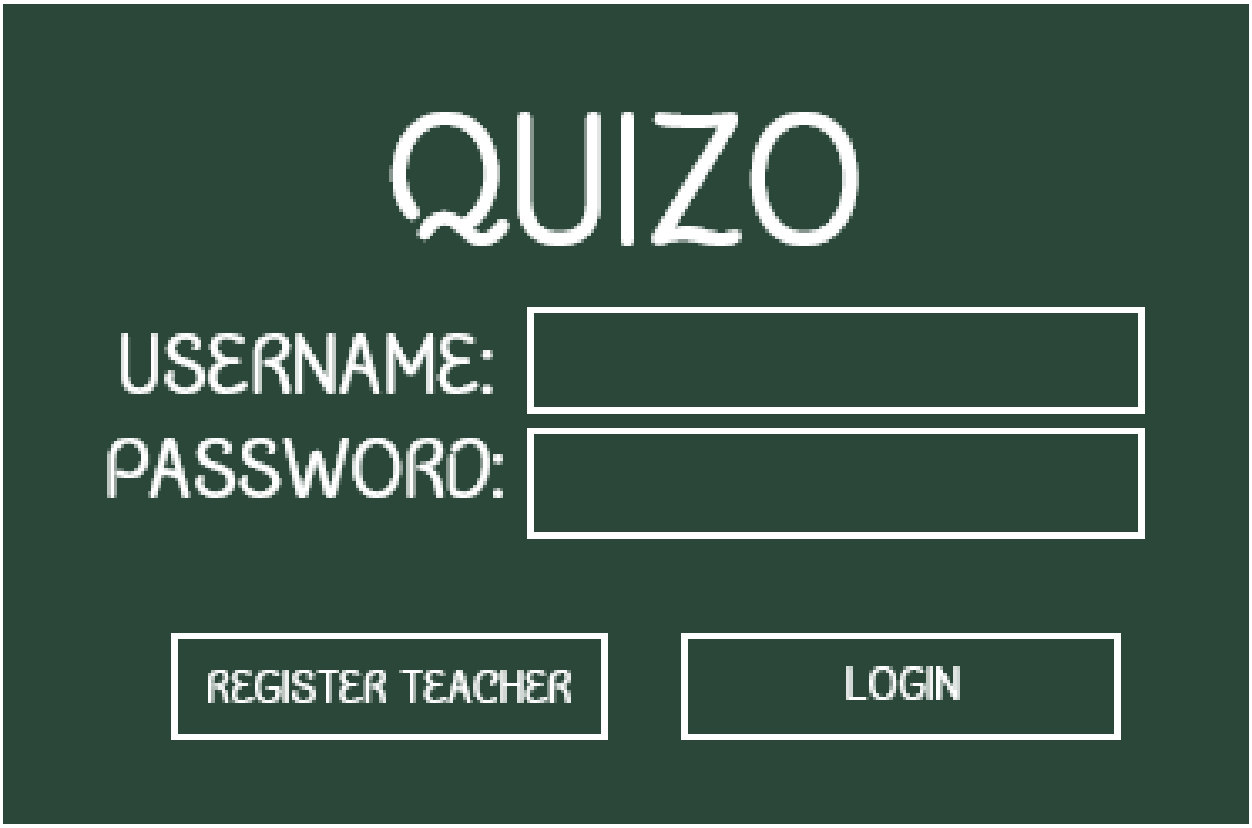
```
class ResetPassword()

        NewPassword = input("Enter new password")
        Confirm = input("Confirm new password")

        If NewPassword == Confirm Then
                Errors = []

                If length(Password) > 6 Then
                        PasswordResults = StringValidation(Password)
                        If PasswordResults[2] == False Then
                                Add "Only alphanumerical values are allowed in passwords" to Errors
                        End If

                        If PasswordResults[3] == 0 Or PasswordResults[4] == 0 Then
                                Add "Password must contain at least 1 uppercase and 1
                                    lowercase letter" to Errors
                        EndIf
                Else
                        Add "Password must be longer than 6 characters" to Errors
                EndIf

                If Length(Errors) == 0 Then
                        Database = openRead(ProgramDatabase)
                        AllStudents = Database.read("Students")
                        For i=0 to Length(AllStudents)

                                If AllStudents[i][0] == Username Then
                                        TempStudent = AllStudents[i]
                                        TempStudent[1] = Hash.Hash(NewPassword)
                                        AllStudents.delete(AllStudents[i])
                                        AllStudents.append(TempStudent)
                                        Break
                                EndIf

                        Next i
                Else
                        print(Errors)
                EndIf
        Else
                print("Passwords do not match")
        EndIf

        If BackButton Pressed Then
                new Login()
        EndIf

end class
```

ResetPassword
NewPassword: STRING Confirm: STRING Errors:LIST PasswordResults: LIST Database: DB AllStudents: LIST TempStudent: LIST
Constructor() Back()

USABILITY FEATURES

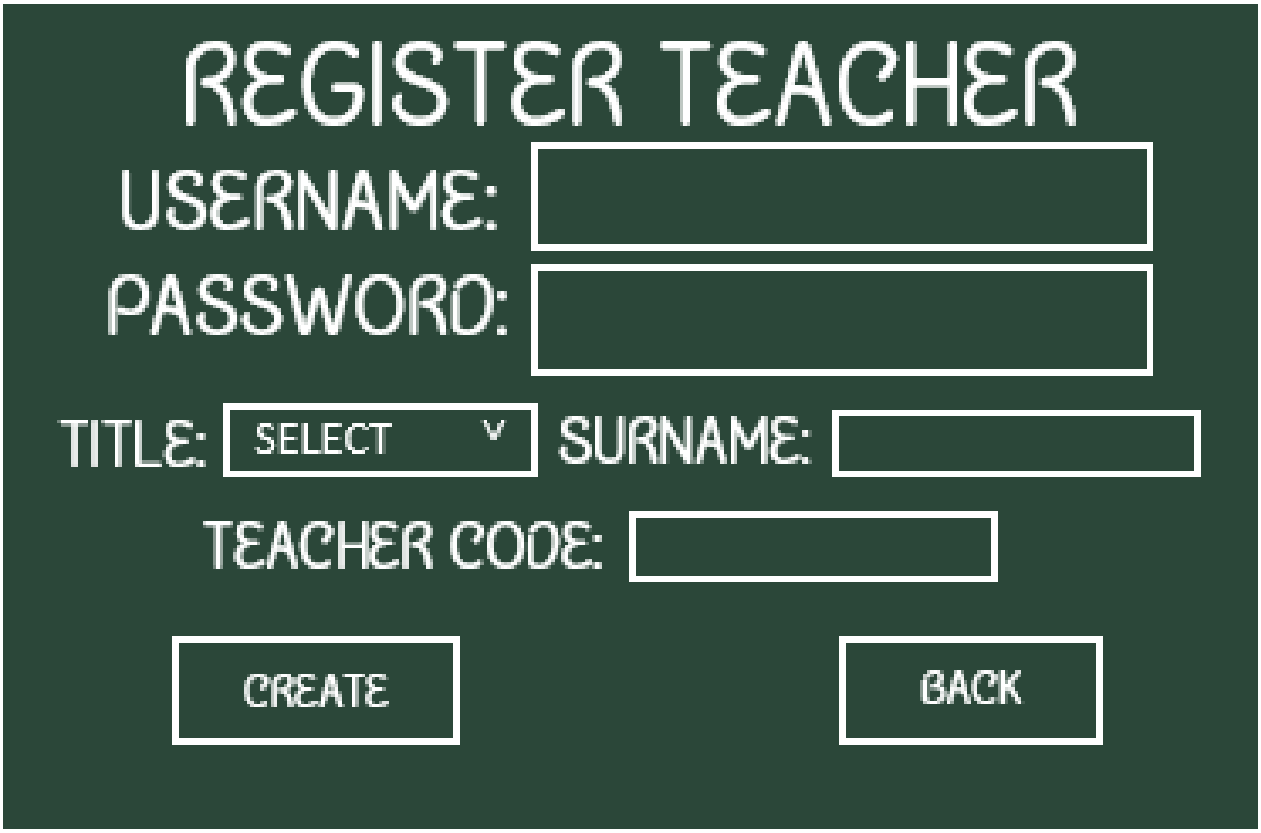
Below are a number of designs I've made using Powerpoint for my program, and I have annotated how their current designs make them suitable for my program and make it more accessible.

LOGIN SCREEN



The above is a design for my login window. After a bit of searching I decided that a chalkboard look would be both appealing and clean, and from that I found a high definition stock photo of a chalkboard and a clean chalk font.

I've drawn some large buttons so they're easy to click on and read, and the two entry boxes are able to be navigated using the tab button rather than having to use the mouse to make it faster to move around the screen.



This is the design for the teacher registration screen. When the button on the login screen is pressed, the user will be shown this screen. There are text entry boxes for each of the necessary details for a teachers account, such as username, password, surname, and teacher code.

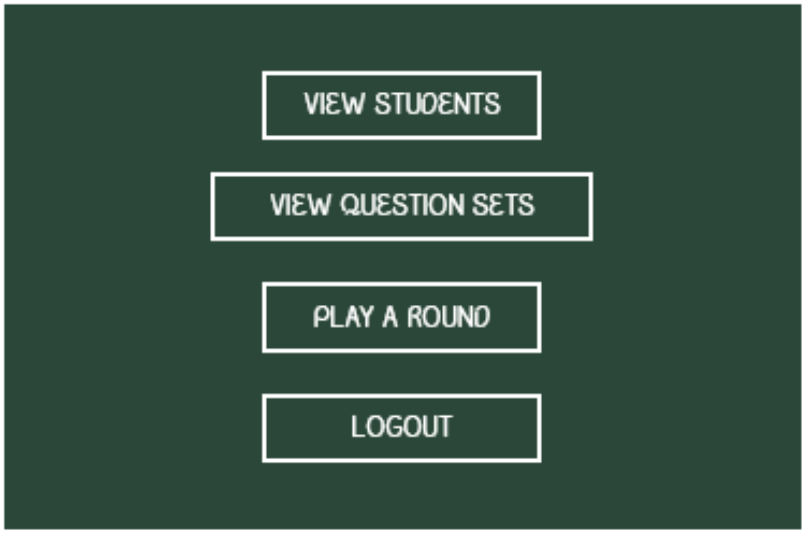
The teacher code is used to check that the person trying to add a teacher account is actually a teacher who has been given the code by the administrator for the program.

A drop down box is used for selecting the title as there are only a select number of possible titles that a teacher could have.

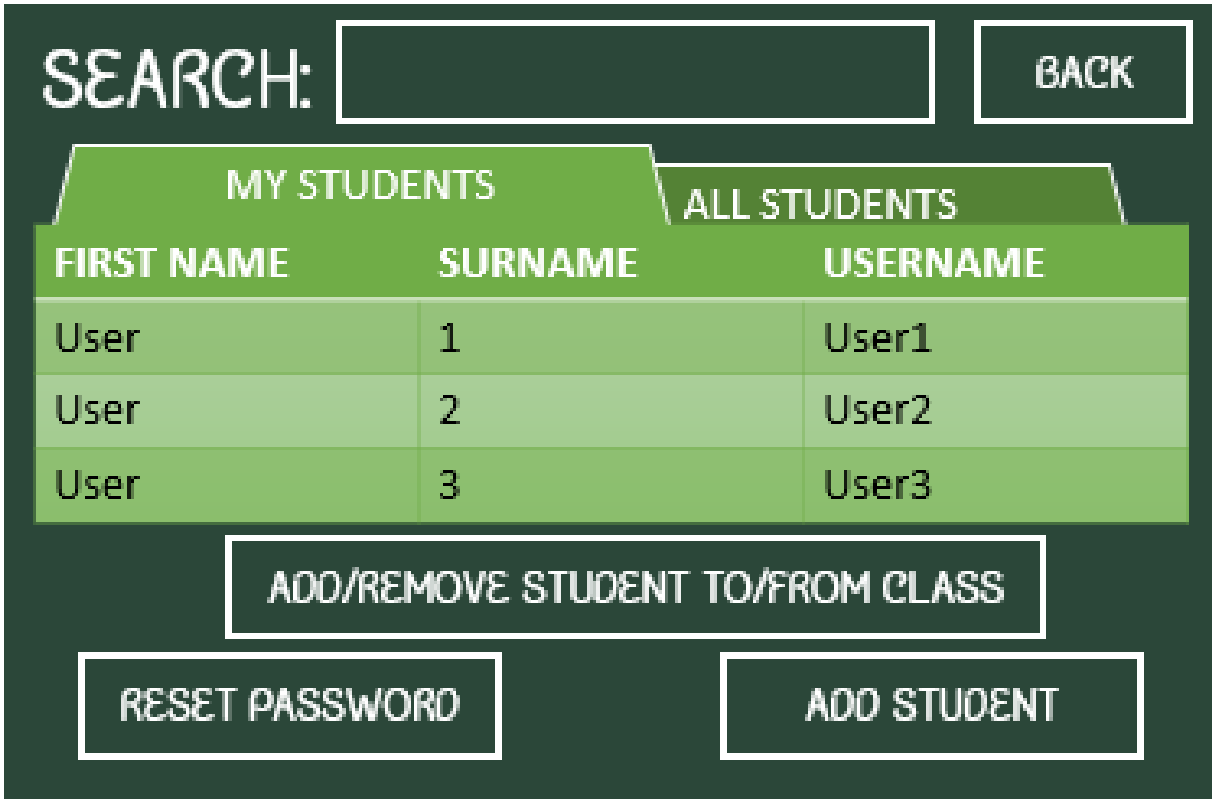
RESET PASSWORD

This is the screen for if the user is logging in for the first time since their account being created, or if they have asked a teacher to reset their password. This will open up as a smaller window on top of the program, where they will be able to insert a new password for their account. As long as their two passwords are identical in the two search boxes, they will be able to reset their password.

TEACHER'S HOME SCREEN



This is the screen that is shown to teachers when they sign into the program. I have designed four large buttons to direct teachers to where they want to go within the program.



This is the screen that shows when a teacher presses the View Students button. There will be a table in the middle of the screen, which will display all students, with their name, and username. The tabs on the left can be used for sorting between all users in the program, and just students that the teacher actually teaches.

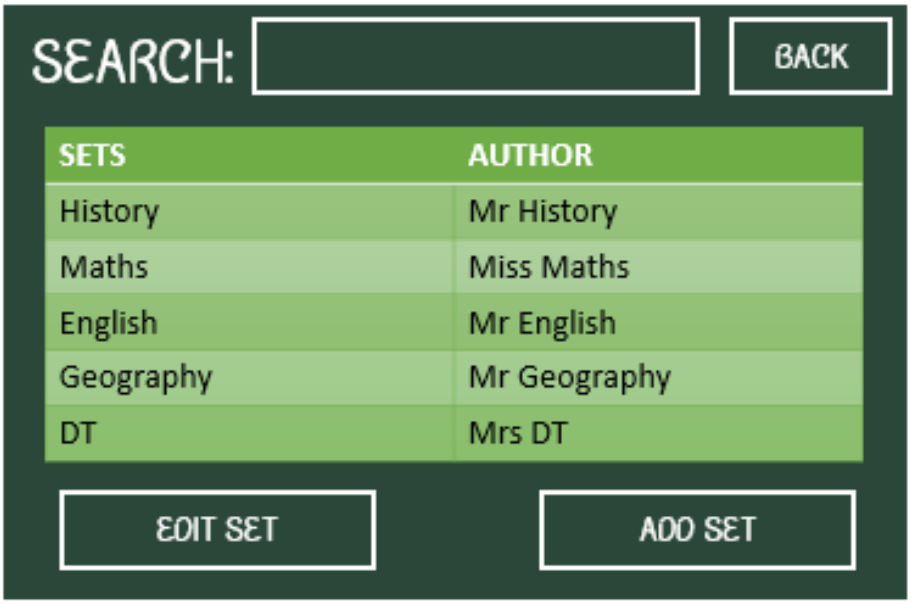
At the top of the screen there is a search bar that will narrow down the results shown in the table. There are also buttons that allow the teacher to add or remove students from their class based on which tab they are currently viewing.

There are two more buttons at the bottom of the screen that will allow teachers to reset the password of a selected student, and let them add a new student.

REGISTERING A NEW STUDENT

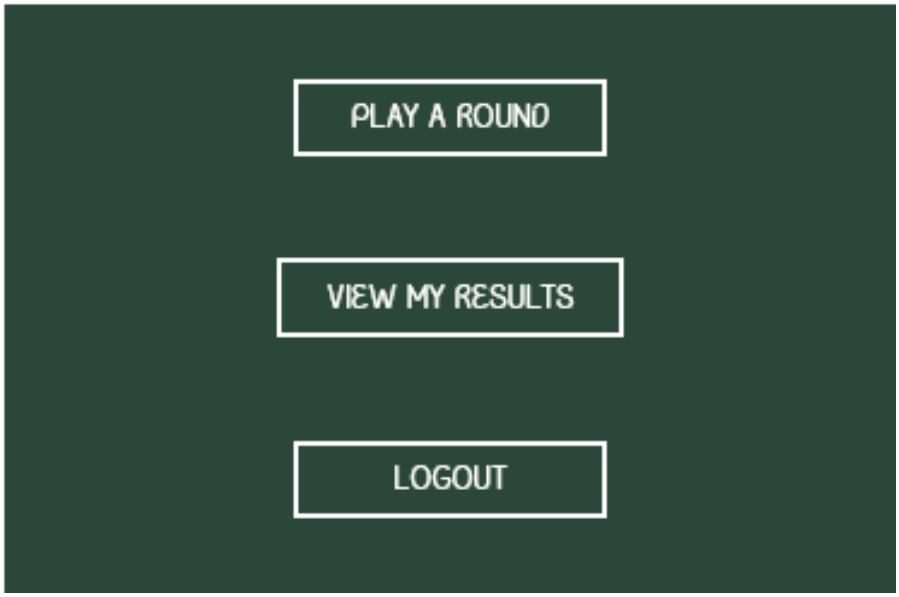
This screen allows teachers to add new students; there are three entry boxes for entering their username, first name and surname, and by pressing the Create button, the student will be added to the database and that teacher's class.

VIEWING SETS



This screen lets teachers view and search through all the sets created for the program, and at the bottom are two buttons that allow them to edit sets that they have created, or make a new set.

STUDENT'S HOME



This is the screen that students will be directed to once they login. There are buttons to let them play a round, view their results, and logout.

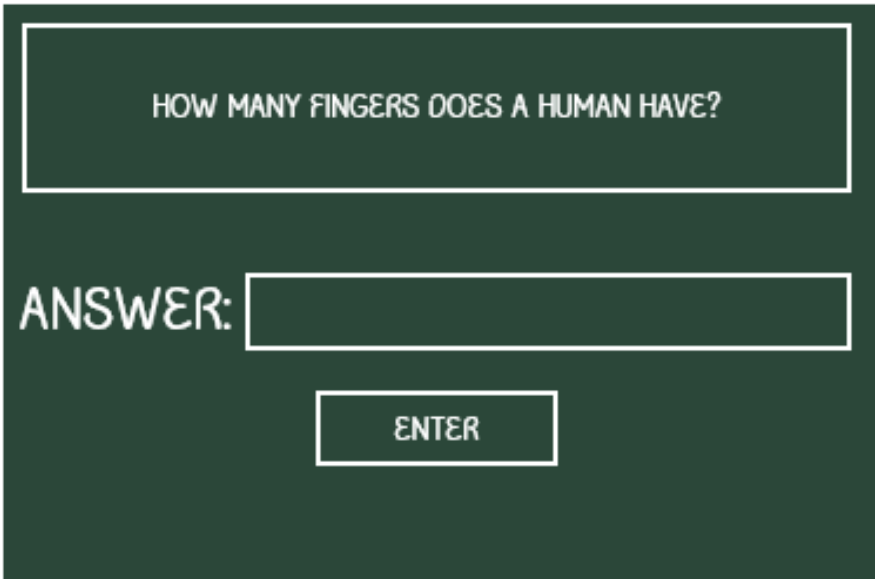
PLAY GAME MENU

This screen is for choosing a set to play. There is a search bar at the top for easier searching, a table in the middle for looking at all the possible sets, and then some options at the bottom to help the user decide how they want to learn, according to what suits them. The "?" button chooses a random set for the user to try.

GAMEPLAY SCREEN – MULTIPLE CHOICE

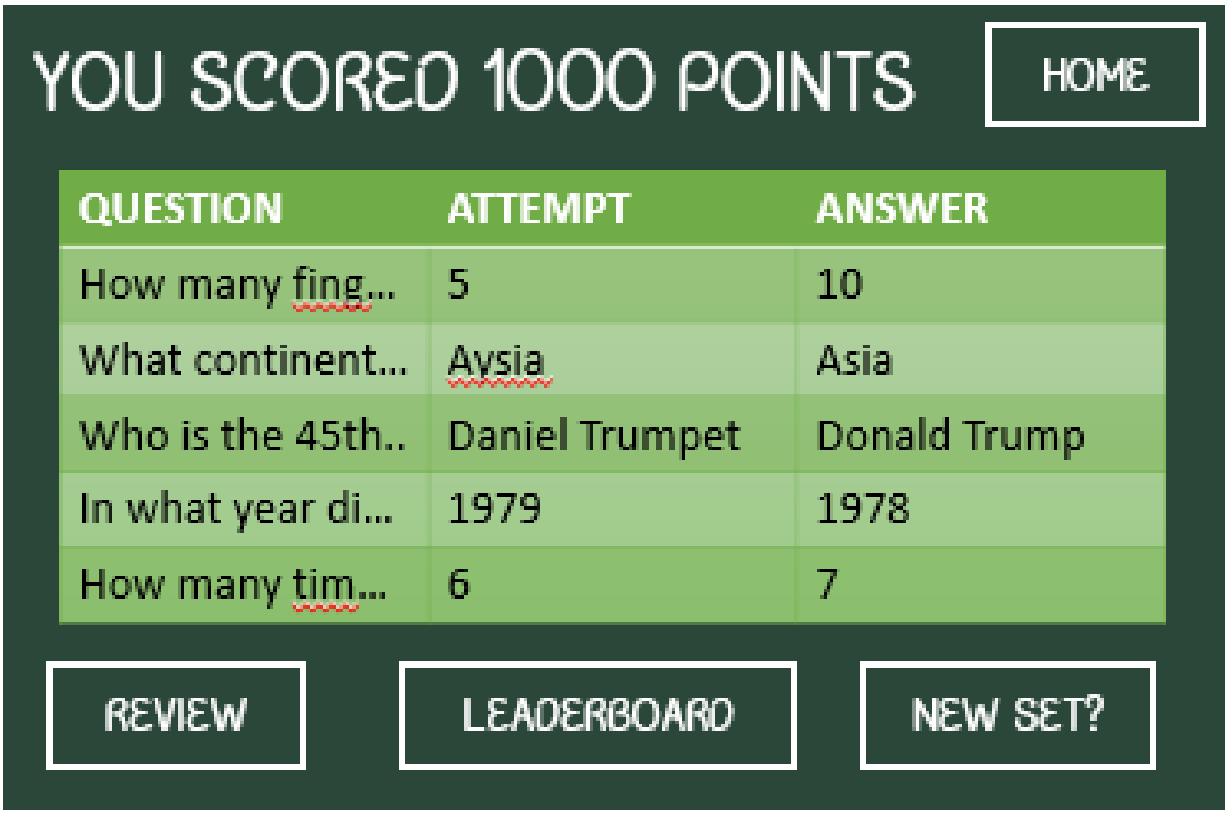
When a question is drawn from the database, it will be displayed in the box at the top of the screen so that it is easy to view for the user. The four possible answers for the question will be displayed in the four boxes beneath it, each being its own button. Once a button is pressed, the next question will be loaded into the same UI.

GAMEPLAY SCREEN – SINGLE ANSWER



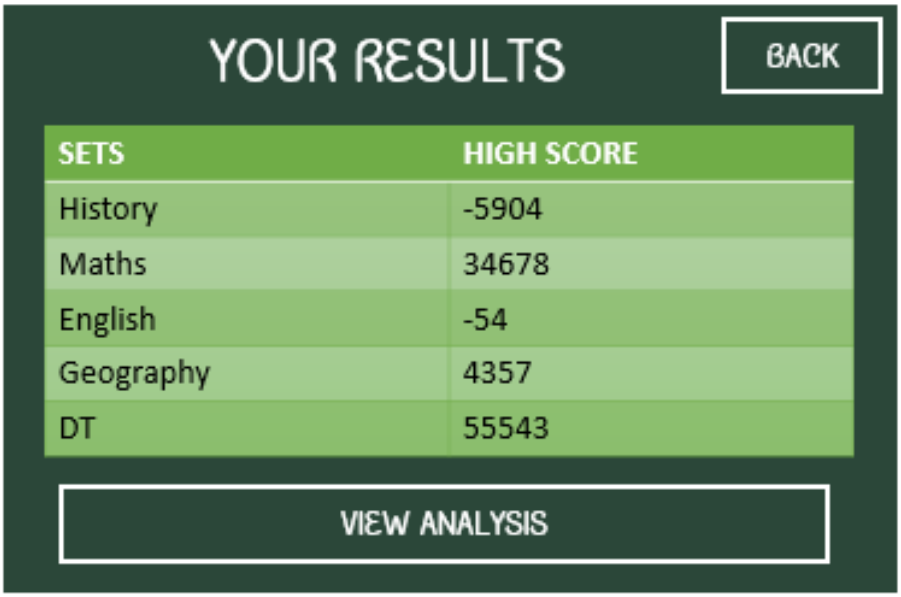
This is similar to the last mode, however instead of the buttons with the answers, there is a box for the user to type their answer, and then they can press the Enter button when they are ready to submit.

RESULTS SCREEN



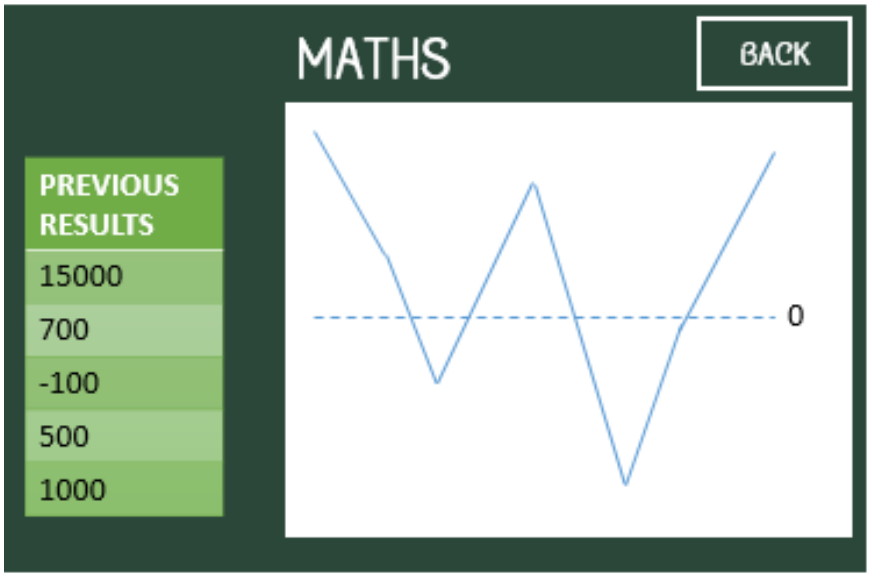
Once a user has finished their set they will be sent to this screen, where they will be shown their score and a table with all the questions they answered incorrectly, along with the correct answers. There are also three buttons at the bottom allowing them to play a new set, open a pop-up leader board with the top performing students ranked by their scores, or to review their progress over time.

VIEWING ALL RESULTS



This screen allows users to see their best scores for all the sets they've attempted, with the ability to use the button at the bottom to get a more in-depth review of their scores per set.

IN DEPTH RESULTS ANALYSIS



This screen will have a list of all the user's previous results in the given set, alongside a graph on the right which will display how they've progressed through each attempt of the set.

SUCCESS CRITERIA AND TEST PLAN

As I am creating my program I will need to test each function as I go, ensuring they work as intended and do not allow for any form of exploitation or cause a crash. In order to test my program efficiently as I go, I have devised a list of test data and requirements that will need to be met before my program is ready to be released as a product to my stakeholders. I have decided my tests based on the objective list that I created in my analysis section and the list goes as follows:

SETUP/GENERAL

Objective Number	What is being tested	How is it being tested	Special Type
1 & 3	Load PyQt UI files from a folder AND Generate the GUI from these files	I will create a python file and try and load one of my PyQt design files	Isolated Test
2	Load SQLite database from a file	I will create a python file and try and load the SQLite database and execute some SQL commands to see if it works as intended	Isolated Test
4	Have an easy-to-use and intuitive design throughout	As this is subjective, I will go to my stakeholders when I have a beta build and get their opinions	Post-Development
5	Load graphic resources from a file	When loading the PyQt files into my program, as I have decided I will use Resource Files, I will know if they are working correctly if my program looks as it does in the Designer application	None
6	Import necessary modules into python (e.g. time, hashlib)	I will import all the necessary modules and test a command from each to ensure they are all working	None
7	Use object oriented programming to create a class for each window in the program	I will create a class for the login window and check to see that all of its functions are working	None
8	Swap between windows seamlessly when a function is called to move between them	I will move between the login, registering teachers, and home screen windows to test that the transitions are fast and work properly	Extreme: Check to see if swapping between windows too quickly causes a crash or error
9	Allow user to input data into text boxes	On the login screen, I will check that the input boxes allow the user to enter text	Normal: Check that data can be entered and used in the program
10	Recognize double clicking a row of a table	In the Play Game screen, I will double click one of the question sets, and if working, it will begin the test	Extreme: Check if double clicking too many different rows quickly causes an error or crash
11	Working search bars to search through tables	In the Play Game screen, I will begin searching for a question set, by name and then author, seeing that it works correctly	Normal: Check that valid inputs generate a filtered list from a table Invalid: Prevent invalid characters being used in a search in the database for data security reasons
12	Validation for entry boxes throughout	I will be taking screenshots for every input box, ensuring it meets its own requirements for validation – For each input box, I will be testing both normal and invalid data	Normal: Check that valid inputs are accepted by the program Invalid: Ensure that any invalid input is prevented from being accepted into the program, and is identified to the user

12.1	Username Validation	Username must be checked as valid if it contains only alphanumerical characters, and is less than 15 characters long	Valid: "johnsmith" and "wooding123" Invalid: "john_smith", and "abcdefghijklmnop"
12.2	Password Validation	Password must be checked as valid if it contains only alphanumerical characters, at least one uppercase and lowercase letter, and is greater than 6 characters long	Valid: "Password" and "Welcome123" Invalid: "this is password","hi", and "h!9_js;"
12.3	Search Bar Validation	Any search bar inputs must only be validated for use in SQL queries if they do not contain quotation marks, apostrophes, or semi-colons	Valid: "Maths", and "Science Unit 1" Invalid: "Maths;" and "hi;"'"
12.4	Question Input Validation	Any question input must only be validated for being entered into the database if they do not contain quotation marks, apostrophes, or semi-colons	Valid: "What is pi?", and "9+2?" Invalid: "What is 'fracking'?", and "Question;"
12.5	First Name Validation	First name must be checked as valid if it contains only letters	Valid: "John", and "Matt" Invalid: "James99", and "hi123"
12.6	Surname Validation	Surname must be checked as valid if it contains only letters and may have 1 dash maximum for double barrelled surnames	Valid: "Smith", and "Jones-Smith" Invalid: "Smith12", and "Jones-N-Smith"

LOGIN SCREEN

Objective Number	What is being tested	How is it being tested	Special Type
13	When inputting a password, only display bullet points for security reasons	I will start typing text into the password box and see whether it covers the password with bullet points	None
14	Fetch usernames and passwords from the database when attempting to login	I will use the Python IDLE to print the data that is fetched from the database when the login button is pressed and check to see that it matches the database records	None
15	Hash passwords for security reasons	I will enter a password, and then print the hashed form of it to check that it is working	None
16	Be able to differentiate between a student and teacher account, moving the user to the according home screen	I will try to login as both a student and a teacher, and print the type of user account that is being logged into, and check it matches	None
17	Check if the user's password has been reset and if so, allow them to change it	I will login as a teacher and reset a student's password, then I will try to login as that student with the default password, and if working it will ask for me to change the password	None
18	When changing password, check that the new password is confirmed before editing the database records	I will attempt to press the change password button having typed in 2 different strings into the boxes, and see if it fails to change the password. Then I will try with the same strings and see that it writes the new password to the database	None
19	Provide an output when an account hasn't been found	I will enter a username that doesn't exist in the database and see if an output pops out	None
20	Provide an output when an entered password is incorrect	I will enter a correct username from the database along with an incorrect password and see if an output pops out	Invalid: Check that invalid inputs cause an output
21	Create a button to allow new teachers to be registered	I will press the button and see if it takes me to the Teacher Registration screen	Extreme: Check that pressing the button multiple times quickly doesn't add an account more than once.
22	When registering a new teacher ensure validation is working correctly	I will test all possible configurations of inputs to check the validation of adding a new teacher including, empty fields, taken usernames and incorrect teacher codes	Normal: Check that valid inputs are accepted by the program Invalid: Any invalid input, such as a taken username or weak password prevents any new account from being created.
23	Provide an output when the account has been successfully created	I will try to create a valid teacher account and see if an output pops out	None
24	Add new teacher accounts to the database	After creating a valid account I will check the database to see if the new user has been added	Normal: Check that accounts are successfully added when all inputs are valid Invalid: Check that accounts with invalid inputs are not added to the database

HOME SCREEN

Objective Number	What is being tested	How is it being tested	Special Type
25	Allow access to viewing all student accounts (Teachers Only)	I will press the view students button and see if it takes me to that screen	Extreme: Check that clicking multiple times only opens the window once
26	Allow access to viewing all question sets (Teachers Only)	I will press the view question sets button and see if it takes me to that screen	Extreme: Check that clicking multiple times only opens the window once
27	Allow access to playing a round	I will press the play a round button and see if it takes me to that screen (will test for teacher and students)	Extreme: Check that clicking multiple times only opens the window once
28	Allow access to view own results (Students Only)	I will press the view results button and see if it takes me to that screen	Extreme: Check that clicking multiple times only opens the window once
29	Allow user to logout	I will press the logout button and see if it takes me back to the login screen (will test for teachers and students)	Extreme: Check that clicking multiple times only opens the window once

VIEWING STUDENTS

Objective Number	What is being tested	How is it being tested	Special Type
30	Fetch all students' data from the database	I will print the fetched data from the SQL statement used to fetch all students' data	None
31	Generate a table of all students, displaying name and username	I will see if the table created is using the correct data from the database	None
32	Be able to filter by students that the teacher teaches	I will test the students of a teacher against the database to see if the lists are the same	None
33	Allow teachers to add/remove students to/from their class	I will attempt to add and remove different students from a teacher's class	Normal: Only new links can be created, and only existing links can be deleted Invalid: Prevent any non-existent links from trying to be deleted, and any existing links from being overwritten
34	Reset a student's password to a default one	I will select a student and reset their password, then I will try to login with that student and see if their old password stops working, and the program asks for them to change password	None
35	Ability to add new students to the database so long as a unique username is chosen	I will try to add a student with a username already in the database to see if it stops me, and then I will try a new username to see that it works	Normal: Check that only valid usernames, first names, and surnames are used when adding students Invalid: Prevent invalid inputs (numbers and punctuation in names, and too long usernames) from being added to the database
36	Show results of any students from a teacher's class	I will double click a student from that teacher's class and see if it takes me to their results screen	None

VIEWING RESULTS

Objective Number	What is being tested	How is it being tested	Special Type
37	Display a student's highest score in each of their attempted tests	I will check the database to find what the student's highest score in each test they've attempted is, and see if it matches the program	None
38	Allow access to in-depth analysis of progress in a given test	I will double click a question set and see if it takes me to the analysis screen. I will also press the button that should do the same to see if it works	Extreme: Check that clicking multiple times only opens the window once
39	Show a list of all scores ever received in a set, ordered by date	I will use the database to make a list of every result the student has ever had in a set and see if it matches the program	None
40	Generate a line graph showing scores over time	Using JSBin I will create a line graph using some pre-defined data to see that it works before implementing it into the program. After implementation I will test it again with data from the program	Isolated Test
41	Scale the graph depending on how many results the student has and the range of the scores	Similar to the previous test, I will try in JSBin before adding it to the program, then I will try different ranges of data to see if it scales correctly	Extreme: Try using a large number of points and check that it doesn't affect performance too badly
42	Show a specific selected score on the graph	I will click one of the scores and see if the selected score is shown on the graph	None

VIEWING QUESTION SETS

Objective Number	What is being tested	How is it being tested	Special Type
43	Fetch all question sets and their authors, and display in a table	I will print the fetched data from the SQL statement used to fetch all the question sets and authors	None
44	Allow access to adding new sets	I will press the add sets button and see if it takes me to that section	Extreme: Check that clicking multiple times only opens the window once
45	Have a browser for selecting a file from storage	I will create a test file and see if selecting it with the browser works correctly to get me the location of it in storage	Normal: Check that valid files can be selected such as text and csv files Invalid: Check that invalid files cannot be selected such as jpeg or mp4 files
46	Import contents of file into database	I will try to import the test file, generating a new question set in the database, I will then try to play the set to make sure it works	Extreme: See how large a file can be imported without causing performance issues or crashing
47	Allow access to editing sets	I will press the edit sets button and see if it takes me to that section	Extreme: Check that clicking multiple times only opens the window once
48	Allow user to upload file with updated/edited questions	I will edit some of the details from the test file, and try to select it again with the browser, then I will see when playing if the set has been updated	Normal: Allow only valid details to be updated to the database Invalid: Ensure no invalid questions or answers can be uploaded
49	View a leaderboard of everyone who's attempted a quiz	I will see if the leaderboard matches with the results I will check in the database	Extreme: See how many users can attempt a quiz and be featured on the leaderboard without a crash
50	Allow leaderboard to be printed	I will attempt to print a leaderboard from the program	None

PLAY A ROUND

Objective Number	What is being tested	How is it being tested	Special Type
51	Fetch all question sets and their authors	I will print the fetched data from the SQL statement used to fetch all the question sets and authors	None
52	Create a table displaying all sets and authors	I will check that the table matches the data from the database	None
53	Create a checkbox for whether a user wants to play multiple choice	I will try to play a game with the checkbox checked, and then unchecked, and make sure they start the test in easy mode, and hard mode respectively	None
54	Allow playing a selected quiz by pressing 'Play'	I will select one set with a single click, then press the "Play" button	Normal: Only one question set should be selectable at a time Extreme: Ensure pressing play more than once doesn't start multiple quizzes

GAMEPLAY

Objective Number	What is being tested	How is it being tested	Special Type
55	Fetch all questions from database for the given set	I will print the fetched data from the SQL statement used to fetch all the questions and check that it has fetched every question for that set	None
56	Randomly select a question from the pool and then remove it from the pool	I will check the order that the questions are in from the database, and check that the order in which they appear in the program is different each time	None
57	Display the question in a text box	I will see if the full question has been set in the text box	None
58.1	In hard mode, generate an input box	I will see when playing hard mode if an input box is generated, and I will try to type into it to make sure it is working correctly	None
58.2	In hard mode, allow entry of answer by pressing enter or pressing 'Play'	I will enter an answer and then press the enter button to see if it takes me to the next question, I will do the same with the "Play" button	None
59.1	In easy mode, shuffle the 4 possible answers	I will check the database to see the order of the 4 answers for a question, and then I will print the order of the answers from the program	None
59.2	In easy mode allocate each answer to a button	I will check that the buttons have the same text as the answers for a specific question	None
59.3	In easy mode, question moves on when a button is pressed	I will click one of the button and see that it moves to the next question	Extreme: Ensure pressing a button multiple times only moves forward one question
60	When a question starts, start a timer	I will print the time at which the timer is started when a question starts	None
61	When a question is answered, stop the timer	I will print the time at which the timer is stopped and see that the elapsed time is correct	None
62	Generate a number of points based on elapsed time (kn where k is a constant and n is elapsed time)	I will print elapsed time for each question and check that the formula is working correctly	None
63	If answer is correct, add score to total	I will print the score for each question in the set to see that it is adding points when I get questions correct	None
64	If answer is incorrect, deduct score from total	I will print the score for each question in the set to see that it is deducting points when I get questions incorrect	None
65	When there are no more questions left, go to the results screen	I will check to see that the number of questions I answer before going to the results screen is the number of question in that set in the database	Extreme: Check that clicking the final answer multiple times only opens the results window once

RESULTS SCREEN

Objective Number	What is being tested	How is it being tested	Special Type
66	Display total score on the screen	I will time myself using a stopwatch to make sure that the timings are accurate and so the score is correct	None
67	Display all incorrectly answered questions in a table along with the user's attempt and the correct answer	I will attempt a set multiple times, noting which answers I am getting wrong and check that they are the same ones in the table at the end	None
68	If not all text fits in a row, allow user to double click to view in a separate box	I will double click a row and see that it outputs the correct data	Normal: Allow only one row to be selected at a time Extreme: Ensure clicking a row more than twice doesn't open up more than one separate box
69	Fetch the top 10 unique scores for the quiz	I will check that the top 10 unique scores from the database are the same as the ones shown in the leaderboard	None
70	Display scores on a leaderboard	I will check that the layout and order of the leaderboard is correct	None
71	Allow user to review their progress of the set if they've attempted it more than once	I will try to review progress having only attempted the set once, and see if it prevents me from reviewing, and then I will try again when I've completed the set more than once to check that it lets me in	None
72	Create a button to let the user return to the play screen	I will press the button and make sure that it takes me to the correct screen	Extreme: Check that clicking multiple times only opens the window once
73	Writing scores to the database	I will run an insert statement to add scores to the scores table in the program, an then I will check the database to see if the score was added successfully	None

TYPES OF TESTING

For some of the criteria I have outlined above, I will need to apply different styles of testing to ensure that the program runs well and without any crashes or errors. I have outlined a few different types of tests that I will be going through during my development, and afterwards.

VALIDATING USER INPUTS

In my program there will be a fair number of entry boxes where users will be able to input data, such as their username or password, and this data will either be used in SQL statements, or for processing in Python. The types of data that I will be testing are:

- Normal Data – I will be testing data that should work in the program to check that it does in fact provide the results I expect, such as strong passwords and valid usernames.
- Extreme Data – I will be testing data that although is valid in the program, will try to push the program to its limits, such as adding question sets with a large number of questions, and plotting a large number of points on a graph. This section is mostly about testing the time complexity for the program and making sure it can keep up with large amounts of processing.
- Invalid Data – I will be testing data that I see unfit for the program after I have added the necessary validation, to check that the program is correctly preventing any invalid data from making changes to the database, or crashing the program.

INTERFACE AND SYSTEM TESTING

In order to make sure that my program can handle any kind of user input, I will need to ensure that the program can still run without crashes or multiple window executions when users try to press a button too many times, or too quickly in succession. Other things such as making sure they can't crash the program by trying to add a large number of rows to a table, and trying to cheat the quiz section by making changes to system clocks.

POST-DEVELOPMENT TESTING

During the development of my program, I will be testing everything that I have described above as I go, however there will be certain elements of my success criteria that I will not be able to test until I have finished development, such as the subjective criteria like whether the interface is easy to use, as that will require me going back to the RGS and providing the program to my audience and listening to feedback.

Additional testing such as testing the program runs well on different types of machines (different processor architectures, operating systems, and specifications to name a few) will need to take place after I have developed the prototype so that I can ensure that the program will be able to scale as more users start utilising the application.

DEVELOPMENT

Having designed each of the sub-programs within my program, I will begin developing each section at a time, getting them all feature complete and test them thoroughly before merging them all into a single program. Below I will document my progress as I go, talking about how the program works, why I've designed it the way I have, and any issues that arise and how I plan to overcome them.

SETTING UP AND USING THE DATABASE

Before beginning the process of coding the program I have created the database that will be the backend of the program. I have built the database using SQLite, and its design is exactly the same as the plan I laid out in the Design section. I have also taken the liberty of adding some sample data into the database just so that I am able to test features as I build the program.

Name	Type	Schema
Tables (8)		
QSLinks		CREATE TABLE "QSLinks" (`SID` INTEGER, `QID` INTEGER)
Questions		CREATE TABLE "Questions" (`QID` INTEGER, `Question` TEXT, `Correct` TEXT, `Wrong1` TEXT, `Wrong2` TEXT, `Wrong3` TEXT, PRIMARY KEY(QID))
STLinks		CREATE TABLE "STLinks" (`TUser` TEXT, `SUser` TEXT)
Scores		CREATE TABLE "Scores" (`Username` TEXT, `SID` INTEGER, `Score` INTEGER, `Date` TEXT)
Sets		CREATE TABLE "Sets" (`SID` INTEGER, `SetName` TEXT, `Author` TEXT, PRIMARY KEY(SID))
Students		CREATE TABLE "Students" (`Username` TEXT, `Password` TEXT, `FirstName` TEXT, `Surname` TEXT, PRIMARY KEY(Username))
Teachers		CREATE TABLE "Teachers" (`Username` TEXT, `Password` TEXT, `Title` TEXT, `Surname` TEXT, PRIMARY KEY(Username))

Table: Students

	Username	Password	FirstName	Surname
	Filter	Filter	Filter	Filter
1	woodingmp	af808b049bef56...	Matthew	Wooding
2	WillTaylor	5e884898da280...	Will	Taylor

Table: Scores

	Username	SID	Score	Date
	Filter	Filter	Filter	Filter
1	WillTaylor	3	12315	2017-03-10 10:5...
2	woodingmp	3	-79486	2017-03-10 10:5...
3	woodingmp	3	14786	2017-03-10 10:5...
4	woodingmp	3	-105075	2017-03-10 10:5...

Table: Questions

	QID	Question	Correct	Wrong1	Wrong2	Wrong3
	Filter	Filter	Filter	Filter	Filter	Filter
1	1	TEMP	TEMP	TEMP	TEMP	TEMP
2	2	For which genr...	Cookery	Reality TV	Teleshopping	Talk Show
3	3	In which countr...	Netherlands	United Kingdom	United Sates	Australia
4	4	In Monty Pytho...	Ni	Aye	Ooh	Ah
5	5	On a film set w...	Boom	Shotgun	Radio	Splash
6	6	Who is the dru...	Mick Fleetwood	Dave Grohl	Ringo Starr	Travis Barker
7	7	What actor play...	Robert Downey...	Benedict Cumb...	Jude Law	Tom Hiddleston
8	8	What 2013 spac...	Gravity	The Martian	Interstellar	Prometheus
9	9	The 2010 film T...	Facebook	Twitter	Digg	eBay
10	10	Which actor ha...	Chris Evans	Chris Hemsworth	Mark Ruffalo	Jeremy Renner
11	11	What Black Eye...	I Gotta Feeling	Meet Me Half ...	My Hump	Pump It
12	12	Which House a...	Hugh Laurie	Jesse Spencer	Olivia Wilde	Lisa Edelstein
13	13	What former b...	Mike Tyson	George Foreman	Lennox Lewis	Vitali Klitschko
14	14	What Hollywoo...	Paramount	Universal	Lionsgate	Disney
15	15	What is the birt...	Kal-El	Clark	Jor-El	Tom

Table: QSLinks

	SID	QID
	Filter	Filter
1	1	1
2	2	33
3	3	2
4	3	3
5	3	4
6	3	5
7	3	6
8	3	7
9	3	8
10	3	9
11	3	10
12	3	11
13	3	12
14	3	13
15	3	14

Table: Teachers

	Username	Password	Title	Surname
	Filter	Filter	Filter	Filter
1	theteacher	5e884898da280...	Mrs	Teacher
2	teacher2	5e884898da280...	Mr	Teacher

Table: STLinks

	TUser	SUser
	Filter	Filter
1	theteacher	woodingmp
2	theteacher	WillTaylor

Table: Sets

	SID	SetName	Author
	Filter	Filter	Filter
1	1	TEMP	TEMP
2	2	Set	theteacher
3	3	Entertainment	theteacher

In order to test using SQLite and interacting with the database, I have created a test file to try and connect to and use the database.

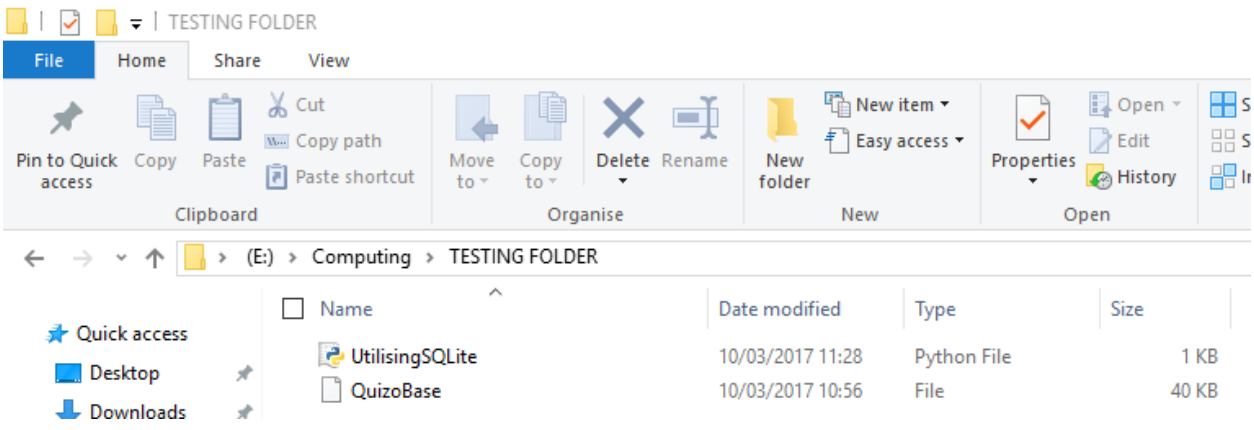
CONNECTING TO A DATABASE

I've created a python script which attempts to connect to the database, which is saved in the same folder. I will try accessing it from the same folder, and then try to do it when the folder is not stored locally, as will be the case when user's use the full program. If the database is connected to, the program will print "FOUND", otherwise it will print "NOT FOUND" to the IDLE. I have also added a SQL statement to select all data from the "Sets" table, and to print all the fetched data. If the database is connected to successfully, the following data should be printed.

Table: Sets

	SID	SetName	Author
	Filter	Filter	Filter
1	1	TEMP	TEMP
2	2	Set	theteacher
3	3	Entertainment	theteacher

1ST TEST – SAME FOLDER



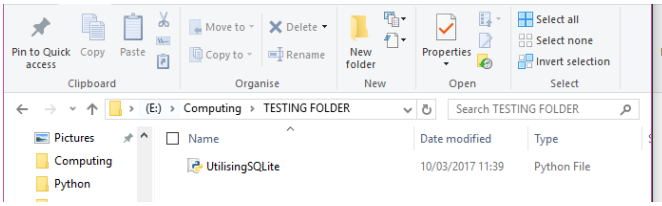

```
## Utilising the SQLite Database

import sqlite3 as lite

try:
    connect = lite.connect('QuizoBase')
    cursor = connect.cursor()
    print("FOUND")
    query = "SELECT * FROM Sets;"
    cursor.execute(query)
    print (cursor.fetchall())
except:
    print("NOT FOUND")
```

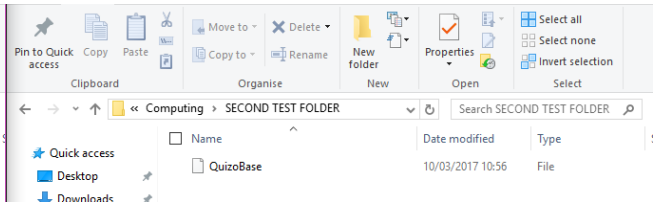
```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 20:20:57) [MSC v.1600 64 bit (AMD64
)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: E:\Computing\TESTING FOLDER\UtilisingSQLite.py ==========
FOUND
[(1, 'TEMP', 'TEMP'), (2, 'Set', 'theteacher'), (3, 'Entertainment', 'theteacher')]
>>>
```

Test was successful and the database was located when stored in the same folder as the code being executed. The correct data was fetched and printed from the database.

2ND TEST – DIFFERENT FOLDER

For this test the database and the python file will be stored in different locations, and the same code will be used.






```
## Utilising the SQLite Database

import sqlite3 as lite

try:
    connect = lite.connect('E:\Computing\SECOND TEST FOLDER\QuizoBase')
    cursor = connect.cursor()
    print("FOUND")
    query = "SELECT * FROM Sets;"
    cursor.execute(query)
    print (cursor.fetchall())
except:
    print("NOT FOUND")
```

```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 20:20:57) [MSC v.1600 64 bit (AMD64
)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: E:\Computing\TESTING FOLDER\UtilisingSQLite.py ==========
FOUND
[(1, 'TEMP', 'TEMP'), (2, 'Set', 'theteacher'), (3, 'Entertainment', 'theteacher')]
>>>
```

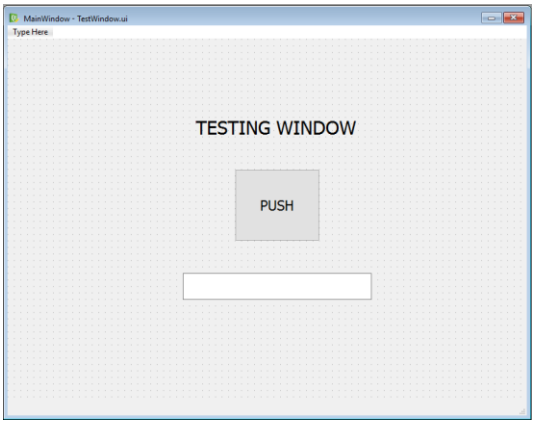
Test was successful and the database was located even in a different folder, fetching the correct data from the database.

OBJECTIVE 2 SUCCESSFUL – PROGRAM CAN CONNECT TO A DATABASE FROM A FILE

CONNECTING AND CREATING A GUI WITH PYQT

Before starting work on my program's interface, I am going to create a test window to make sure I know how to implement PyQt into my program correctly. As I also want my program to have a theme and look aesthetically pleasing, I am going to test some fonts and backgrounds until I find something I like.

Using the QtDesigner, I have created a simple window with a title, button, and text box. I will try to load this window in Python and I will have some basic functionality so that when you press the button, it shows how many times it has been pressed in the text box.



Having done some researching into PyQt and how to use it with Python, I have created the below code to try and use the window as I intend it to.

As I have PyQt Version 4 installed, I Have imported the necessary modules from PyQt4; QtCore, QtGui, and uic. I have then loaded the ui file to a class, which is used to create the GUI for the window. I have found that each window that I create for my program will need its own class, and each class needs the code shown to initialise.

Using the PyQt Sourceforge site, which explains all the different methods for PyQt widgets and windows, I found the necessary method for checking for when a button is clicked, and then I have created a function which it runs when clicked. The function simply adds 1 to the total number of clicks, and then displays this in the line edit box.

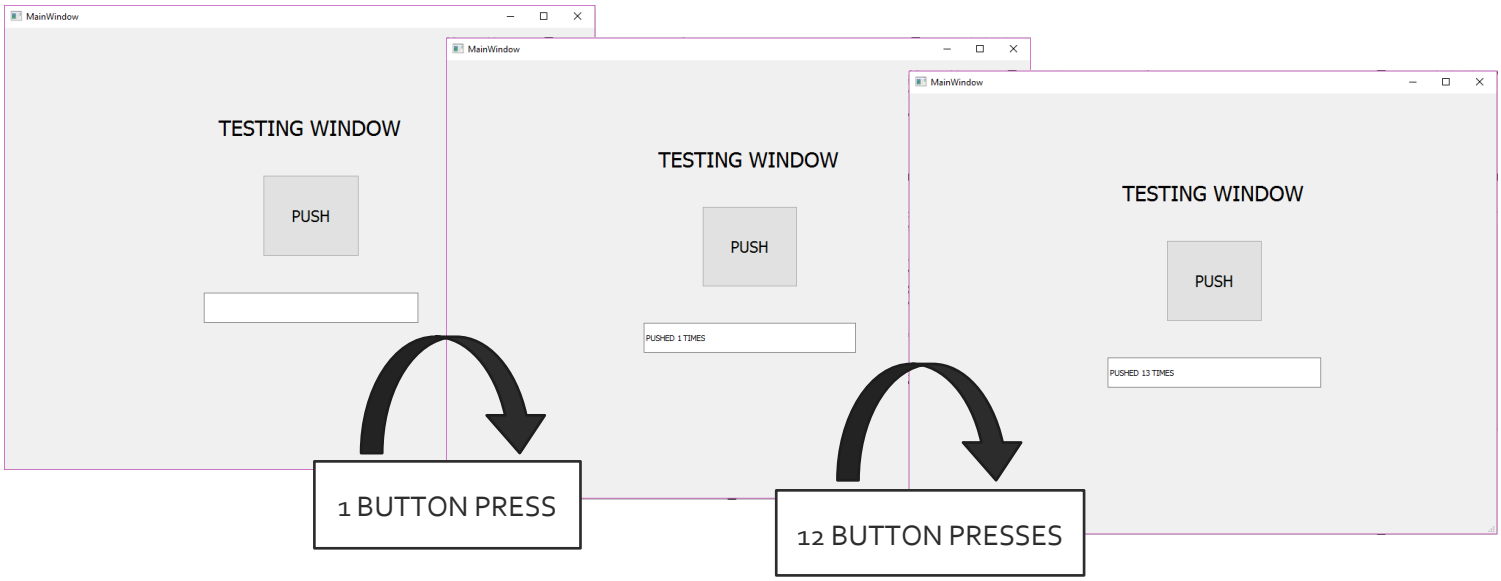
TestingPyQt.py - E:/Computing/TESTING FOLDER/TestingPyQt.py (3.4.4)

File Edit Format Run Options Window Help

```
## PyQt Testing

import sys
from PyQt4 import QtCore,QtGui,uic

Test_class = uic.loadUiType("TestWindow.ui")[0]

class TestWindow(QtGui.QMainWindow,Test_class):
    def __init__(self,parent=None):
        global times
        QtGui.QMainWindow.__init__(self,parent)
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.setupUi(self)
        self.pushButton.clicked.connect(self.pushed)
        times = 0

    def pushed(self):
        global times
        times += 1
        self.lineEdit.setText("PUSHED "+str(times)+" TIMES")

app = QtGui.QApplication(sys.argv)
Test_Window = TestWindow(None)
Test_Window.show()
app.exec()
```

The final 4 lines of code are for starting the application and displaying the window.

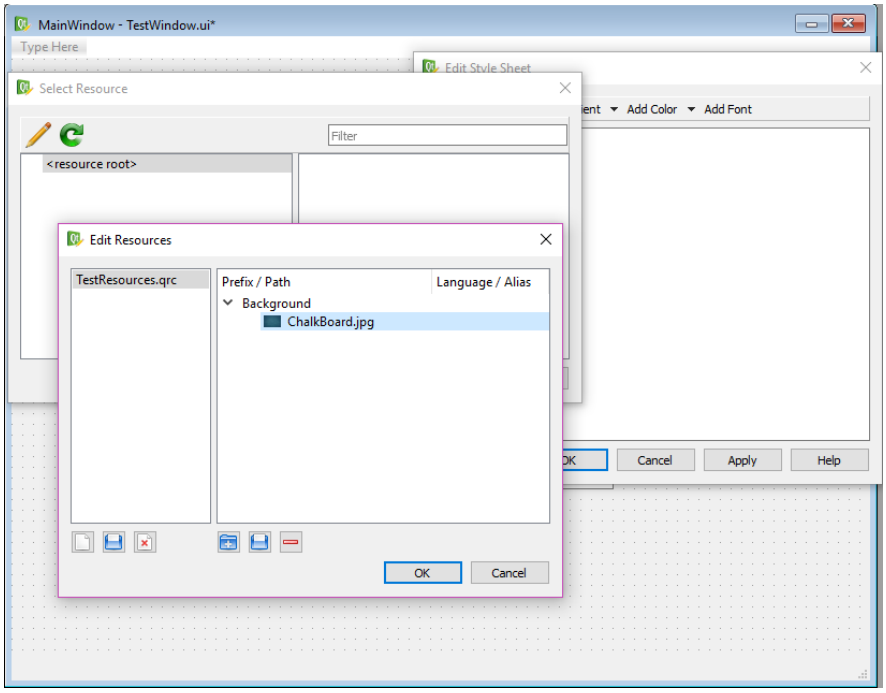
The results of executing this program can be seen below, and as hoped, the program runs as expected.



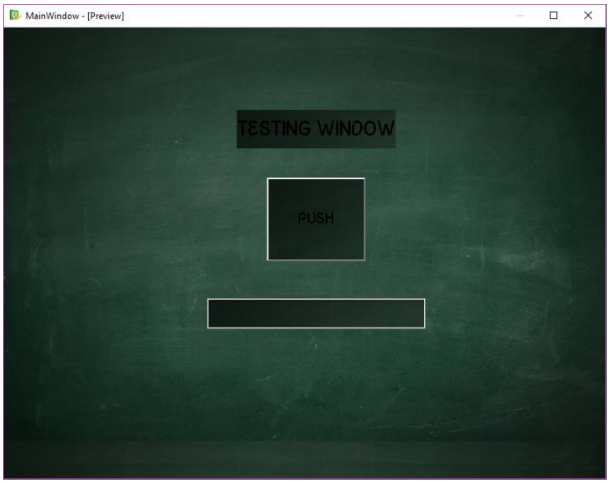
OBJECTIVES 1 AND 3 SUCCESSFUL – PROGRAM CAN LOAD AND SHOW PYQT UI FILES

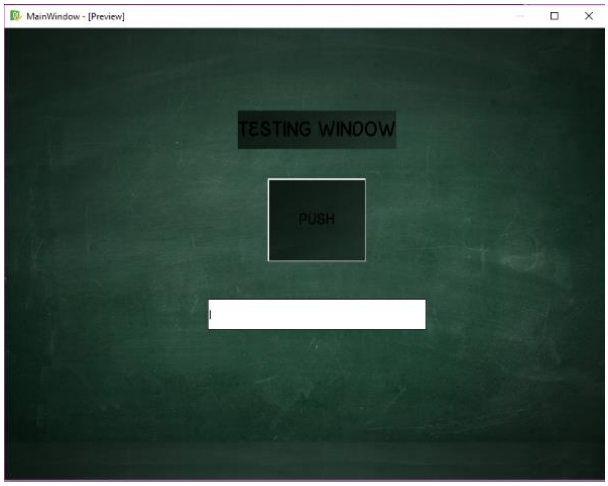
Having though about the design of my program, I think a retro chalkboard look could be quite aesthetically pleasing. Based on this idea, I have found a green chalkboard background image for my program, and found a chalk font that I think could look good with it. Before trying to implement these into my actual program, I think it would be best to test backgrounds and using resource files on a test file.

Setting the font of all the widgets was simple as I could just select it from the list of system fonts, however adding a background was not as simple: in order to use external images, I have found that a resource file needs to be created. In this resource file I was able to add the background image, and then I could set the background image of the window in its Stylesheet.

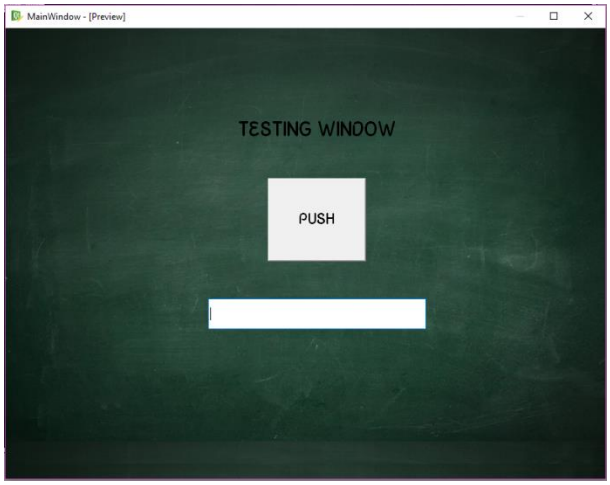


Using QtDesigner's "Preview" mode I was able to get an idea of how the window would like when used in a program, however it wasn't as clean and aesthetically pleasing as I would have hoped. Each of the widgets seems to inherit the same background as the window and so everything looks a bit clunky. Additionally, the line edit box has some strange flashing when trying to mouse over it which I will need to look into.



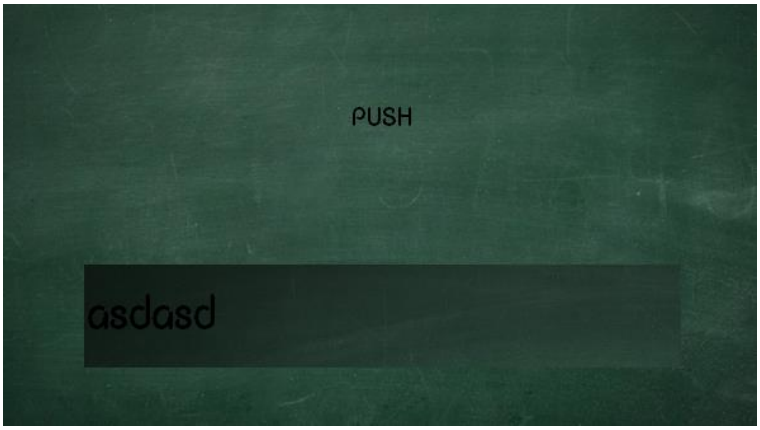


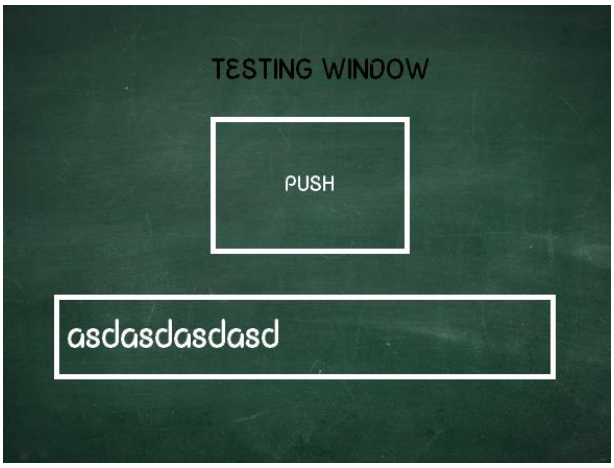
To try and prevent this I have created a blank png, with the idea being to set the background of the individual widgets to this blank png, to make them have a transparent background. Having attempted this, I can say that for the label widget it worked as intended, but for the button and line edit, it has just given them plain white backgrounds, and I will need to find another method.



Having gone through some of the settings for the button and line edit widgets, and with a bit of trial and error, I have found that I can make the button

"flat" which removes all background from the button. However, by doing this I can't find a way to give the widget a border which is necessary for a button so the user knows where they can click. Additionally, I have found that by setting a border image for the line edit, it stops any of the flashing from mousing over it, however I can't get its background transparent, and as with the button, has no border.



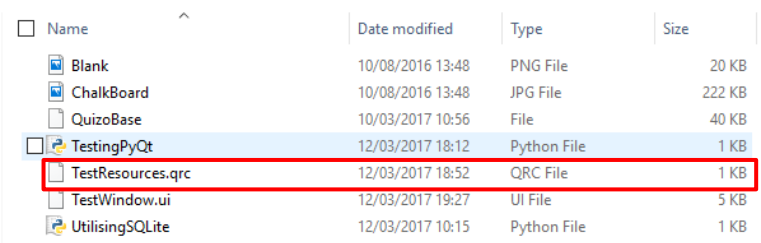


Seeing as I can't get a border for these widgets, my next idea was to try putting a frame around the objects, and then giving the frame a border, making it look like the widgets have their own borders. As shown to the left, using the frame method, and giving the frame a transparent background has worked to create the aesthetic I was after. By setting the colour of the frame to white, it also changed the text colour to white, and now seeing the white side by side with the black font, I have decided that the white font looks better with the green background.

Having now reached the aesthetic I was looking for within the QtDesigner preview, I will now test the new look when running the python program to make sure the looks are the same. However when attempting to launch the program, this error message appeared in my IDLE:

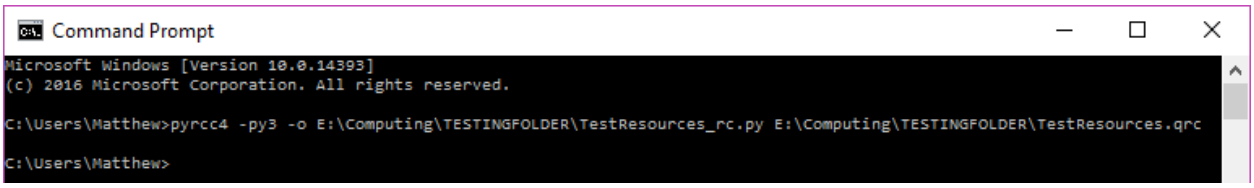
```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 20:20:57) [MSC v.1600 64 bit (AMD64)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>>
============ RESTART: E:/Computing/TESTING FOLDER/TestingPyQt.py ============
Traceback (most recent call last):
  File "E:/Computing/TESTING FOLDER/TestingPyQt.py", line 6, in <module>
    Test_class = uic.loadUiType("TestWindow.ui")[0]
  File "C:\Python34\lib\site-packages\PyQt4\uic\__init__.py", line 211, in loadUiType
    exec(code_string.getvalue(), ui_globals)
  File "<string>", line 93, in <module>
ImportError: No module named 'TestResources_rc'
>>>
```

This error appears to be linked to the resource sheet that I created for the interface, and the error being that there is no file named 'TestResources_rc' in the folder. The only file with a similar name to this is 'TestResources.qrc' which was created when I first created the resource file.

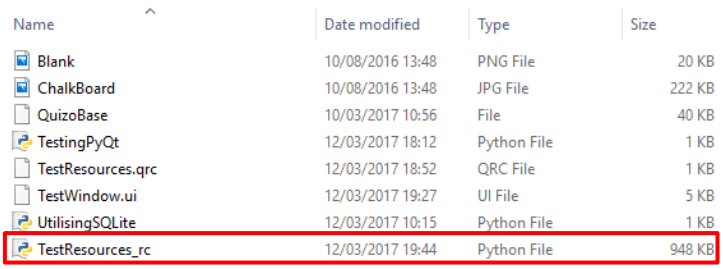


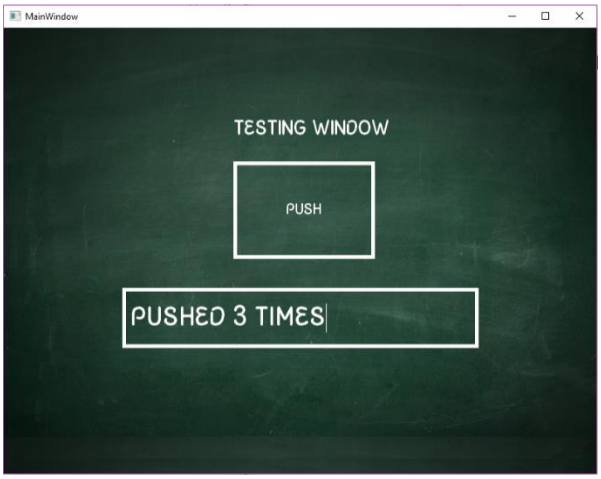
Having searched online for a solution, I have found that the fix for this issue is to write some command line code which will convert the qrc file into a python module that can be used by python code. The code that I found online for this fix is:

```
pyrcc4 -py3 -o E:\Computing\TESTINGFOLDER\TestResources_rc.py E:\Computing\TESTINGFOLDER\TestResources.qrc
```



Having run this code in the command prompt, the new file has been created, and when I now try to run the python program, the correct interface is created, with the background, font, and design.





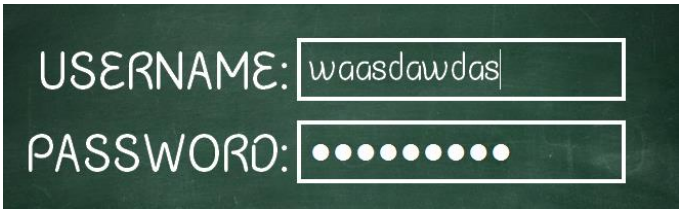
Now that I have been able to load my UI file into python along with a resource file, I feel ready to begin coding my actual program.

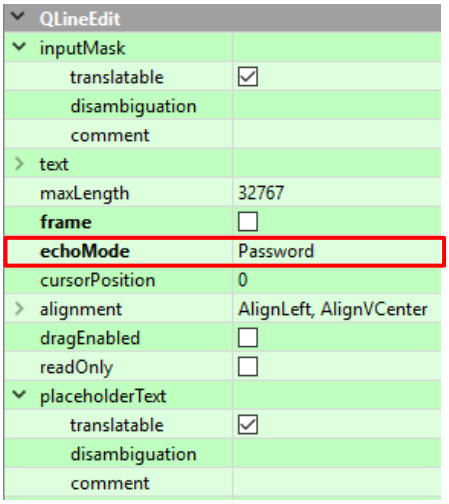
OBJECTIVE 5 SUCCESSFUL – RESOURCE FILE IMPORTED INTO PYTHON

CREATING THE LOGIN SCREEN

Using my original design for login screen from the design section, I have created a similar design in the QtDesigner to be the first screen the user sees when they start up the application.

This login screen has a title, two line edits for inputting username and password, and two buttons for logging in or registering a new teacher. For security reasons, the password field needs to be hidden when typing into it, to prevent anyone other than the user knowing their password. Luckily PyQt has this function built in which displays all entered text as a series of bullet points.





OBJECTIVE 13 SUCCESSFUL – PASSWORDS ARE HIDDEN IN TEXT BOXES

PASSWORD HASHING

For security reasons, when creating a database with user accounts, passwords should not be stored in their raw format, i.e. in plain text. For this reason, I will be using hashing as a way of storing passwords in the database in an encrypted format. Python has a module called 'hashlib' which can achieve hashing. I will then store the hashed passwords in the database, and then when a user tries to login, the program can hash their attempt and check to see if the attempt and the password are the same. To show that the hashing works, I have created a test program which will take a user's input, hash it, then

take another input, hash that and then check whether the two are the same, to simulate checking a password against a user's attempt. To the side is my code for the test.

```
## Hash Testing

import hashlib

userword = input("Enter password:")
userattempt = input("Re-Enter password:")
hashedword = hashlib.sha256(userword.encode()).hexdigest()
hashedattempt = hashlib.sha256(userattempt.encode()).hexdigest()
print(hashedword,hashedattempt)
if hashedword == hashedattempt:
    print("Passwords match!")
else:
    print("Passwords do not match!")
```

To test the hashing, I did 3 tests to make sure it was matching passwords correctly. For the first test I entered the same word but with the first input uppercase, and the second lowercase. As expected, the hashed passwords did not match, ensuring that passwords will be case sensitive in my program. The second test was inputting the same word but having extra spaces, which also resulted in an unmatched result. Finally I tested the exact same string "One" and as hoped, the passwords were matched. The results of the 3 tests can be seen beside, and from it I have decided that this will be the form of hashing I will be using to secure passwords for my program.

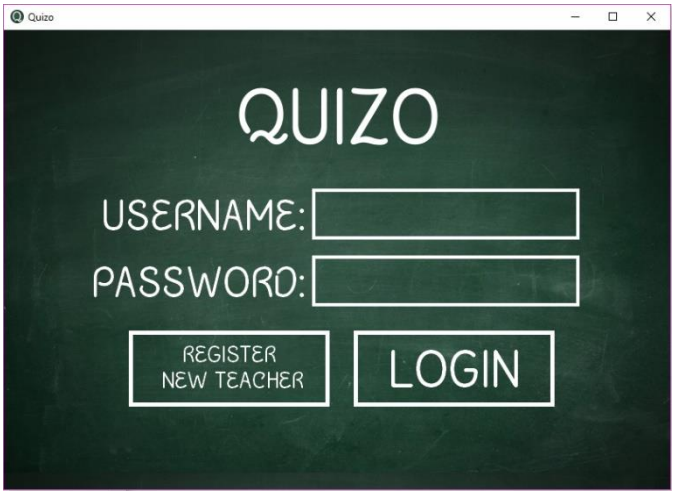
```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 20:20:57) [MSC v.
1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: E:/Computing/TESTINGFOLDER/HashTesting.py
==========
Enter password:One
Re-Enter password:one
8b12507783d5becacbf2ebe5b01a60024d8728a8f86dcc818bce699e8b3320bc
7692c3ad3540bb803c020b3aee66cd8887123234ea0c6e7143c0add73ff431ed
Passwords do not match!
>>>
========== RESTART: E:/Computing/TESTINGFOLDER/HashTesting.py
==========
Enter password:o n e
Re-Enter password:one
94f0f241cbae6931637ccc812621d3917c2ded1db491bff997a7e025bd698faa
7692c3ad3540bb803c020b3aee66cd8887123234ea0c6e7143c0add73ff431ed
Passwords do not match!
>>>
========== RESTART: E:/Computing/TESTINGFOLDER/HashTesting.py
==========
Enter password:One
Re-Enter password:One
8b12507783d5becacbf2ebe5b01a60024d8728a8f86dcc818bce699e8b3320bc
8b12507783d5becacbf2ebe5b01a60024d8728a8f86dcc818bce699e8b3320bc
Passwords match!
>>>
```

OBJECTIVE 15 SUCCESSFUL – PASSWORDS ARE HASHED FOR SECURITY

LOADING THE INTERFACE FOR LOGGING IN

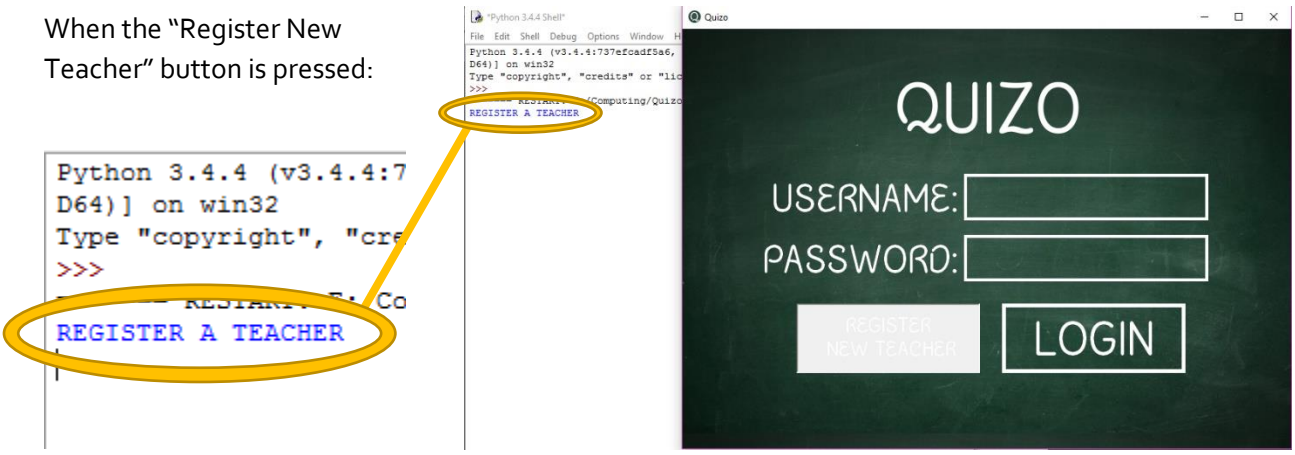
I have created a new file to start programming the first window of my program, and just to make sure everything works I have written it to just load up the UI and print outputs to the IDLE when either of the buttons are pressed. The code I have written can be seen below, as well as some screenshots of the code running as intended.

```
## Login Screen

## IMPORTED ALL MODULES NEEDED FOR THIS WINDOW
import sys,os
import sqlite3 as lite
import hashlib
from PyQt4 import QtCore,QtGui,uic


Login_class = uic.loadUiType("LoginWindow.ui")[0] ## LOADS UI FILE

class LoginWindow(QtGui.QMainWindow,Login_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.setupUi(self)
        self.loginbutton.clicked.connect(self.Login) ## WHEN LOGIN BUTTON PRESSED
        self.registerbutton.clicked.connect(self.Register) ## WHEN REGISTER BUTTON PRESED

    def Register(self):
        print("REGISTER A TEACHER")

    def Login(self):
        print("LOGIN")
        username = self.useredit.text() ## GETS TEXT FROM USERNAME LINE EDIT
        password = self.passedit.text() ## GETS TEXT FROM PASSWORD LINE EDIT
        print(username,password)

app = QtGui.QApplication(sys.argv)
Login_Window = LoginWindow(None)
Login_Window.show() ## SHOWS THE LOGIN WINDOW
app.exec_()
```

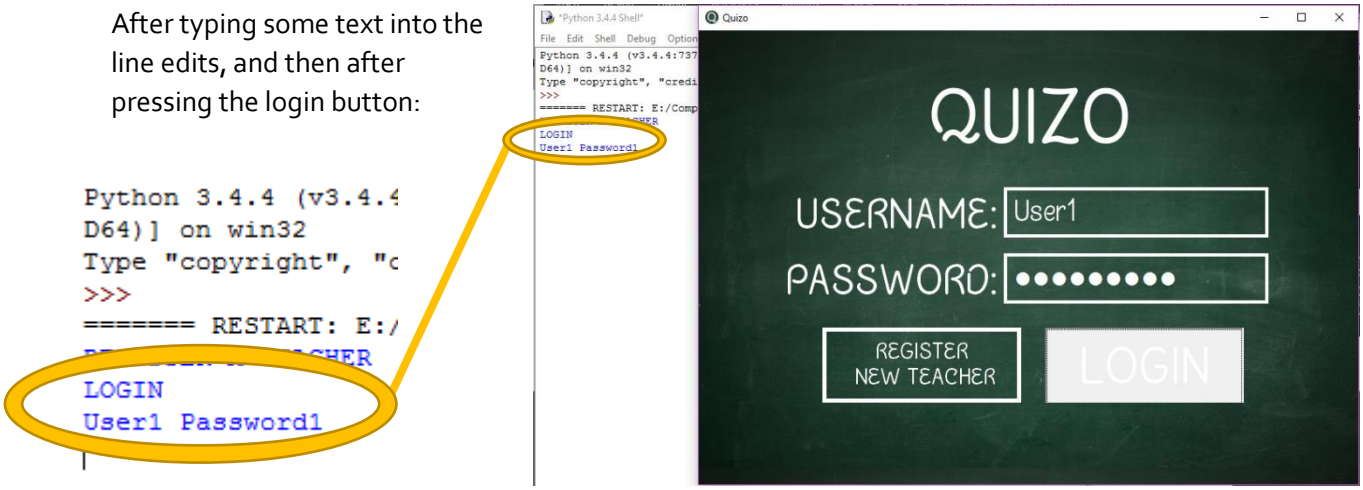


When the script is run initially, the screen appears as intended. In the QtDesigner I also edited the title of the window to match the program's name, and I also created a small icon for the program in photoshop which can be seen below.

When the "Register New Teacher" button is pressed:



After typing some text into the line edits, and then after pressing the login button:



As the buttons and line edits are functioning as expected, the program now has the ability to read user inputs from text boxes, completing another objective.

OBJECTIVE 9 SUCCESSFUL – USERS CAN INPUT DATA INTO TEXT BOXES

FETCHING USER INFORMATION FROM THE DATABASE

I will now add functionality for fetching user accounts from the database and then checking to see whether the account is a student or teacher. When designing my database, in order to fully normalise it in third normal form, I had to create two separate tables for accounts: students and teachers. In order to find which access a user has, I will need to search through one table first, check for the user, then if the user is not found there, check the other table, and if it is still not there then the user account doesn't exist in the database. Before adding functionality for checking which access an account has, I will make sure that the program can fetch data from the database correctly.

```
con = lite.connect('QuizoBase') ## CONNECTING TO THE DATABASE
cur = con.cursor()

def Login(self):
    print("LOGIN")
    username = self.useredit.text() ## GETS TEXT FROM USERNAME LINE EDIT
    password = self.passedit.text() ## GETS TEXT FROM PASSWORD LINE EDIT
    print(username,password)

    query = ["SELECT Username,Password FROM Students;","SELECT Username,Password FROM Teachers;"] ## SELECTS USERNAME AND PASSWORD FOR ALL ACCOUNTS
    cur.execute(query[0]) ## EXECUTES STUDENT SEARCH
    allstudents = cur.fetchall() ## FETCHES ALL DATA FROM STUDENT QUERY
    cur.execute(query[1]) ## EXECUTES TEACHER SEARCH
    allteachers = cur.fetchall() ## FETCHES ALL DATA FROM TEACHER QUERY
    print("Students:",allstudents)
    print("Teachers:",allteachers)
```

I have added the two queries for fetching the usernames and passwords of the accounts, and then I have added print statements to make sure that the correct data has been fetched. The correct data to be fetched from the table is as follows:

Table: Students

	Username	Password	FirstName	Surnam
	Filter	Filter	Filter	Filter
1	woodingmp	af808b049bef56...	Matthew	Wooding
2	WillTaylor	5e884898da280...	Will	Taylor

Table: Teachers

	Username	Password	Title	Surname
	Filter	Filter	Filter	Filter
1	theteacher	5e884898da280...	Mrs	Teacher
2	teacher2	5e884898da280...	Mr	Teacher

Executing the code and then attempting to login results in this output in the IDLE:

```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 20:20:57) [MSC v.1600 64 bit (AMD64)] on win3
2
Type "copyright", "credits" or "license()" for more information.
>>>
======= RESTART: E:\Computing\Quizo - Development File\LoginScreen.py =======
LOGIN
user1 password1
Students: [('woodingmp', 'af808b049bef56c641c0b75f1636c1f3106f6efc177a8d9130b4867b97f00680'),
('WillTaylor', '5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8')]
Teachers: [('theteacher', '5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8')
, ('teacher2', '5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8')]
```

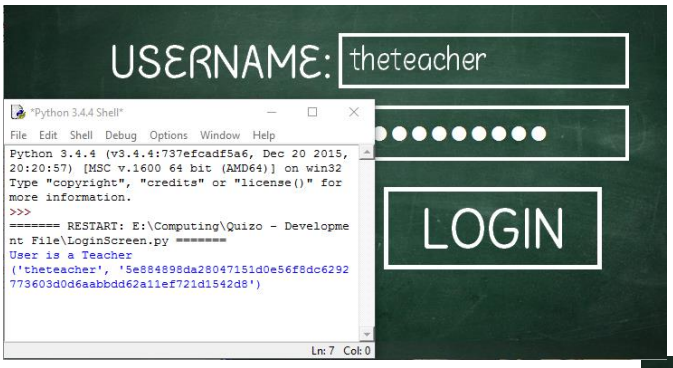
The output in the IDLE is identical to the contents of the database, therefore the SQL statements are working as expected, and the test was successful.

OBJECTIVE 14 SUCCESSFUL – USERNAMES AND PASSWORDS CAN BE FETCHED FROM DATABASE

IDENTIFYING ACCESS LEVELS OF ACCOUNTS

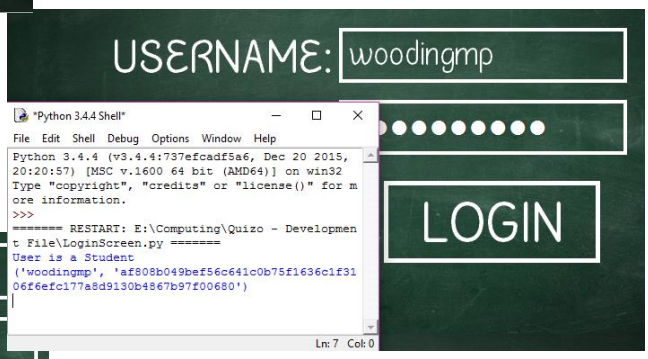
Now that the program is able to fetch all the accounts from the database, the program needs to be able to tell whether the user trying to log on is a teacher, student, or doesn't have an account. The code below should be able to identify whether a user exists in the database, and if so, what level of access they have. I have also done some testing which can be seen below.

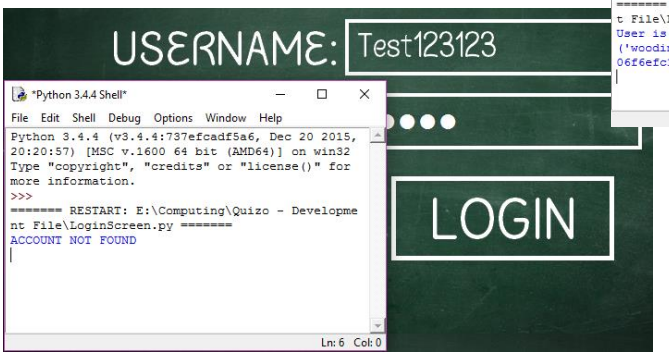
```
## CHECKING USER ACCESS

tempuser = [] ## RESETS TEMPORARY USER
access = "" ## RESETS CURRENT USERS ACCESS
for account in allstudents: ## SEARCHES THROUGH ALL STUDENTS
    if username == account[0]: ## CHECKS TO SEE IF USERNAMES ARE THE SAME
        tempuser = account ## SETS THIS ACCOUNT TO THE NEW TEMPORARY USER
        access = "Student" ## SETS USER'S ACCESS TO STUDENT
if access == "": ## CHECKS TO SEE IF THE ACCOUNT HAS NOT BEEN FOUND YET
    for account in allteachers: ## SEARCHES THROUGH ALL TEACHERS
        if username == account[0]: ## CHECKS TO SEE IF THE USERNAMES ARE THE SAME
            tempuser = account ## SETS THIS ACCOUNT TO THE NEW TEMPORARY USER
            access = "Teacher" ## SETS USER'S ACCESS TO TEACHER
if access == "": ## CHECKS TO SEE IF THE ACCOUNT WAS NOT FOUND
    print("ACCOUNT NOT FOUND")
else:
    print("User is a",access)
    print(tempuser)
```



In each of the tests I entered a username, and then just a random string of letters for the password, because for this test, the password doesn't yet matter. First I tested to see if a teaching account could be recognised, "theteacher", and the program successfully printed the username and password of the account, and recognised it was a teaching account.

In my second test, I tried to find a student account, using "woodingmp" and the program also successfully returned the username and password, and identified it as a student account.





For the last test I entered an unknown username to see if the program could recognize when an account didn't exist, and it was successful in this, printing that the account couldn't be found.

OBJECTIVE 16 SUCCESSFUL – ACCOUNT ACCESS LEVELS CAN BE IDENTIFIED

CHECKING FOR CORRECT LOGIN DETAILS

Now that my program can identify whether or not an account exists in the database, I now need to add the functionality for it to be able to check if a user has entered their password correctly. As seen when I fetched data from the database, I have already hashed some passwords for the initial users in the database, and eventually when new users are added to the program, it will be the hashed form of their passwords that are stored, ensuring better security. For this reason, when checking if a user has entered their password correctly, I will need to hash their attempt, and then check to see if the attempt is the same as the password; if so then the user will have logged in successfully. In order to test this, I have added the hashing of the user's attempt when they try to login, and I have added a simple print command that will print if the user is successful in entering their password or not. These additions can be seen in the code below, followed by my testing for this feature.

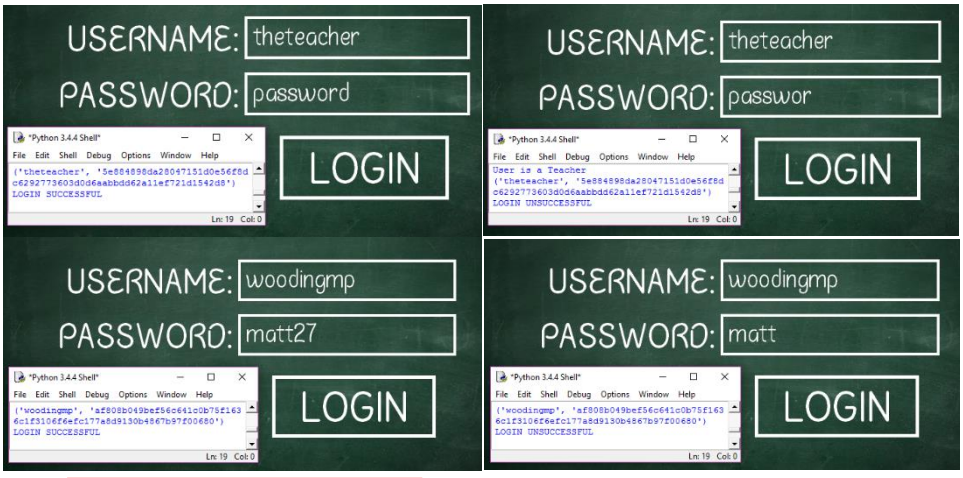
```
if access == "": ## CHECKS TO SEE IF THE ACCOUNT WAS NOT FOUND
    print("ACCOUNT NOT FOUND")
else:
    print("User is a",access)
    print(tempuser)
    hashedattempt = hashlib.sha256(password.encode()).hexdigest() ## HASHES USER'S ATTEMPT
    if hashedattempt == tempuser[1]: ## CHECKS TO SEE IF THE HASHED ATTEMPT IS THE SAME AS THE HASHED PASSWORD
        print("LOGIN SUCCESSFUL")
    else:
        print("LOGIN UNSUCCESSFUL")
```

When I created the database, I used my hash testing program to hash the passwords for the initial users.

```
============= RESTART: E:\Computing\TESTINGFOLDER\HashTesting.py =============
Enter password:password
5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8
>>>
============= RESTART: E:\Computing\TESTINGFOLDER\HashTesting.py =============
Enter password:matt27
af808b049bef56c641c0b75f1636c1f3106f6efc177a8d9130b4867b97f00680
>>>
```

I will test both a teaching and student account with this new feature, testing both correct and incorrect password attempts. The plain text password for "theteacher" was "password" and for "woodingmp" was "matt27". So that it is easier to see what is happening I will be taking the echo mode of the password entry box in the window off so that the password inputs can be seen without being blocked with bullet points.

As can be seen in the tests to the right, the passwords were successfully identified as correct and then incorrect for both teachers and students, meaning that this function is working as intended.

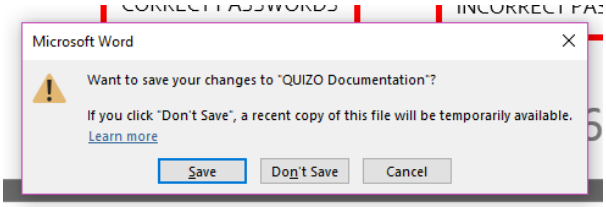


CORRECT PASSWORDS

INCORRECT PASSWORDS

GENERATING OUTPUT WITH PYQT

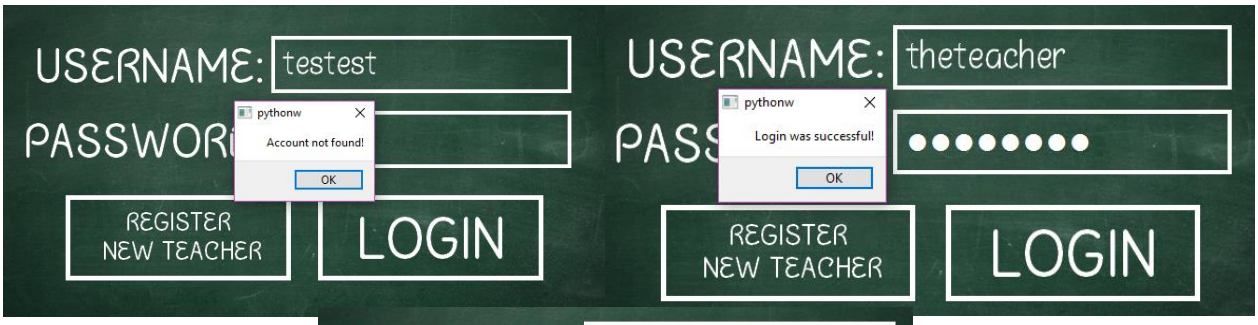
In its current state, my program is using the print command to provide me with information on outputs, which has been useful for checking to see if my functions are working, but is quite unattractive and impractical for my program when it is released for the users. The goal with the program is for all input and output to be achieved without the IDLE to make it a more user friendly and accessible program. Having done a little bit of research, I have decided that pop-up message boxes would be a good way to generate output. An example of what I'm trying to achieve is when you close a program and it asks if you would like to save as can be seen in Microsoft word.



Having looked at how this can be possible with PyQt I have replaced the print statements in my program with code to create a basic message box that will generate the same output, and I have pointed out which parts of the code I have added to do this.

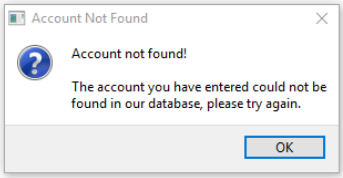
```
outbox = QtGui.QMessageBox() ## CREATES A BLANK MESSAGE BOX

if access == "": ## CHECKS TO SEE IF THE ACCOUNT WAS NOT FOUND
    #print("ACCOUNT NOT FOUND")
    outbox.setText("Account not found!")  ## SETS THE TEXT FOR THE MESSAGE BOX
else:
    #print("User is a",access)
    #print(tempuser)
    hashedattempt = hashlib.sha256(password.encode()).hexdigest() ## HASHES USER'S ATTEMPT
    if hashedattempt == tempuser[1]: ## CHECKS TO SEE IF THE HASHED ATTEMPT IS THE SAME AS THE HASHED PASSWORD
        #print("LOGIN SUCCESSFUL")
        outbox.setText("Login was successful!")
    else:
        #print("LOGIN UNSUCCESSFUL")
        outbox.setText("Login was unsuccessful!")
outbox.exec() ## EXECUTES THE MESSAGE BOX
```

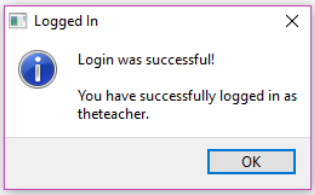
I have tested to make sure all the outputs work by entering a false account "testest", entering a real account, "theteacher", with a false password, and the same real account with the correct password. As can be seen below, the tests worked as expected, however the message boxes look quite bland and uninteresting so I will look into how to spruce them up a bit.



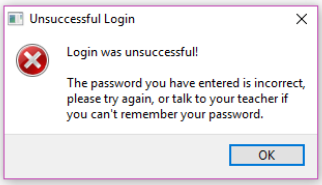
I have added some extra lines of code to make the message boxes look a bit more professional, and they also give a bit more information to the user. I have shown the code for these changes below along with how the new message boxes look.

```
outbox.setText("Account not found!")  ## SETS THE TEXT FOR THE MESSAGE BOX
outbox.setWindowTitle("Account Not Found") ## SETS THE TITLE FOR THE MESSAGE BOX
outbox.setInformativeText("The account you have entered could not be found in our database, please try again.")
outbox.setIcon(QtGui.QMessageBox.Question) ## SETS THE ICON FOR THE MESSAGE BOX
```






```
outbox.setText("Login was successful!")
outbox.setWindowTitle("Logged In")
outbox.setInformativeText("You have successfully logged in as "+username+".")
outbox.setIcon(QtGui.QMessageBox.Information)
```



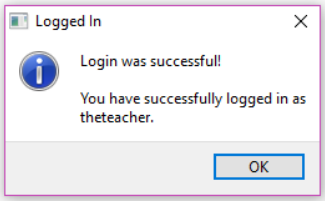

```
outbox.setText("Login was unsuccessful!")
outbox.setWindowTitle("Unsuccessful Login")
outbox.setInformativeText("The password you have entered is incorrect, please try again, or talk to your teacher if you can't remember your password.")
outbox.setIcon(QtGui.QMessageBox.Critical)
```

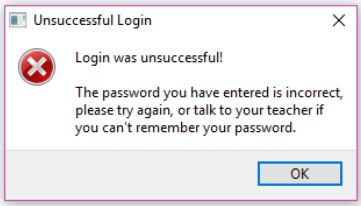
Having written the code for these boxes, and thinking ahead to how many other times outputs like these will be helpful and necessary in my program, I think it would be a good idea to create a function specific for creating message boxes to reduce the amount of code I'll need to write to make them, and also to try and reduce any errors I could make over and over.

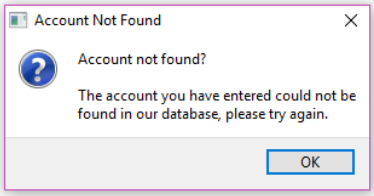
```
def CreateOutbox(Title,Text,InfoText,Icon):
    outbox = QtGui.QMessageBox() ## CREATES THE MESSAGE BOX
    outbox.setWindowTitle(Title) ## SETS THE TITLE OF THE MESSAGE BOX
    outbox.setText(Text) ## SETS THE MAIN TEXT OF THE MESSAGE BOX
    outbox.setInformativeText(InfoText) ## SETS ADDITIONAL TEXT OF THE MESSAGE BOX
    outbox.setIcon(Icon) ## SETS THE ICON OF THE MESSAGE BOX
    outbox.exec() ## EXECUTES THE MESSAGE BOX
```

```
if access == "": ## CHECKS TO SEE IF THE ACCOUNT WAS NOT FOUND
    CreateOutbox("Account Not Found","Account not found?","The account you have entered could not be found in our database, please try again.",QtGui.QMessageBox.Question)
else:
    hashedattempt = hashlib.sha256(password.encode()).hexdigest() ## HASHES USER'S ATTEMPT
    if hashedattempt == tempuser[1]: ## CHECKS TO SEE IF THE HASHED ATTEMPT IS THE SAME AS THE HASHED PASSWORD
        CreateOutbox("Logged In","Login was successful!","You have successfully logged in as "+username+".",QtGui.QMessageBox.Information)
    else:
        CreateOutbox("Unsuccessful Login","Login was unsuccessful!","The password you have entered is incorrect, please try again, or talk to your teacher if you can't remember your password.",QtGui.QMessageBox.Critical)
```

The new function "CreateOutbox" I have created is outside the window's class as I anticipate using it in other classes too, and I have also shown how I am using this function to replace all the code I have before. Having now tested the function, I can say that the outputs are identical to as they were before creating the new function, therefore I hope this will make creating more output boxes as I continue easier. The results of doing the same three tests as above can be seen below.





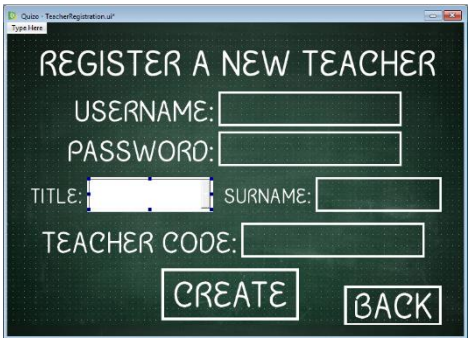


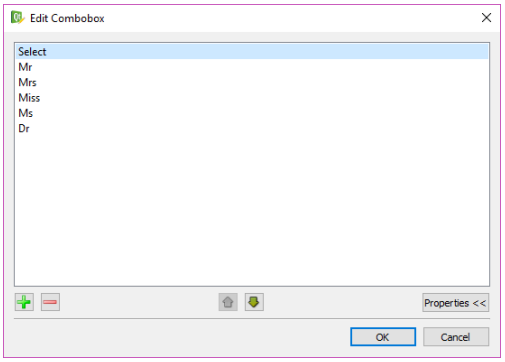
OBJECTIVES 19 AND 20 SUCCESSFUL – PROGRAM PROVIDES OUTPUT WHEN ACCOUNT CAN'T BE FOUND OR WHEN USER ATTEMPTS TO LOG IN UNSUCCESSFULLY

BONUS OBJECTIVE SUCCESSFUL – PROGRAM PROVIDES OUTPUT WHEN USER LOGS IN SUCCESSFULLY – Hadn't thought of this objective when designing my program, however now it seems like a good idea and I will keep it as a part of the program.

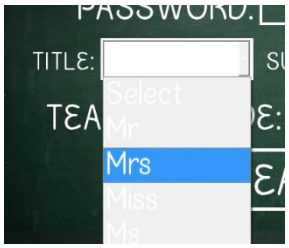
CREATING THE TEACHER REGISTRATION SCREEN

The next window in my program that the user can access is the teacher registration screen, where the user will be able to create new teaching accounts, given that they register with a code only available to teachers, in order to prevent students making their own teacher accounts. Before I start adding functionality for this, I need to create the design for the window in QtDesigner using my original idea from my design section as a starting point.



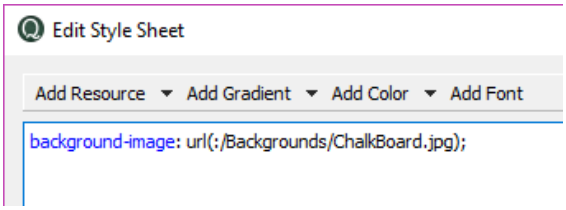


In order to create the drop down menu I had to add some items to the menu in QtDesigner, so I added a few titles. If necessary I could add more titles to the program in future when I go back with my first program to the school, however for now I think the ones I have added will be sufficient.

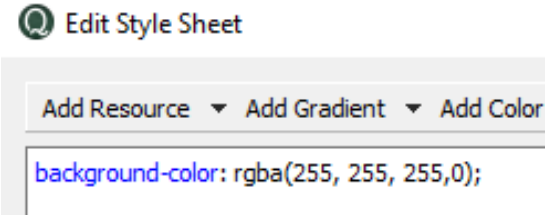


The line edits, buttons, and labels all look in line with the aesthetic with the program, however having put a drop down menu into a frame has resulted in a completely white background, along with white text making it impractical to use, so I will need to find a way to make this also fit in.

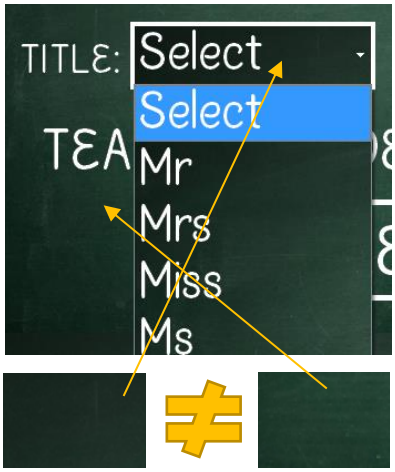
My first attempt at trying to fix this was to edit the stylesheet of the drop box, adding a background image, however when I tried this it only changed the background for the boxes when the user clicks on the box as can be seen to the right.

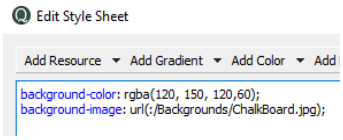


My next idea for fixing this issue was to try and make a transparent background for the box, therefore making the box look in line with the rest of the screen. I achieved this by setting the background colour to white, but then setting alpha to zero, making the colour "transparent". This successfully made the first box transparent, however caused the rest of the drop down menu to have a black background and to cause a strange graphical glitch where all the items could be selected at once.



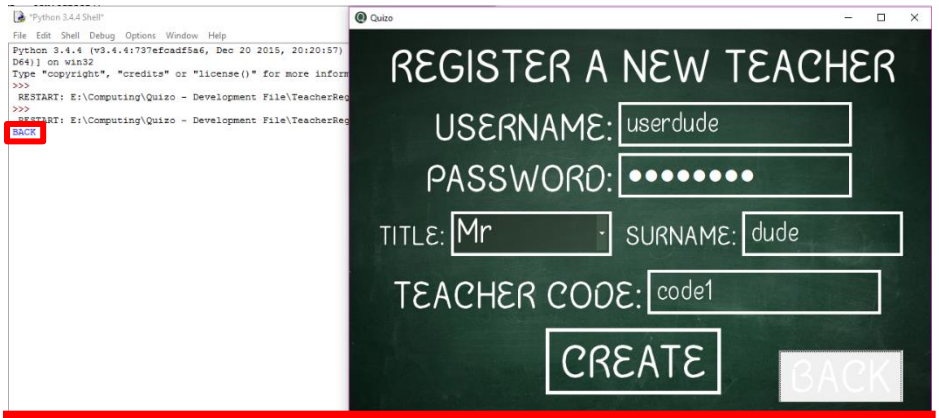
Seeing as how each of these ideas succeeded in one aspect, I decided to merge the two into one style sheet to see if that could fix the issue. Luckily this did fix the issue, both fitting in with the aesthetic and only being able to select one item at a time. However, on closer inspection, the background of the drop down menu was darker than the background, and while I could overlook this, I decided that I should try get it as close as possible. I achieved this by increasing the alpha level for the background colour, and increasing the proportion of green until I reached a point where the two looked close enough. Although this method may not be the best way to match the backgrounds, it worked well enough to the point where I don't think it is necessary to spend even more time looking for a better solution, considering it is such a small portion of the overall program. The results of my final design can be seen below.



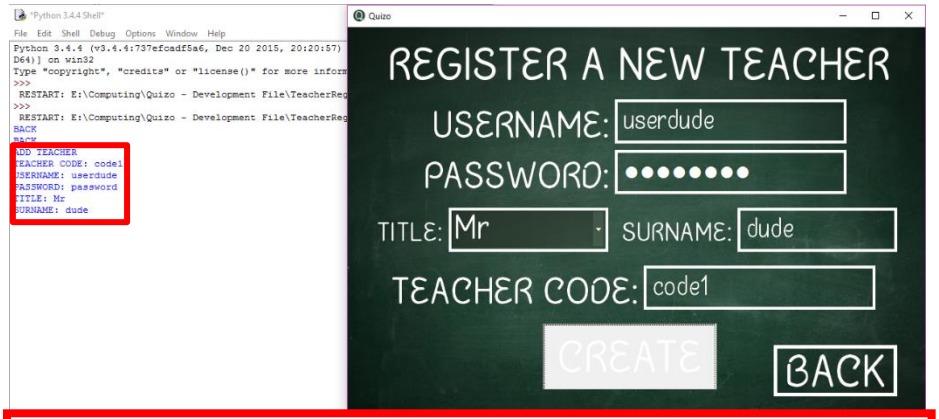


To check that all the widgets in the window are working as intended I have created a function when the "create" button is clicked, which will try and fetch all the data from the widgets and then print it to the IDLE. The code for this can be seen below, along with some testing to check that the widgets are working.

```
## Teacher Registration

## IMPORTED ALL MODULES NEEDED FOR THIS WINDOW
import sys,os
import sqlite3 as lite
import hashlib
from PyQt4 import QtCore,QtGui,uic

con = lite.connect('QuizoBase') ## CONNECTING TO THE DATABASE
cur = con.cursor()

TeacherRegistration_class = uic.loadUiType("TeacherRegistration.ui")[0] ## LOADS UI FILE

def CreateOutbox(Title,Text,InfoText,Icon):
    outbox = QtGui.QMessageBox() ## CREATES THE MESSAGE BOX
    outbox.setWindowTitle(Title) ## SETS THE TITLE OF THE MESSAGE BOX
    outbox.setText(Text) ## SETS THE MAIN TEXT OF THE MESSAGE BOX
    outbox.setInformativeText(InfoText) ## SETS ADDITIONAL TEXT OF THE MESSAGE BOX
    outbox.setIcon(Icon) ## SETS THE ICON OF THE MESSAGE BOX
    outbox.exec() ## EXECUTES THE MESSAGE BOX

class TeacherRegistration(QtGui.QMainWindow,TeacherRegistration_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.setupUi(self)
        self.backbutton.clicked.connect(self.Back) ## STARTS BACK FUNCTION WHEN BACK BUTTON CLICKED
        self.createbutton.clicked.connect(self.addteacher) ## STARTS ADDTEACHER FUNCTION WHEN CREATE BUTTON CLICKED

    def addteacher(self):
        print("ADD TEACHER")
        print("TEACHER CODE:",self.codeedit.text()) ## PRINTS CONTENTS OF TEACHER CODE BOX
        print("USERNAME:",self.useredit.text()) ## PRINTS CONTENTS OF USERNAME BOX
        print("PASSWORD:",self.passedit.text()) ## PRINTS CONTENTS OF PASSWORD BOX
        print("TITLE:",self.comboBox.currentText()) ## PRINTS SELECTED ITEM IN THE TITLE DROP DOWN
        print("SURNAME:",self.suredit.text()) ## PRINTS CONTENTS OF SURNAME BOX

    def Back(self):
        print("BACK")

app = QtGui.QApplication(sys.argv)
TeacherRegistration_Window = TeacherRegistration(None)
TeacherRegistration_Window.show() ## SHOWS THE TEACHER REGISTRATION WINDOW
app.exec_()
```



When the back button is pressed, "BACK" is printed to the IDLE as intended



When the create button is pressed, "ADD TEACHER" is printed to the IDLE, along with the correct data fetched from the entry boxes and drop down menu. Therefore, this test has been successful.

Now that I have seen that all the widgets are working as intended and data can be fetched from them, I will write the function for collecting all the data and validating that the entered data is in the correct format for the database. In order for the data to be in the correct format:

- The username must be a non-null string, with only letters and numbers, and at least three characters
- The password must be at least 6 characters, containing at least 1 capital, and 1 lowercase letter, and only numbers and letters
- Surnames must begin with a capital letter and not contain any numbers or punctuation except "-"
- The selected title must not be "Select"

I have written code to check whether each of these conditions are met, and if so, a message will be printed saying that the account has been created. However, if any of these conditions are not met, they will be printed to the user.

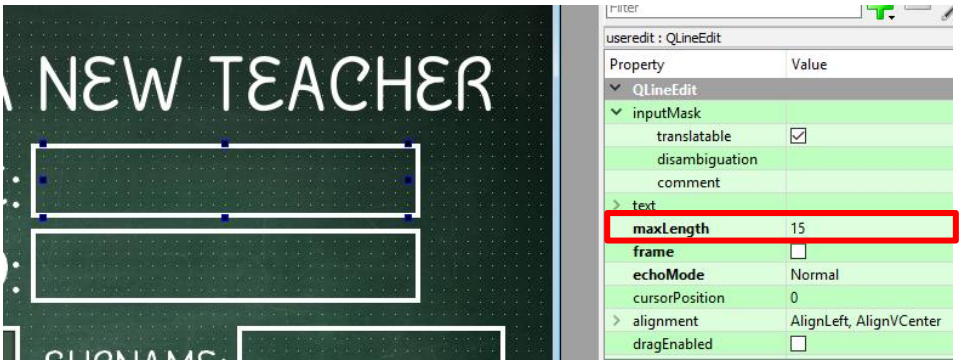
USERNAME VALIDATION

I have edited the "addteacher" function so that it begins by setting variables for each of the inputs in the window, and I did this simply to make it easier to reference inputs. I have also created a new variable called "allowedcharacters", which is a string of all the characters that are allowed in the username, and password for the user. The same characters are for the surname aswell, however surnames can also contain dashes due to double barrelled surnames, and I will address that when I come to surname validation.

For validating usernames, first the program checks that the username is at least three characters, and if not an error is produced. If the username is longer than three characters, the program will then start a loop to go through each character in the username and check that each character is allowed. If a character is registered as disallowed, then the program will print the error and break out of the loop so that it can only be printed once.

```
def addteacher(self):
    print("ADD TEACHER")
    teachercode = self.codeedit.text() ## FETCHES CONTENTS OF TEACHER CODE BOX
    username = self.useredit.text() ## FETCHES CONTENTS OF USERNAME BOX
    password = self.passedit.text() ## FETCHES CONTENTS OF PASSWORD BOX
    title = self.comboBox.currentText() ## FETCHES SELECTED ITEM IN THE TITLE DROP DOWN
    surname = self.suredit.text() ## FETCHES CONTENTS OF SURNAME BOX

    allowedcharacters = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789" ## STRING CONTAINING ALL ALLOWED CHARACTERS FOR USERNAMES

    if len(username) > 2: ## CHECKS THAT THE USERNAME IS LONGER THAN 2 CHARACTERS, AS USERNAME MUST BE AT LEAST 3 CHARACTERS LONG
        for char in username: ## STARTS FOR LOOP WHICH GOES THROUGH EACH CHARACTER IN USERNAME
            if char not in allowedcharacters: ## CHECKS IF CHARACTER IS DISALLOWED
                print("USERNAME HAS INVALID CHARACTER(S)")
                break ## PREVENTS THE ERROR FROM BEING PRINTED MORE THAN ONCE
    else:
        print("USERNAME MUST BE AT LEAST 3 CHARACTERS LONG")
```

To prevent usernames being exceedingly long, I have also set the max length of the line edit for the username as 15 which can be seen to the right.



In order to test that the username validation is working as expected I will do a number of different tests as follows:

Test Number	Username Entered	Test Purpose
1	JS	Tests that minimum length checking is working
2	12345678910111213	Tests that the text box cannot hold more than 15 characters
3	John!& "£Smith	Tests that disallowed characters will print an error, and only one error is printed even if there are multiple disallowed characters
4	JohnSmith	Tests that the program prints no errors when a valid username is entered

Test 1 – Successful – Must be at least 3 characters long

Test 2 – Successful – Username could only be entered up to 123456789101112, which is 15 characters long, any extra characters I tried to enter wouldn't be entered into the text box.

Test 3 – Successful – Error message was printed for the invalid character "!", and then the program broke out of the loop, preventing any more errors being printed for the remaining "&", " ", "'", and "£".

Test 4 – Successful – No errors were printed and the username was validated.

All tests were successful and therefore username validation is working as intended and needs no further amendments.

PASSWORD VALIDATION

For validating passwords in my program I have settled on a few standards including that the password must be at least six characters long, consist of one uppercase and one lowercase letter, and only use alphanumerical characters. First the program will check for the length of the password: unlike the usernames, I have not set a max number of characters for passwords as the longer they are the more secure they are in theory. The program will then check for disallowed characters and give an error if one is found. Finally, it will check that at least one uppercase and one lowercase letter are present, giving an error if there are none of one or the other. In order to test this section more easily, I have commented out the print functions from the username validation so that I will only see details for password validation.

```
if len(password) > 5: ## CHECKS THAT PASSWORD IS LONGER THAN FIVE CHARACTERS
    uppercase = 0 ## RESETS NUMBER OF UPPERCASE CHARACTERS FOUND IN PASSWORD
    lowercase = 0 ## RESETS NUMBER OF LOWERCASE CHARACTERS FOUND IN PASSWORD
    for char in password: ## STARTS FOR LOOP WHICH GOES THROUGH EACH CHARACTER IN PASSWORD
        if char not in allowedcharacters: ## CHECKS IF CHARACTER IS DISALLOWED
            print("PASSWORD HAS INVALID CHARACTER(S)")
            break ## PREVENTS THE ERROR FROM BEING PRINTED MORE THAN ONCE
        elif char.isupper() == True: ## CHECKS IF CHARACTER IS UPPERCASE
            uppercase += 1 ## INCREMENTS NUMBER OF UPPERCASE LETTERS
        elif char.islower() == True: ## CHECKS IF CHARACTER IS LOWERCASE
            lowercase += 1 ## INCREMENTS OF LOWERCASE LETTERS
    if (uppercase == 0) or (lowercase == 0): ## CHECKS TO SEE IF PASSWORD DOESN'T REACH REQUIREMENTS
        print("PASSWORD HAS INVALID NUMBER OF UPPERCASE AND LOWERCASE LETTERS")
else:
    print("PASSWORD MUST BE LONGER THAN 6 CHARACTERS")
```

Similarly, to username validation I have come up with a few tests to make sure that it is working correctly:

Test Number	Password Entered	Test Purpose
1	Pass	Tests that minimum length checking is working
2	P123456	Tests that lowercase counting is working
3	p123456	Tests that uppercase counting is working
4	Pass£& word	Tests that disallowed characters are checked and only one error printed
5	Password	Tests that the program prints no errors when a valid username is entered

In order to show the inputs for the test, I have turned off the echo mode in QtDesigner so that the entries can be seen alongside the outputs.

Test 1 – Successful – Error printed as password is only 4 characters long, minimum is 6

Test 2 – Successful – Error printed as password contains no lowercase characters, minimum is 1

Test 3 – Successful – Error printed as password contains no uppercase characters, minimum is 1

Test 4 – Successful – Error printed as password contains "£" which is an invalid character, and only one error is printed even though there are also "&" and " " characters

Test 5 – Successful – No errors are produced as password meets all the requirements

As all tests were successful, this validation is complete and requires no further amendments.

SURNAME VALIDATION

Validating surnames is similar to validating usernames, however they can only contain letters, and must start with a capital letter. Instead of having the user be forced to input their surname in the correct format, when a surname is entered, the program will set it to all lowercase except for an uppercase initial. Additionally, I am aware of users with double barrelled surnames, and so the dash character will be allowed once in a surname. The code for this validation that I have written can be seen below.

```
if surname != "":
    dashes = 0 ## RESETS NUMBER OF DASHES IN SURNAME
    for char in surname: ## STARTS FOR LOOP WHICH GOES THROUGH EACH CHARACTER IN SURNAME
        if char == "-": ## CHECKS IF CHARACTER IS A DASH
            dashes += 1 ## INCREMENTS THE NUMBER OF DASHES IN THE SURNAME
        elif char.isalpha() == False: ## CHECKS IF CHARACTER IS NOT A LETTER
                    print("SURNAME HAS INVALID CHARACTER(S)")
                    break ## PREVENTS THE ERROR FROM BEING PRINTED MORE THAN ONCE
    if dashes > 1: ## CHECKS IF USER HAS MORE THAN ONE DASH IN THEIR SURNAME
        print("YOU MAY ONLY HAVE ONE DASH IN YOUR SURNAME")
    else:
        surname = surname.lower() ## SETS SURNAME TO LOWERCASE
        if dashes == 1: ## CHECKS IF USER HAS A DOUBLE BARRELLED SURNAME
            surname = surname.split("-") ## SPLITS SURNAME INTO SEPARATE BARRELS
            if surname[0] == "" or surname[1] == "": ## CHECKS THAT NEITHER BARREL IS NULL
                print("AT LEAST ONE BARREL IS EMPTY")
            else:
                barrel1 = surname[0].replace(surname[0][0],surname[0][0].upper(),1) ## SETS FIRST CHARACTER OF FIRST BARREL TO UPPERCASE
                barrel2 = surname[1].replace(surname[1][0],surname[1][0].upper(),1) ## SETS FIRST CHARACTER OF SECOND BARREL TO UPPERCASE
                surname = (barrel1 + "-" + barrel2) ## REJOINS THE TWO BARRELS IN CORRECT FORMAT WITH A DASH
        else:
            surname = surname.replace(surname[0],surname[0].upper(),1) ## SETS FIRST CHARACTER OF SURNAME TO UPPERCASE
        print(surname)
else:
    print("SURNAME IS EMPTY")
```

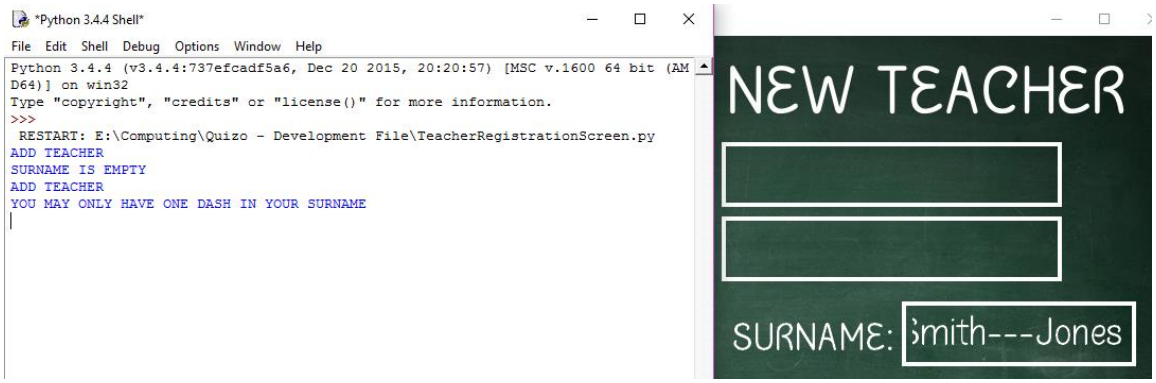
For this section of code, instead of creating a new string with all the allowed characters, I have found that the "isalpha" function is able to return whether a passed character is a letter or not, and I may add this back in the username and password validation to save having to create the "allowedcharacters" string.

To show that the validation is working as it should I have devised some tests to check that it produces errors when necessary, and can output surnames in the correct format:

Test Number	Surname Entered	Test Purpose
1	[NO ENTRY]	Tests that an error is provided when no surname is entered
2	Smith---Jones	Tests that no more than one dash can be used in a surname
3	Smith&123	Tests that only letters can be used in a surname
4	Smith-Jones	Tests that a double barrelled surname can be used without error
5	smitH-joneS	Tests that a double barrelled surname can be entered in an incorrect format, and will be formatted correctly
6	smitH	Tests that single barrelled surnames work without error even in an incorrect format
7	Smith	Tests that single barrelled surnames work without error in a correct format

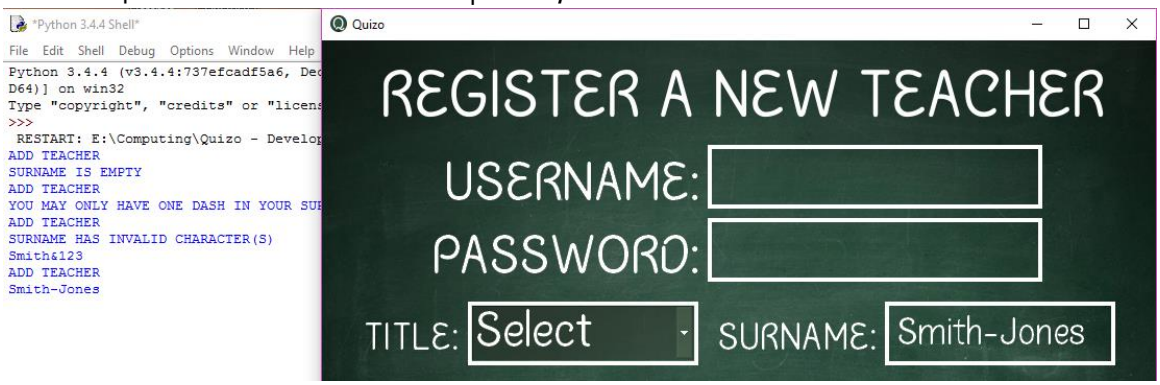
Test 1 – Successful – Error printed as no surname was entered

Test 2 – Successful – Error printed as there were too many dashes



Test 3 – Successful – Error printed as there were invalid characters ("&","1","2", and "3") and only printed the error once as expected

Test 4 – Successful – No error was printed, and the surname was left in a correct format



Test 5 – Successful – No error was printed, and the surname was formatted correctly

Test 6 – Successful – No error was printed, and the surname was formatted correctly

Test 7 – Successful – No error was printed, and the surname was left in a correct format

As all tests were successful, I can safely say that surname validation is complete and needs no further amendments.

TITLE VALIDATION

For the drop down title select box, there isn't too much validation that needs to be done as there are only a few possible inputs that it can take. The only thing that needs to be checked here is that the user doesn't have the "Select" item chosen when they try to create an account. The code for this is below, along with my two tests to check that it works correctly.

```
if title == "Select": ## CHECKS TO SEE IF THE SELECT OPTION HAS BEEN SELECTED
    print("PLEASE CHOOSE A TITLE FROM THE GIVEN OPTIONS")
else:
    print(title,"SELECTED")
```

Test 1 – Select Option Chosen – Successful – Program successfully printed an error

Test 2 – Mr Option Chosen – Successful – Program didn't print an error, and printed the selected option "Mr"

As both tests were successful, this validation testing is complete, and no further amendments will need to be made to the title box.

TEACHER CODE VALIDATION

In this section of the program, the teacher code is used to verify that the user trying to create a teacher account is another teacher. The actual code that is given to teachers will be hard-coded into the program for now as I don't see it being something that will need to be changed in the future. However, I will inform the school of this plan when I next speak with them and hear whether their opinion differs. For now, I have set the code to "YC71N" just as a random five-character string.

As this code can only be either correct or wrong when entered, and as it has no interaction with the database, it is unnecessary for it to have any kind of validation, except for checking whether it is correct or not. The code for this check is below, along with two tests, showing a correct and incorrect entry of the code.

```
if teachercode == "YC71N": ## CHECKS IF THE ENTERED CODE MATCHES
    print("CODE CORRECT")
else:
    print("CODE INCORRECT")
```

Test 1 – Incorrect Code Entered – Successful – Code recognised as incorrect

Test 2 – Correct Code Entered – Successful – Code recognised as correct

As the tests were successful, validation for this window is complete, and next I will need to add functionality for actually adding the accounts to the database.

OBJECTIVE 22 COMPLETED – VALIDATION WORKS WHEN TRYING TO ADD A NEW TEACHER

EFFICIENCY FIX – CHECKING DISALLOWED CHARACTERS

Before I add the database functionality, I went back and adapted the username and password checking for disallowed characters. Instead of having a list of disallowed characters, I've used the "isalnum" function to test that only alphanumerical characters are used. This function does exactly as the program was doing before, but without that extra string taking up unnecessary storage in memory for the program. The code can be seen below.

```
if len(username) > 2: ## CHECKS THAT THE USERNAME IS LONGER THAN 2 CHARACTERS, AS USERNAME MUST BE AT LEAST 3 CHARACTERS LONG
    if username.isalnum() == False: ## CHECKS IF STRING IS MADE UP OF ONLY ALPHANUMERICAL CHARACTERS
        print("USERNAME HAS INVALID CHARACTER(S)")
else:
    print("USERNAME MUST BE AT LEAST 3 CHARACTERS LONG")


if len(password) > 5: ## CHECKS THAT PASSWORD IS LONGER THAN FIVE CHARACTERS
    uppercase = 0 ## RESETS NUMBER OF UPPERCASE CHARACTERS FOUND IN PASSWORD
    lowercase = 0 ## RESETS NUMBER OF LOWERCASE CHARACTERS FOUND IN PASSWORD
    if password.isalnum() == False: ## CHECKS IF STRING IS MADE UP OF ONLY ALPHANUMERICAL CHARACTERS
        print("PASSWORD HAS INVALID CHARACTER(S)")
```

CREATING ERROR MESSAGES

At the moment, this screen prints all errors to the IDLE instead of using the error message boxes I created for the login screen. For this screen, instead of creating an error message for every issue that arises, I will create a list of all the errors the user has, and just display them in a single message box. If the user gets no error messages, then the account will be added to the database. The code for each of the errors can be seen below.

```
def addteacher(self):
    print("ADD TEACHER")
    teachercode = self.codeedit.text() ## FETCHES CONTENTS OF TEACHER CODE BOX
    username = self.useredit.text() ## FETCHES CONTENTS OF USERNAME BOX
    password = self.passedit.text() ## FETCHES CONTENTS OF PASSWORD BOX
    title = self.comboBox.currentText() ## FETCHES SELECTED ITEM IN THE TITLE DROP DOWN
    surname = self.suredit.text() ## FETCHES CONTENTS OF SURNAME BOX

    errors = [] ## RESETS THE ERRORS WHEN ADDING A NEW ACCOUNT
```

When the create button is pressed an error list is reset.

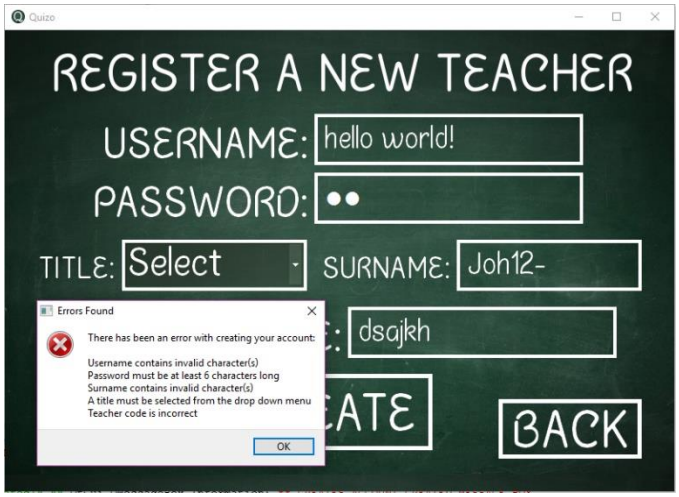
Errors are added as strings to the error list where before there were print statements.

```
if len(username) > 2: ## CHECKS THAT THE USERNAME IS LONGER THAN 2
    if username.isalnum() == False: ## CHECKS IF STRING IS MADE UP
        errors.append("Username contains invalid character(s)")
else:
    errors.append("Username must be at least 3 characters long")
```

```
if errors != []: ## CHECKS IF USER HAS HAD ANY ERRORS
    errormessage = "\n".join(errors) ## CONVERTS LIST OF ERRORS INTO SINGLE STRING
    CreateOutbox("Errors Found","There has been an error with creating your account:",errormessage,QtGui.QMessageBox.Critical) ## CREATES ERROR MESSAGE BOX
else:
    CreateOutbox("Account Created","Your account has been created!","",QtGui.QMessageBox.Information) ## CREATES ACCOUNT CREATED MESSAGE BOX
```

After all validation checks are complete, the program checks to see if any errors occurred, and if so, all the errors from the list are joined into a single string, and then a message box is created to display them. If no errors are found, a message box will be created to show that the account has been created.

Running this code and entering in some data that would produce errors results in message boxes looking like this:



Entering perfect data without any errors results in a message box looking like this:

As can be seen in these screenshots, the error messages are working as intended, and now I am amble to add database functionality to the program.

DATABASE FUNCTIONALITY AND CREATING ACCOUNTS

In order to check that usernames have not previously been taken, and to be able to add new teaching accounts to the database, this window needs to be able to connect to the database and make changes. To start, I will test the two SQLite statements I plan on using in this window, in the SQLite Browser to check that they provide the results needed. If both work successfully, then I will add the functionality to my program.

Fetching all account usernames:

In order to prevent two users having the same account, when a new account is made, all previously made usernames need to be fetched so the program can make sure the user isn't trying to make a new account with the same username as a pre-existing one. To do this I will have to do two SQL statements; one for the student accounts, and one for the teacher accounts. The statements for this I will try are:

"SELECT Username FROM Teachers WHERE Username=(username);" and

"SELECT Username FROM Students WHERE Username=(username);"

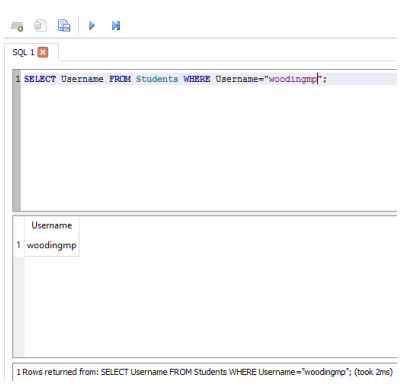
Below I have added screenshots of the current state of both the Students and Teachers tables, along with the results of executing the SQLite statements.

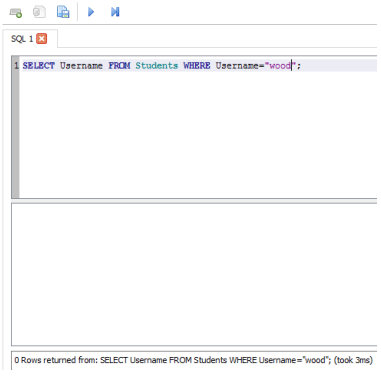
Table: Students

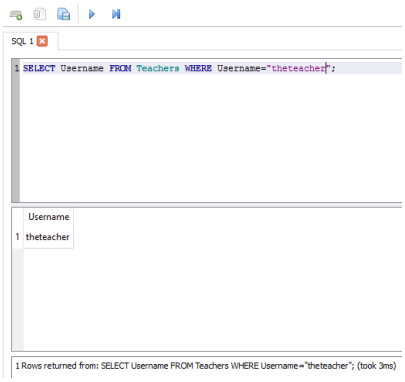
	Username	Password	FirstName	Surname
	Filter	Filter	Filter	Filter
1	woodingmp	af808b049bef56...	Matthew	Wooding
2	WillTaylor	5e884898da280...	Will	Taylor

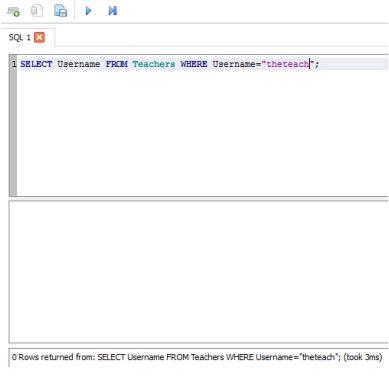
Table: Teachers

	Username	Password	Title	Surname
	Filter	Filter	Filter	Filter
1	theteacher	5e884898da280...	Mrs	Teacher
2	teacher2	5e884898da280...	Mr	Teacher









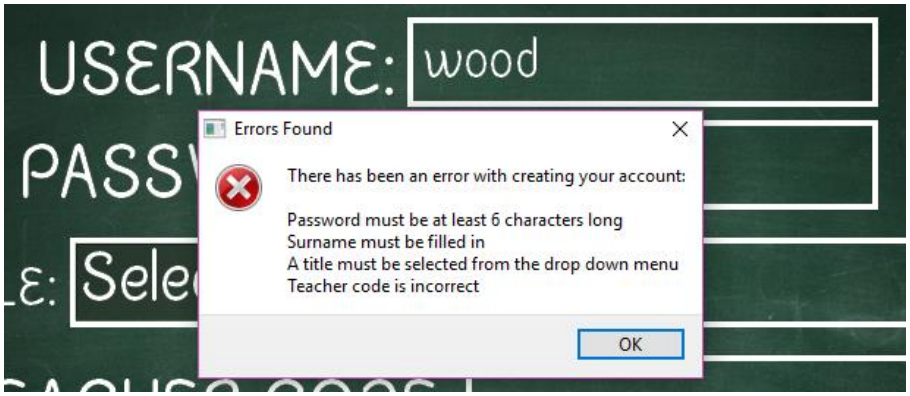
As can be seen above, both statements successfully fetched the username I searched for from their respective tables when they existed, and where they didn't, nothing was fetched. I can use this for checking if usernames have been taken, by seeing whether the fetched data is empty or not. Below I have added the code for this into my program and the tests I will do to make sure that the functionality is working as intended.

```
query = "SELECT Username FROM Teachers WHERE Username='"+username+"';" ## CREATES QUERY TO CHECK TEACHERS TABLE FOR USERNAME
cur.execute(query) ## EXECUTES QUERY
temp1 = cur.fetchall() ## FETCHES DATA FROM QUERY
query = "SELECT Username FROM Students WHERE Username='"+username+"';" ## CREATES QUERY TO CHECK STUDENTS TABLE FOR USERNAME
cur.execute(query) ## EXECUTES QUERY
temp2 = cur.fetchall() ## FETCHES DATA FROM QUERY
if temp1 != [] or temp2 != []: ## CHECKS IF ANY USERNAMES HAVE BEEN FETCHED
    errors.append("Username already taken")
```

To test this functionality in the program, I will do the same four tests as I did in the database.

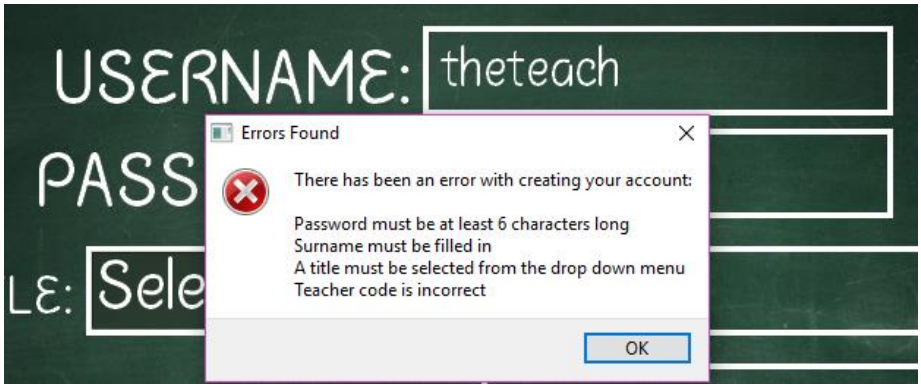
Test 1 – Username "woodingmp" – Successful – Error box shows that username already taken

Test 2 – Username "wood" – Successful – Username taken error not seen in error message



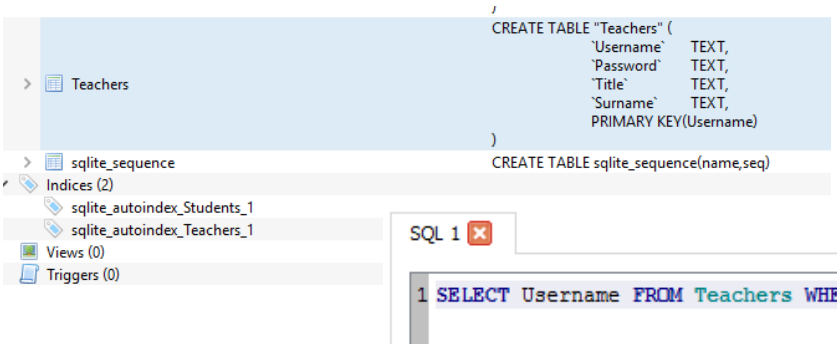
Test 3 – Username "theteacher" – Successful – Error box shows that username already taken

Test 4 – Username "theteach" – Successful – Username taken error not seen in error message

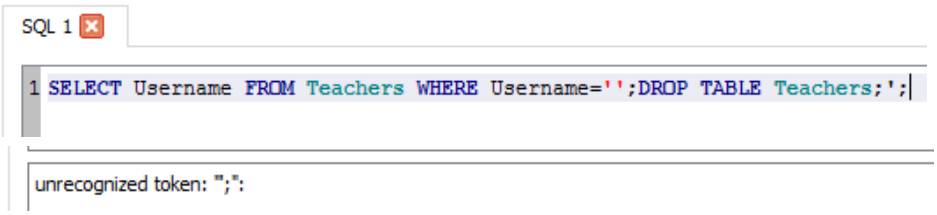


Although the four tests I have done have been successful, there is one error that I could see be exploited, which is that currently, the username is checked in the database before it is checked for disallowed characters. For this reason, it is possible that users could try to access the database and run

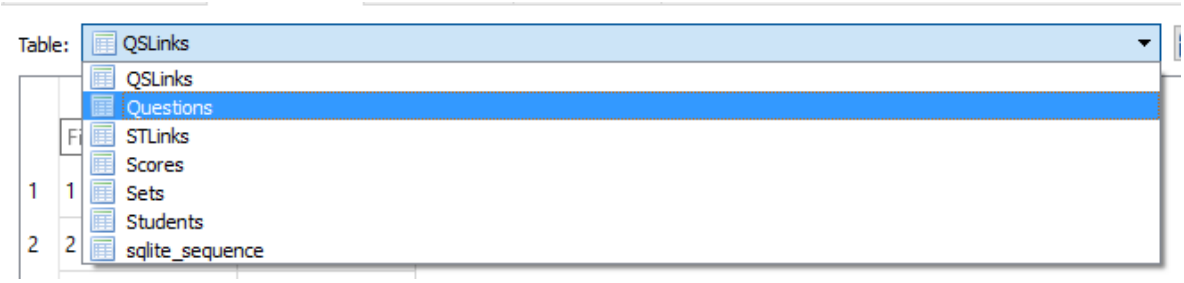
some SQLite queries such as "DROP TABLE" which could be done by entering the username "';DROP TABLE Teachers;'" which can be seen below, working in the SQLite Browser.



Before executing the query, the Teachers table still exists.



If the user entered the username "';DROP TABLE Teachers;'" then this is the statement that is run in SQLite. Even though it says unrecognized token, the DROP TABLE function has already been executed, and the Teachers table no longer exists in the database as seen below.



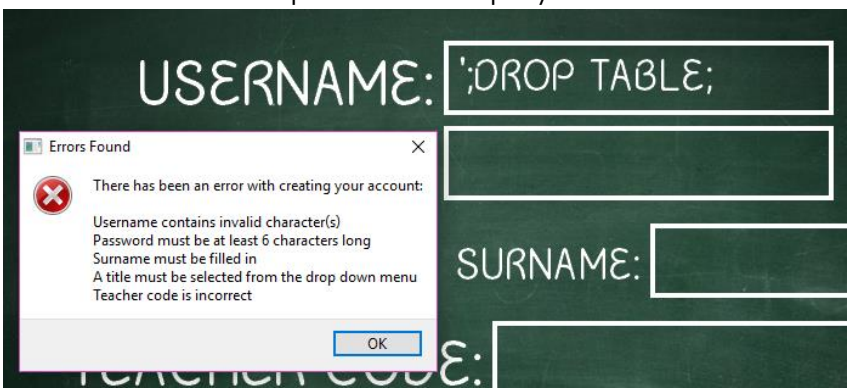
Even though a user trying to enter this username would be a breach of the Computer Misuse Act 1990, it is still a possibility that needs to be taken into account. When attempting to enter this username into my program, I physically couldn't due to the fifteen-character limit of the username field. However, I still think that it would be a good idea to make some prevention against this, in case any other function of SQLite that I haven't thought of is attempted to be entered into the program.

In order to do this, I will just search for pre-existing usernames only if the string entered contains no invalid characters, which should solve the issue.

I have also decided that I can likely find a more efficient way to search the database using a loop, so I have implemented my attempt, along with some tests below to show that my new and more efficient way works.

```
if len(username) > 2: ## CHECKS THAT THE USERNAME IS LONGER THAN 2 CHARACTERS, AS USERNAME MUST BE AT LEAST 3 CHARACTERS LONG
    if username.isalnum() == False: ## CHECKS IF STRING IS MADE UP OF ONLY ALPHANUMERICAL CHARACTERS
        errors.append("Username contains invalid character(s)")
    else:
        tables = ["Teachers","Students"] ## LIST CONTAINING TABLE NAMES TO BE SEARCHED
        for x in tables: ## STARTS A FOR LOOP FOR SEARCHING THE TABLES
            query = "SELECT Username FROM "+x+" WHERE Username='"+username+"';" ## CREATES THE QUERY FOR SEARCHING THE CURRENT TABLE
            cur.execute(query) ## EXECUTES THE QUERY
            temp = cur.fetchall() ## FETCHES ANY DATA FROM THE DATABASE THAT WAS SELECTED
            if temp!= []: ## CHECKS IF ANY DATA WAS FETCHED
                errors.append("Username already taken")
                break ## BREAKS OUT OF THE LOOP IF THE USERNAME HAS BEEN FOUND
else:
    errors.append("Username must be at least 3 characters long")
```

Test 1 – Username ';DROP TABLE; - Successful – The invalid characters were successfully identified, and no attempt was made to query the database.



Now that there is no worry of users being able to make unnecessary changes to the database, I am able to add the code for inserting new teachers into the Teachers table in the database. The program will only attempt to add a teacher when there have been no errors in any of the input fields, so all of the data should be in a format fit for the database, except for the password which needs to be hashed before entering it into the database. The code I have written to add the accounts can be seen below, along with some tests to show that it works as intended.

```
if errors != []: ## CHECKS IF USER HAS HAD ANY ERRORS
    errormessage = "\n".join(errors) ## CONVERTS LIST OF ERRORS INTO SINGLE STRING
    CreateOutbox("Errors Found","There has been an error with creating your account:",errormessage,QtGui.QMessageBox.Critical) ## CREATES ERROR MESSAGE BOX
else:
    password = hashlib.sha256(password.encode()).hexdigest() ## HASHES THE PASSWORD FOR STORING IN THE DATABASE
    query = 'INSERT INTO Teachers (Username,Password,Title,Surname) VALUES ("'+username+'","'+password+'","'+title+'","'+surname+'");' ## INSERTS NEW ACCOUNT INTO DATABASE WITH CORRECT DATA
    cur.execute(query) ## EXECUTES THE QUERY
    con.commit() ## WRITES CHANGES TO THE DATABASE
    CreateOutbox("Account Created","Your account has been created!","",QtGui.QMessageBox.Information) ## CREATES ACCOUNT CREATED MESSAGE BOX
```

Test Data for Adding a Teaching Account

Field	Input	Expected Format in DB	Successful?
Username	MathsTeacher	MathsTeacher	Yes
Password	Ilovemaths	52715652966a88ace2b230973b60f7c7fa319ae060ac6b74a906e0376c0bfeb5	Yes
Title	Mr	Mr	Yes
Surname	jonah-jameson	Jonah-Jameson	Yes

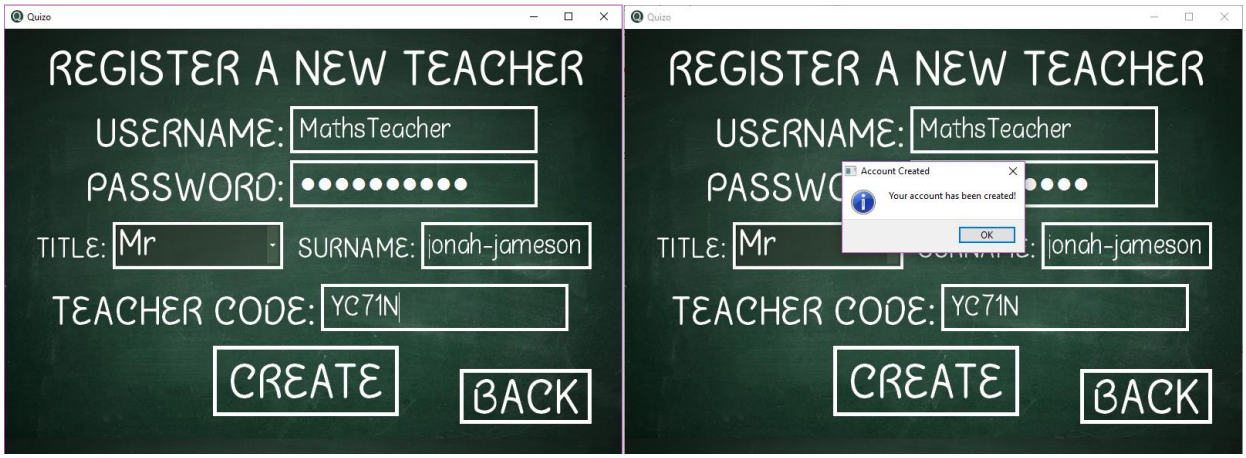


Table: Teachers

	Username	Password	Title	Surname
	Filter	Filter	Filter	Filter
1	theteacher	5e884898da280...	Mrs	Teacher
2	teacher2	5e884898da280...	Mr	Teacher
3	MathsTeacher	52715652966a8...	Mr	Jonah-Jameson

Test was successful as the new teaching account was added to the database with the expected values.

OBJECTIVES 21,23, and 24 COMPLETED – TEACHERS ARE ABLE TO CREATE NEW ACCOUNTS BY PRESSING THE "CREATE" BUTTON, AND UPON SUCCESSFULLY ADDING THE ACCOUNT TO THE DATABASE, A MESSAGE BOX INFORMS THEM OF THIS

Now that all the validation, and being able to add new teaching accounts to the database works, I can safely say that this window is very nearly complete. The only part of this window that has not been complete is pressing the back button which will take the user back to the login screen. I will deal with this functionality next so that users are able to seamlessly switch between windows throughout the program.

SEAMLESS WINDOW SWAPPING

My program requires the ability to swap between windows so that the user can easily manoeuvre the program and all its menus. In order to test this, I have created a new main python file which I will copy the contents of the each of the windows into. This way, all the classes are inside the same file and are able to interact with each other. With all the classes in the same file, I can use the ".show()" and ".hide()" commands to show and hide windows when needed so that when the user swaps between them, the program will just show the window they are going to, and hide the one they are already on. Below is the new file along with the code that allows the user to swap between the two windows I have currently completed.

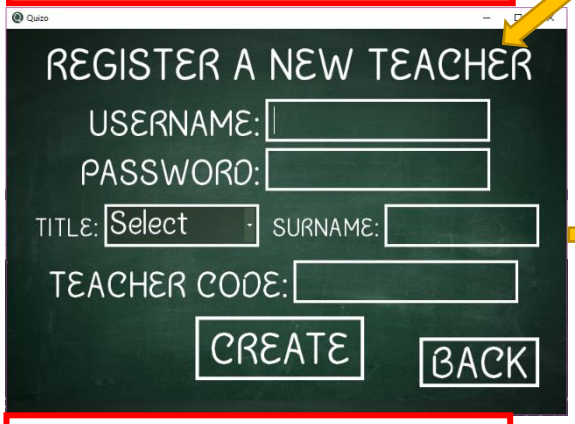
```
def Register(self):
    TeacherRegistration_Window.show() ## SHOWS TEACHER REGISTRATION WINDOW
    Login_Window.hide() ## HIDES LOGIN WINDOW
```

```
def Back(self):
    Login_Window.show() ## SHOWS LOGIN WINDOW
    TeacherRegistration_Window.hide() ## HIDES TEACHER REGISTRATION WINDOW
```

In order to test that the program can swap between windows, I will use the "Register New Teacher" and "Back" buttons multiple times to check that they are working as intended.

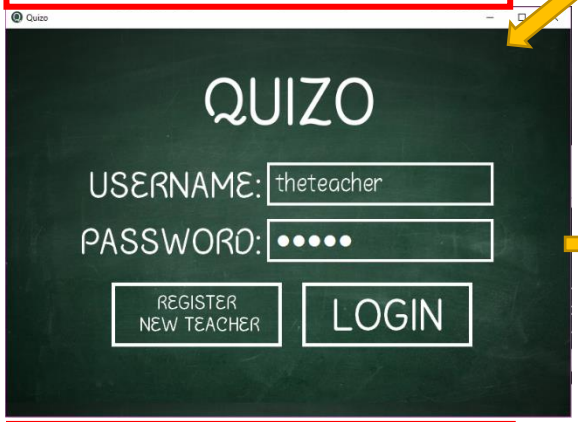
Upon start-up, I entered some details into the Login Screen

The Register New Teacher Button was pressed



The Registration Screen was shown and the Login Screen was hidden

I entered some details into this screen and then pressed the Back Button

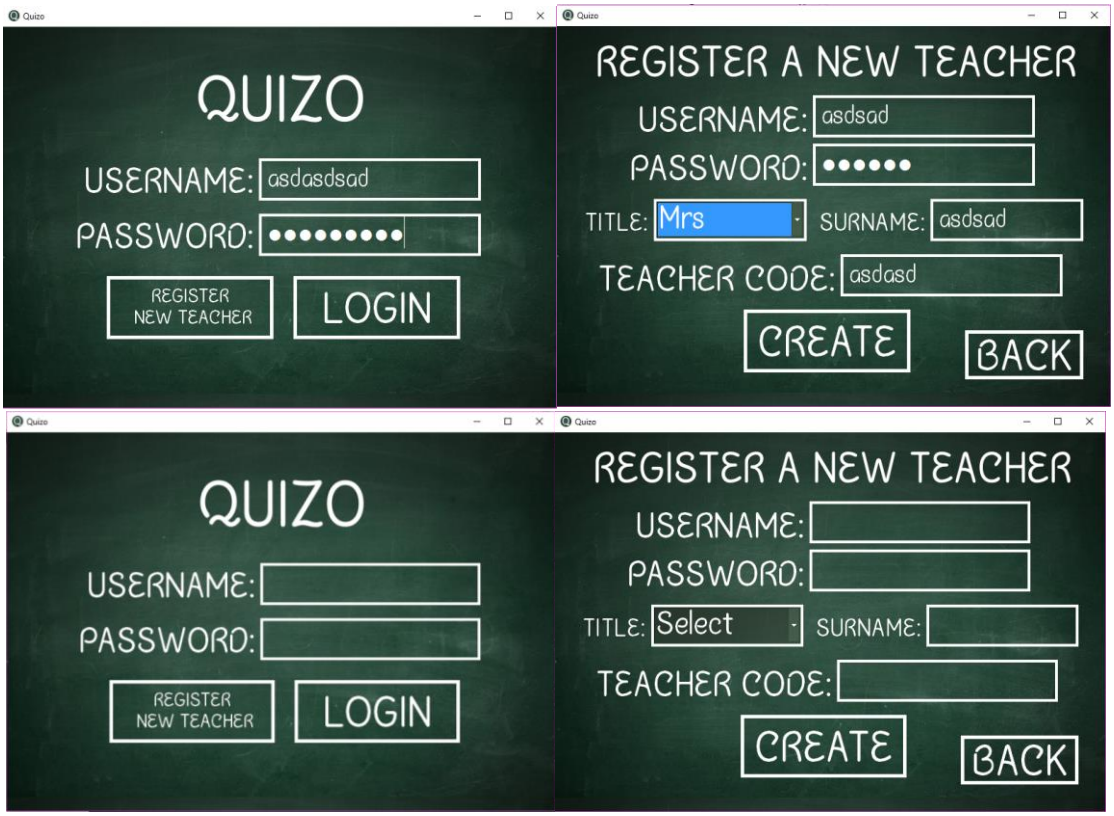


The Login Screen was shown again, along with the data entered before

Upon pressing the Register Button again, the Registration Screen appeared with the data entered before

From the test that I carried out I can see that the showing and hiding of windows worked as expected to seamlessly move between windows, however, upon showing a window that had previously been used, the same data was shown. For a school where multiple users may use the same instance of a program one after the other, having this be the case would waste time, as the next user would have to delete any details currently entered, and then enter their own. Although not being a large issue, it feels like something that should be addressed. For this reason, I have added code for whenever a window is hidden, the contents of its widgets are cleared, which can be seen in the code below along with a test.

```
def Register(self):
    TeacherRegistration_Window.show() ## SHOWS TEACHER REGISTRATION WINDOW
    Login_Window.hide() ## HIDES LOGIN WINDOW
    self.useredit.clear() ## CLEARS THE USERNAME BOX
    self.passedit.clear() ## CLEARS THE PASSWORD BOX

def Back(self):
    Login_Window.show() ## SHOWS LOGIN WINDOW
    TeacherRegistration_Window.hide() ## HIDES TEACHER REGISTRATION WINDOW
    self.codeedit.clear() ## CLEARS THE TEACHER CODE BOX
    self.useredit.clear() ## CLEARS THE USERNAME BOX
    self.suredit.clear() ## CLEARS THE SURNAME BOX
    self.passedit.clear() ## CLEARS THE PASSWORD BOX
    self.comboBox.setCurrentIndex(0) ## RESETS THE TITLE DROP DOWN TO SELECT
```



As can be seen above, when I entered in some data and then swapped between the windows, when I returned to the Login Screen, and Registration Screen, all the details had been cleared as expected.

OBJECTIVE 8 COMPLETED – USER CAN SEAMLESSLY SWAP BETWEEN WINDOWS

EFFICIENCY FIX – FINDING USERS

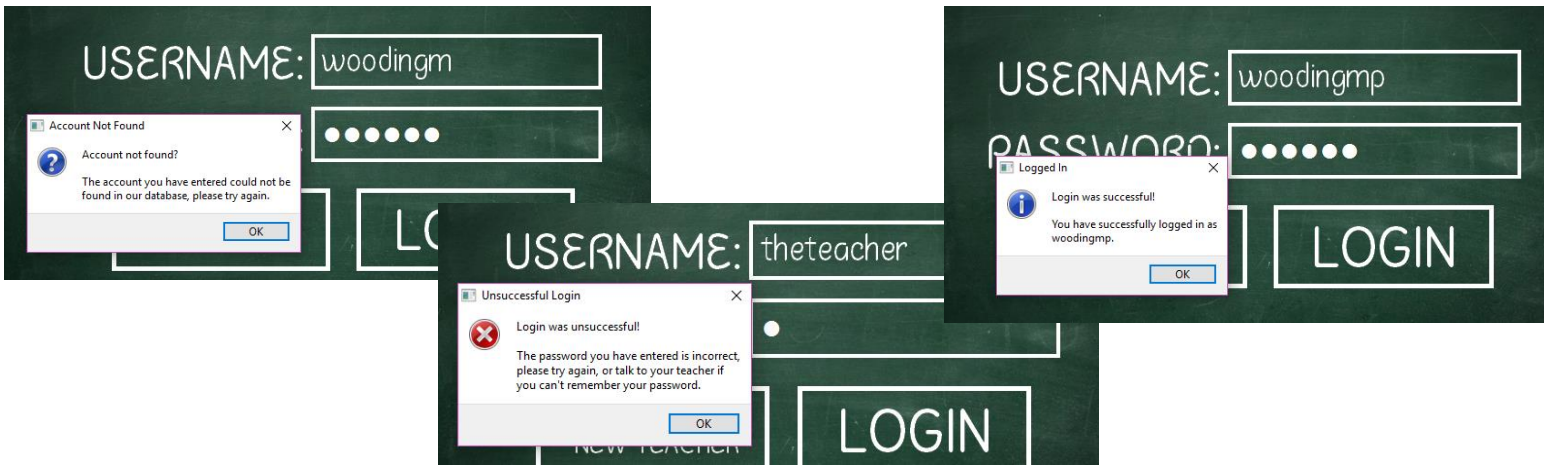
When I originally created the login screen, I had written some code to find whether the username the user had entered belonged to an account in the database, and whether or not they were a student or teacher. The original code can be seen below.

```
query = ["SELECT Username,Password FROM Students;","SELECT Username,Password FROM Teachers;"] ## SELECTS USERNAME AND PASSWORD FOR ALL ACCOUNTS
cur.execute(query[0]) ## EXECUTES STUDENT SEARCH
allstudents = cur.fetchall() ## FETCHES ALL DATA FROM STUDENT QUERY
cur.execute(query[1]) ## EXECUTES TEACHER SEARCH
allteachers = cur.fetchall() ## FETCHES ALL DATA FROM TEACHER QUERY

tempuser = [] ## RESETS TEMPORARY USER
access = "" ## RESETS CURRENT USERS ACCESS
for account in allstudents: ## SEARCHES THROUGH ALL STUDENTS
    if username == account[0]: ## CHECKS TO SEE IF USERNAMES ARE THE SAME
        tempuser = account ## SETS THIS ACCOUNT TO THE NEW TEMPORARY USER
        access = "Student" ## SETS USER'S ACCESS TO STUDENT
if access == "": ## CHECKS TO SEE IF THE ACCOUNT HAS NOT BEEN FOUND YET
    for account in allteachers: ## SEARCHES THROUGH ALL TEACHERS
        if username == account[0]: ## CHECKS TO SEE IF THE USERNAMES ARE THE SAME
            tempuser = account ## SETS THIS ACCOUNT TO THE NEW TEMPORARY USER
            access = "Teacher" ## SETS USER'S ACCESS TO TEACHER
```

While this code did work as intended, one issue with it is that it is having to fetch all the user accounts from the database just to check whether a user exists or not. While developing the program, this doesn't cause any issue in terms of how long it takes to fetch all this data and then search through it all using python and for loops. However, the RGS has around 1400 students aswell as staff, who will all hopefully have accounts on the program. When the program is having to fetch this much data and go through a loop this many times, it is possible that the program will slow down which is not desirable.

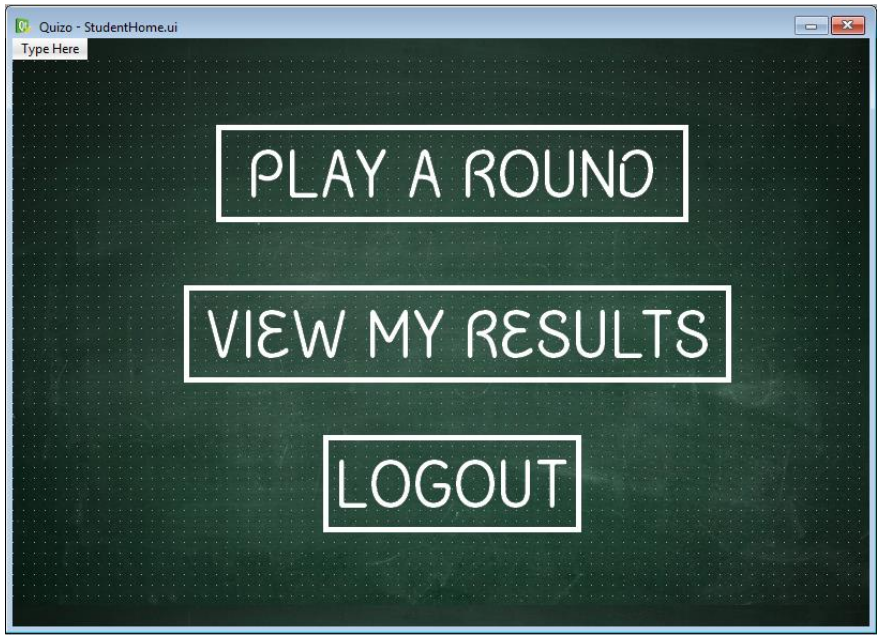
When writing the code for searching for usernames in the Teacher Registration Screen, I used a different method for seeing whether a username existed in the database, which used a WHERE statement is SQLite, and only one for loop in python. This method of finding a user would take considerably less time compared to my first method as more users are added to the database. For this reason I have decided to use this method in the Login Screen now, and its implementation can be seen below, alongside a test to show that it functions the same way.

```
access = "" ## RESETS CURRENT USER'S ACCESS
if username.isalnum() == True: ## CHECKS THAT THE USERNAME CONTAINS ONLY ALPHANUMERICAL VALUES
    tables = ["Teachers","Students"] ## LIST CONTAINING TABLE NAMES TO BE SEARCHED
    for x in tables: ## STARTS A FOR LOOP FOR SEARCHING THE TABLES
        query = "SELECT Username,Password FROM "+x+" WHERE Username='"+username+"';" ## CREATES THE QUERY FOR SEARCHING THE CURRENT TABLE
        cur.execute(query) ## EXECUTES THE QUERY
        temp = cur.fetchall() ## FETCHES ANY DATA FROM THE DATABASE THAT WAS SELECTED
        if temp!= []: ## CHECKS IF ANY DATA WAS FETCHED
            access = x ## SETS USER'S ACCESS LEVEL
            tempuser = temp[0] ## SETS THE ACCOUNT TO THE NEW TEMPORARY USER
            break ## BREAKS OUT OF THE LOOP IF THE USERNAME HAS BEEN FOUND
```



CREATING THE STUDENT HOME SCREEN

When logging into Quizo, depending on the user's access level, the user will be directed to their home screen. The first home screen I will deal with is the student home screen. From my original design, I have used QtDesigner to generate the three boxes required for movement in the program: Play a Round, View Results, and Logout. I used the same methods and style sheets as I did in the previous windows to get the correct aesthetic for the buttons, and I have saved the ui file in the same folder to ensure it works correctly. I have written a small bit of code that just prints the name of the button when it is clicked to make sure that python recognises the widgets. The code and tests can be seen below.




```
Login_class = uic.loadUiType("LoginWindow.ui")[0] ## LOADS UI FILE
TeacherRegistration_class = uic.loadUiType("TeacherRegistration.ui")[0] ## LOADS UI FILE
StudentHome_class = uic.loadUiType("StudentHome.ui")[0] ## LOADS UI FILE
```

Added at beginning of program

```
class StudentHome(QtGui.QMainWindow,StudentHome_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.setupUi(self)
        self.playbutton.clicked.connect(self.play) ## STARTS PLAY FUNCTION WHEN PLAY BUTTON CLICKED
        self.resultsbutton.clicked.connect(self.results) ## STARTS RESULTS FUNCTION WHEN RESULTS BUTTON IS CLICKED
        self.logoutbutton.clicked.connect(self.logout) ## STARTS LOGOUT FUNCTION WHEN LOGOUT BUTTON CLICKED

    def logout(self):
        print("LOGOUT PRESSED")

    def play(self):
        print("PLAY PRESSED")

    def results(self):
        print("RESULTS PRESSED")
```

Added new class in program

```
Login_Window = LoginWindow(None)
TeacherRegistration_Window = TeacherRegistration(None)
StudentHome_Window = StudentHome(None)
```

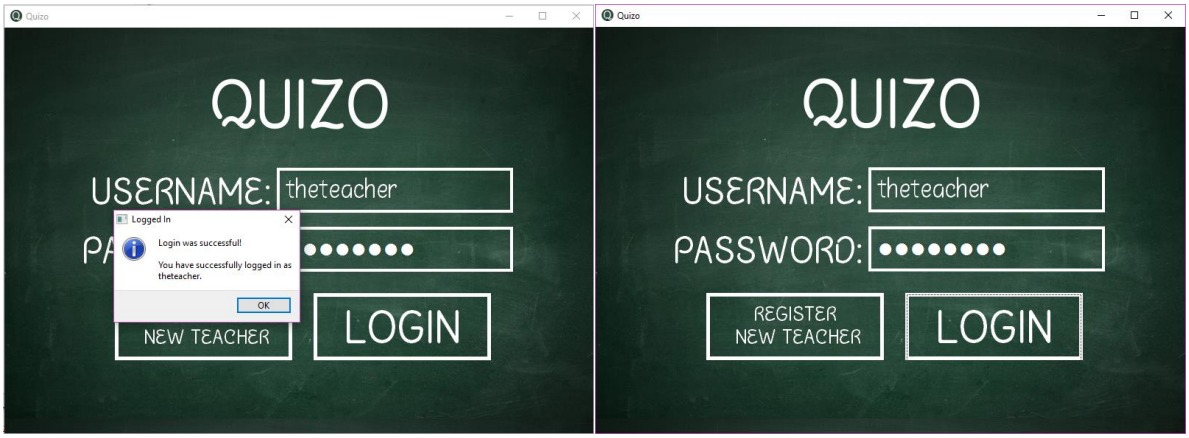
Added at end of program

```
if hashedattempt == tempuser[1]: ## CHECKS TO SEE IF THE HASHED ATTEMP
    CreateOutbox("Logged In","Login was successful!",("You have succes
    if access == "Students":
        StudentHome_Window.show() ## SHOWS STUDENT HOME SCREEN
        Login_Window.hide() ## HIDES LOGIN SCREEN
else:
    CreateOutbox("Unsuccessful Login","Login was unsuccessful!","The pa
```

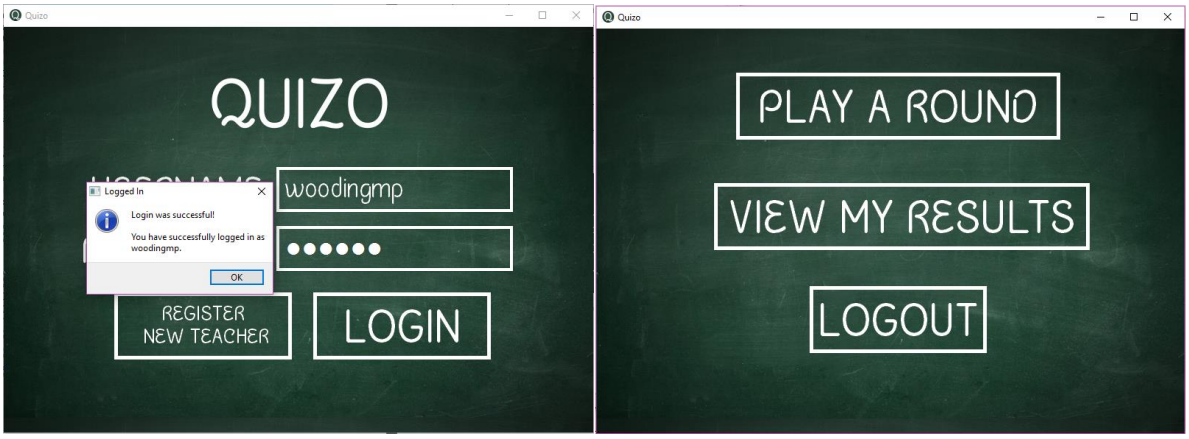
Added section to show the student home screen if logged in user is a student

Test Number	Test explanation	Test purpose
1	Logging in as a teacher	To check that access levels are working correctly and a teacher can't reach the student's home
2	Logging in as a student	To check that the student is taken to the student home screen
3	Pressing the "Play a Round" button	To check that the program recognises the button click
4	Pressing the "View My Results" button	To check that the program recognises the button click
5	Pressing the "Logout" button	To check that the program recognises the button click

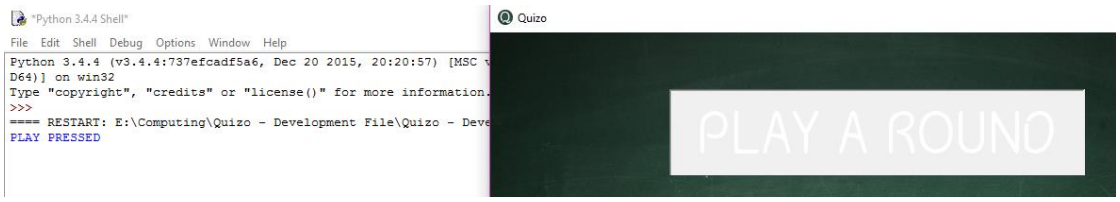
Test 1 – Successful – Logging in as a teacher created the message for a successful login, but remained on the Login Screen



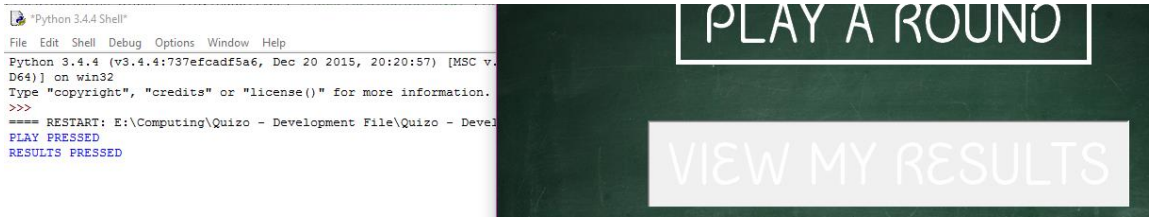
Test 2 – Successful – Logging in as a student created the message for a successful login, and then moved to the Student Home Screen



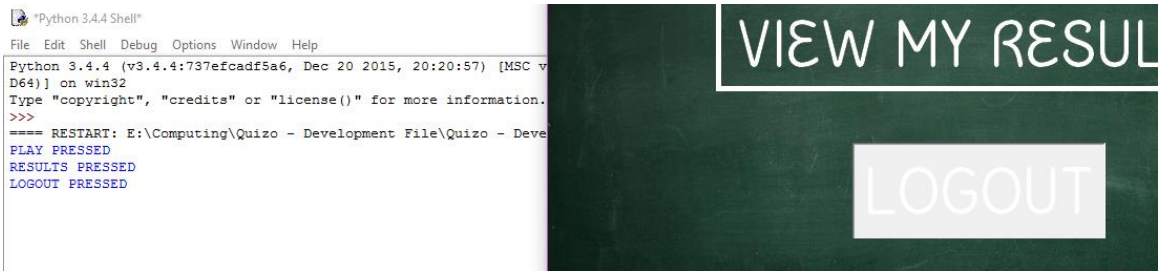
Test 3 – Successful – Pressing the "Play A Round" button printed the message "PLAY PRESSED" to the python IDLE as expected



Test 4 – Successful – Pressing the "View My Results" button printed the message "RESULTS PRESSED" to the python IDLE as expected

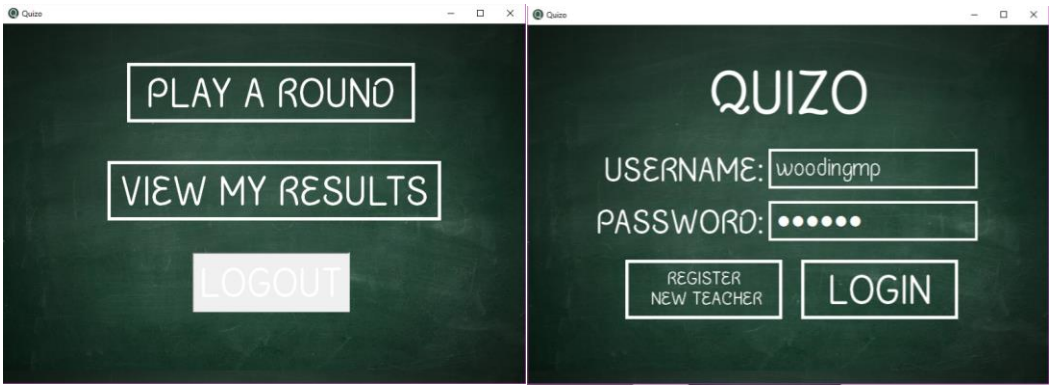


Test 5 – Successful – Pressing the "Logout" button printed the message "LOGOUT PRESSED" to the python IDLE as expected



As all of these tests were successful, I can now add the ability to move back to the login screen when the Logout button is pressed. The code for this can be seen below along with a test showing it working.

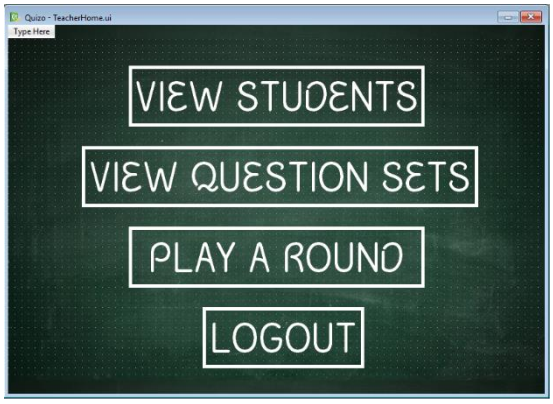
```
def logout(self):
    Login_Window.show() ## SHOWS THE LOGIN WINDOW
    StudentHome_Window.hide() ## HIDES THE STUDENT HOME WINDOW
```



OBJECTIVE 29 COMPLETED – USERS CAN LOGOUT

CREATING THE TEACHER HOME SCREEN

Similar to the student home screen, the teacher home screen is just a menu with a number of buttons, that can only be accessed by teacher accounts. Using my original design, I have created the window in QtDesigner, which can be seen to the right, and I have implemented the code to let teacher accounts access the window, which can be seen below. I have also done some tests just like I did with the Student Home Screen to check all the buttons are working.




```
TeacherHome_class = uic.loadUiType("TeacherHome.ui")[0] ## LOADS UI FILE
```

Added at beginning of program with the other " class" variables

```
if access == "Students":
    StudentHome_Window.show() ## SHOWS STUDENT HOME SCREEN
else:
    TeacherHome_Window.show() ## SHOWS TEACHER HOME SCREEN
Login_Window.hide() ## HIDES LOGIN SCREEN
```

Added in login function so that teacher accounts will be directed to the teacher home screen

```
class TeacherHome(QtGui.QMainWindow,TeacherHome_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.setupUi(self)
        self.playbutton.clicked.connect(self.play) ## STARTS PLAY FUNCTION WHEN PLAY BUTTON CLICKED
        self.studbutton.clicked.connect(self.stud) ## STARTS STUD FUNCTION WHEN VIEW STUDENTS BUTTON CLICKED
        self.setbutton.clicked.connect(self.set) ## STARTS SET FUNCTION WHEN VIEW QUESTION SETS BUTTON IS CLICKED
        self.logoutbutton.clicked.connect(self.logout) ## STARTS LOGOUT FUNCTION WHEN LOGOUT BUTTON CLICKED

    def logout(self):
        Login_Window.show() ## SHOWS THE LOGIN WINDOW
        TeacherHome_Window.hide() ## HIDES THE STUDENT HOME WINDOW

    def play(self):
        print("PLAY PRESSED")

    def set(self):
        print("SETS PRESSED")

    def stud(self):
        print("STUDENTS PRESSED")
```

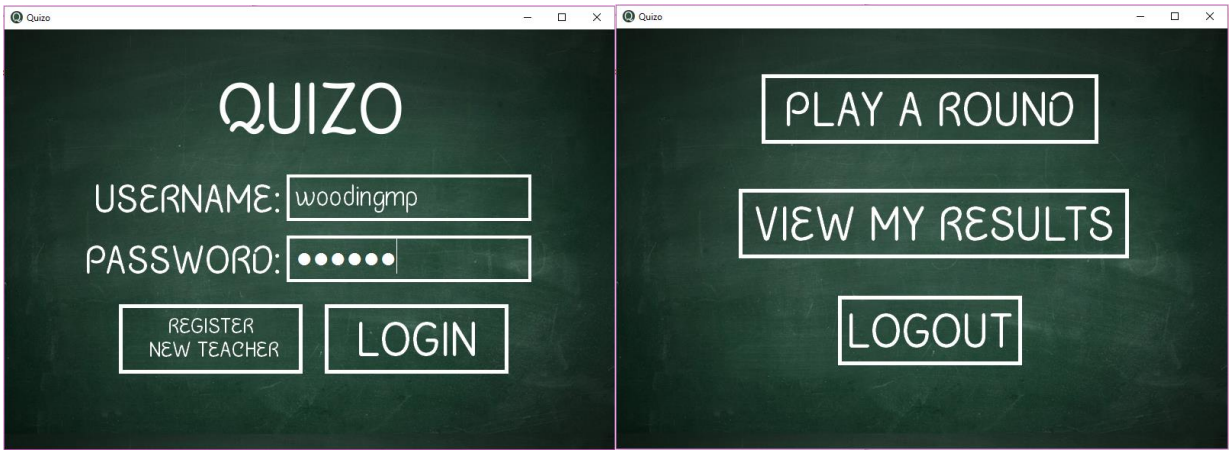
Added TeacherHome class which is almost identical to StudentHome class except for the different buttons

```
TeacherHome_Window = TeacherHome(None)
```

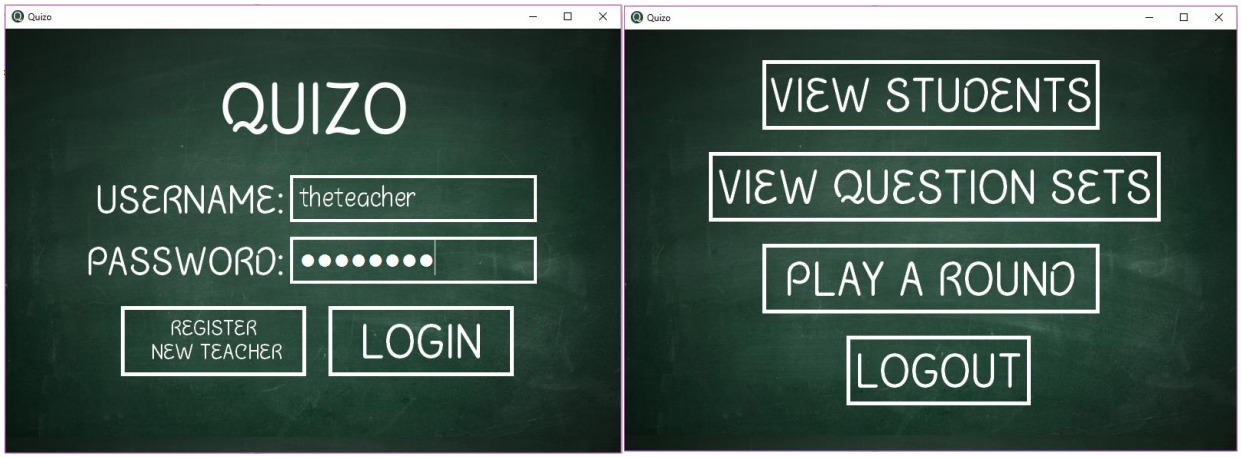
Creates the window for the teacher home screen

Test Number	Test explanation	Test purpose
1	Logging in as a student	To check that access levels are working correctly and a student is sent to the student home, not the teacher home
2	Logging in as a teacher	To check that the teacher is taken to the teacher home screen
3	Pressing the "View Students" button	To check that the program recognises the button click
4	Pressing the "View Question Sets" button	To check that the program recognises the button click
5	Pressing the "Play A Round" button	To check that the program recognises the button click
6	Pressing the "Logout" button	To check that the program recognises the button click

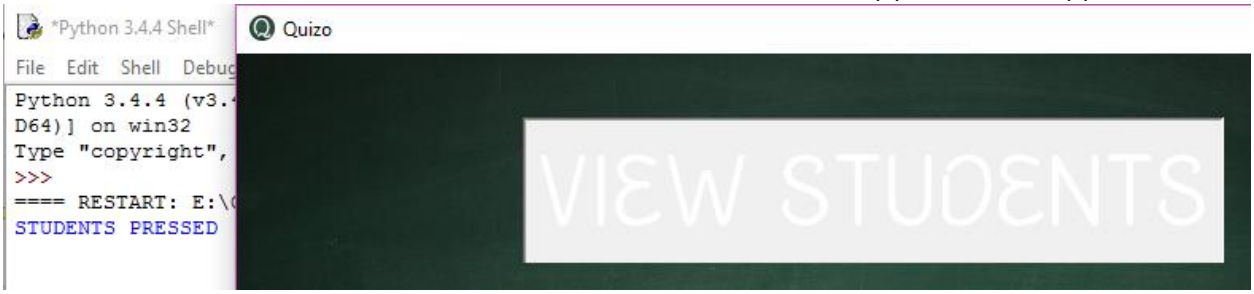
Test 1 – Successful – Student account is correctly sent to the student home rather than the teacher home



Test 2 – Successful – Teacher account is correctly sent to the teacher home screen



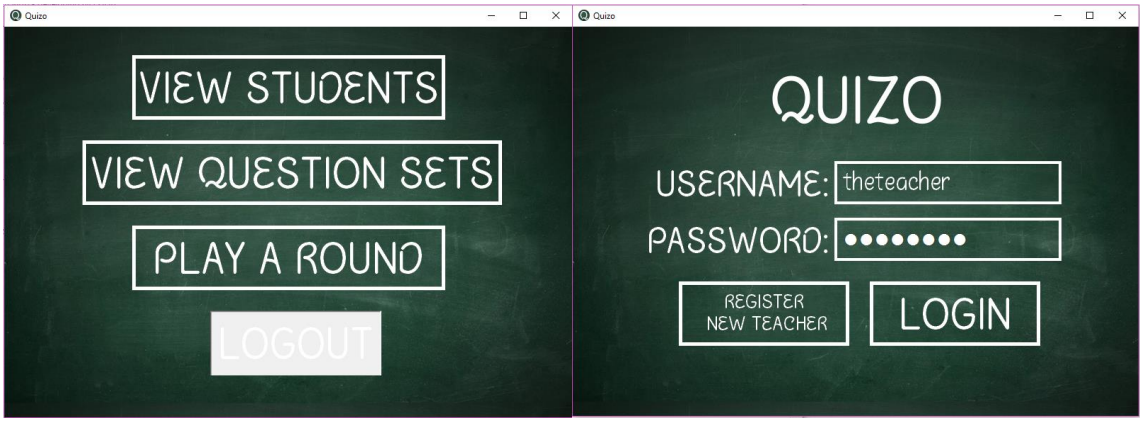
Test 3 – Successful – When clicked, "STUDENTS PRESSED" is correctly printed to the python IDLE



Test 4 – Successful – When clicked, "SETS PRESSED" is correctly printed to the python IDLE

Test 5 – Successful – When clicked, "PLAY PRESSED" is correctly printed to the python IDLE

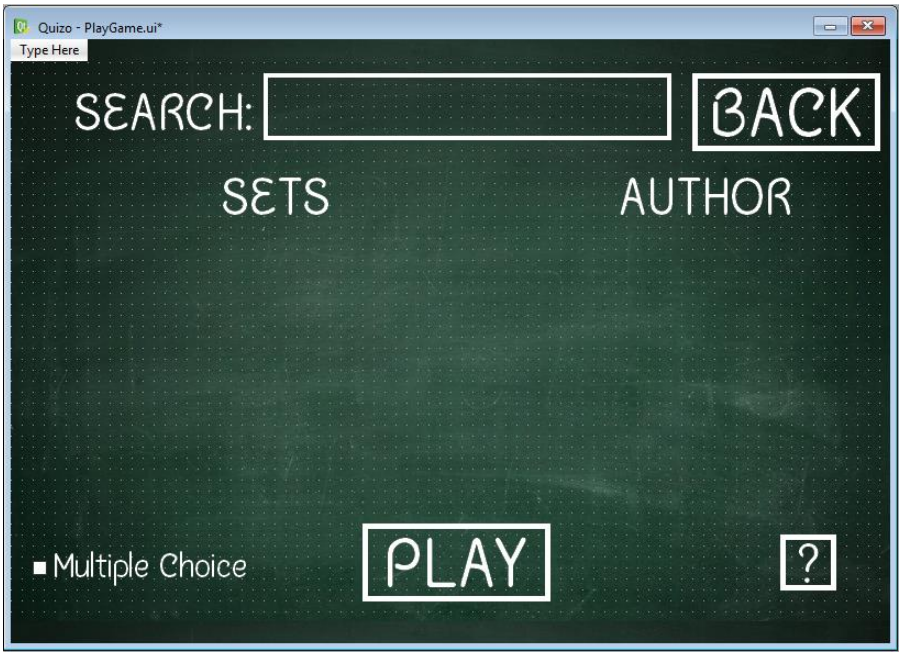
Test 6 – Successful – User is correctly returned to the login screen, and the Teacher Home Screen is hidden



As all tests were successful, I can continue work on other section of the program before coming back to add the functionality for moving towards later windows in the program.

CREATING THE PLAY GAME SCREEN

The next screen that I will create is the screen for selecting question sets to attempt. This screen will be used by both teachers and students, and will follow the design that I had created in the usability features section earlier in my documentation.



Having re-read over my documentation, I also realise that I mentioned the possibility of a random quiz button which I left out of my original design, however I have decided that I will attempt to add the functionality. I have created the design for this window in QtDesigner which can be seen to the right, and I have written the code into my program to be able to show the window when either a teacher or student pressed the "Play A Round" button.

```
Play_class = uic.loadUiType("PlayGame.ui")[0] ## LOADS UI FILE
```

Added at the beginning of the program

```
def Login(self):
    global access ## GLOBALISES ACCESS LEVEL SO ALL WINDOWS CAN USE IT
```

Needed to globalise access variable so that the new window could know the access level of the user

```
def play(self):
    PlayGame_Window.show() ## SHOWS THE PLAY GAME WINDOW
    StudentHome_Window.hide() ## HIDES THE STUDENT HOME WINDOW
```

Added code in student home to allow students to reach the new screen

```
def play(self):
    PlayGame_Window.show() ## SHOWS THE PLAY GAME WINDOW
    TeacherHome_Window.hide() ## HIDES THE TEACHER HOME WINDOW
```

Added code in teacher home to allow teachers to reach the new screen

```
class PlayGame(QtGui.QMainWindow,Play_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.setupUi(self)
        self.backbutton.clicked.connect(self.Back) ## CHECKS IF BACK BUTTON IS PRESSED
        self.playbutton.clicked.connect(self.startgame) ## CHECKS IF PLAY BUTTON IS PRESSED
        self.randombutton.clicked.connect(self.randomquiz) ## CHECKS IF RANDOM BUTTON IS PRESSED

    def Back(self):
        if access == "Teachers": ## CHECKS IF USER IS A TEACHER
            TeacherHome_Window.show() ## SHOWS THE TEACHER HOME WINDOW
        else:
            StudentHome_Window.show() ## SHOWS THE STUDENT HOME WINDOW
        PlayGame_Window.hide() ## HIDES THE PLAY GAME WINDOW

    def startgame(self):
        print("START GAME PRESSED")

    def randomquiz(self):
        print("RANDOM PRESSED")
```

Added the Play Game class to initialise the window and allow the user to go back depending on their access level

```
PlayGame_Window = PlayGame(None)
```

Creates the window for the Play Game screen

In order to make sure that the window is being shown when prompted, and allows the user to return to the previous screen, I have done a test trying to reach the window when logged in as a student, and again as a teacher.

I have also decided that now having built four windows, I will no longer do extra tests to make sure each button is working, as thus far everything has worked without an issue, and I can safely assume the buttons will work as expected.

Test 1 – Successful – Going to the Play Game Screen as a teacher, and then returning to the Teacher Home as expected

Test 2 – Successful – Going to the Play Game Screen as a student, and then returning to the Student Home as expected

OBJECTIVE 27 COMPLETED – USERS CAN ACCESS THE PLAY GAME SCREEN NO MATTER THEIR ACCESS LEVEL

CREATING A TABLE FOR QUESTION SETS

Now that I have created the play game screen, and users are able to reach it, I need to add functionality for fetching all the possible question sets that a user can try and displaying in the table view that I have added in the QtDesigner. The only data that needs to be fetched from the database at this is screen is a list of all the set names, along with their respective authors which I plan to do by executing the SQLite statement:

Table: Sets

	SID	SetName	Author
	Filter	Filter	Filter
1	1	TEMP	TEMP
2	2	Set	theteacher
3	3	Entertainment	theteacher

```
SELECT SetName,Author FROM Sets;
```

SetName	Author
TEMP	TEMP
Set	theteacher
Entertainment	theteacher

"SELECT SetName, Author FROM Sets;"

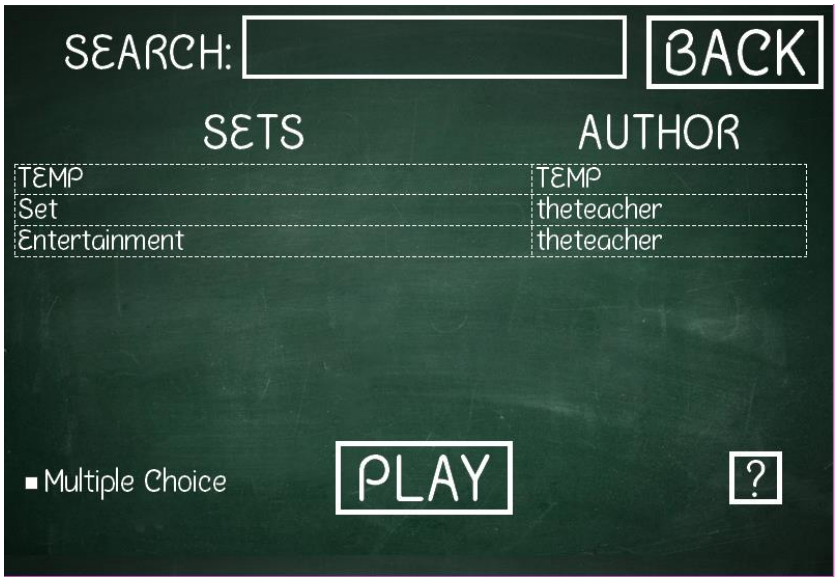
A quick test of this in the SQLite browser shows that this statement will do the desired effect when executed.

Having done some research into tables in PyQt I have decided to use a QTableView with a table model instead of a QTableWidget to generate the table interface in the program, as it is simple and doesn't create a "WidgetItem" for every bit of data that goes into the table like a QTableWidget does. This takes up less memory, and is also faster to process as all that is done is copying data from the fetched list to the model, and then the model is shown. The code for this can be seen below along with a screenshot showing that the correct sets and authors can be seen in the table when the program is run.


```
query = "SELECT SetName,Author FROM Sets;" ## QUERY TO FETCH ALL SET NAMES AND AUTHORS
cur.execute(query) ## EXECUTES QUERY
data = cur.fetchall() ## FETCHES DATA FROM DATABASE
model = QtGui.QStandardItemModel(len(data),2) ## CREATES A MODEL TO PUT THE DATA IN
for row in range(0,len(data)): ## LOOPS THROUGH EACH SET
    for column in range(0,2): ## LOOPS THROUGH SET NAME AND AUTHOR FOR THE CURRENT SET
        model.setData(model.index(row,column),str(data[row][column])) ## SETS THE MODEL DATA ACCORDING TO THE FETCHED DATA
self.setstable.setModel(model) ## SETS THE MODEL TO THE TABLE VIEW
self.setstable.show() ## SHOWS THE TABLE
```

As can also be seen above, the table is squished into the left side of the screen, so I have added two lines of code to stretch the columns to fit the screen, which can be seen below.

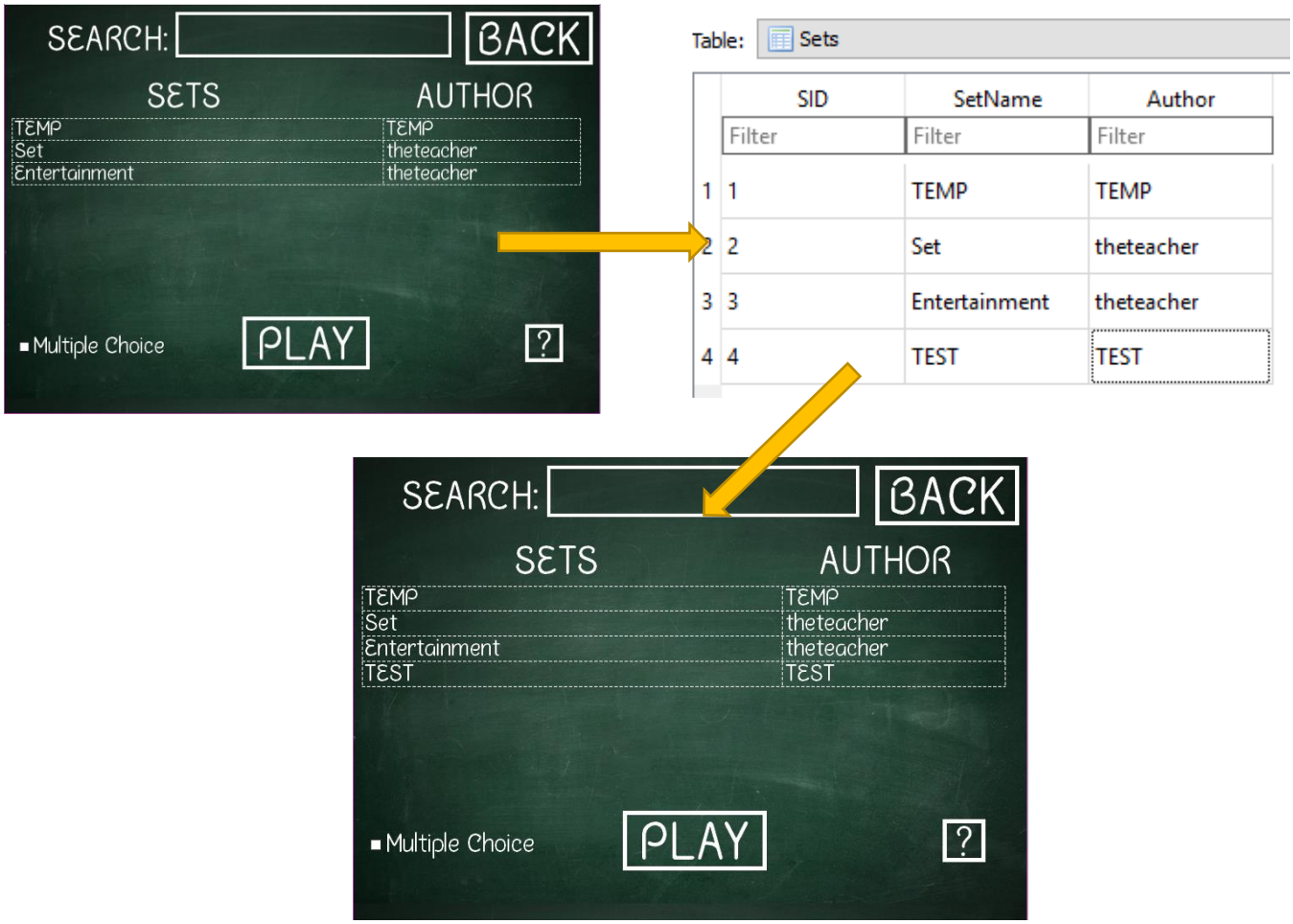
```
self.setstable.setColumnWidth(0,500) ## SETS COLUMN WIDTH FOR SETS COLUMN
self.setstable.setColumnWidth(1,265) ## SETS COLUMN WIDTH FOR AUTHOR COLUMN
self.setstable.show() ## SHOWS THE TABLE
```



At the moment, the table is displaying the correct data, however I have written the code for the table in the "init" method in the class, meaning that it only runs when the class is first initialised. Although this is okay now as I haven't added the ability to create new sets yet, this could cause an issue, because in order to refresh the table, you would need to restart the program.

In order to fix this issue, I have found that there is a showEvent method that can be used with pyqt which will be run whenever the window is shown, which will allow the table to refresh whenever the user moves back to this window. The code can be seen below, along with a test, showing that when I add a new question set "TEST" by author "TEST" to the table, it is updated without having to relaunch the program.

```
def __init__(self,parent=None):
    QtGui.QMainWindow.__init__(self,parent)
    QtGui.QMainWindow.showEvent(self,parent) ## RUNS SHOWEVENT FUNCTION WHENEVER WINDOW IS SHOWN
    self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
    self.setupUi(self)
    self.backbutton.clicked.connect(self.Back) ## CHECKS IF BACK BUTTON IS PRESSED
    self.playbutton.clicked.connect(self.startgame) ## CHECKS IF PLAY BUTTON IS PRESSED
    self.randombutton.clicked.connect(self.randomquiz) ## CHECKS IF RANDOM BUTTON IS PRESSED

def showEvent(self,parent=None):
    query = "SELECT SetName,Author FROM Sets;" ## QUERY TO FETCH ALL SET NAMES AND AUTHORS
    cur.execute(query) ## EXECUTES QUERY
    data = cur.fetchall() ## FETCHES DATA FROM DATABASE
    model = QtGui.QStandardItemModel(len(data),2) ## CREATES A MODEL TO PUT THE DATA IN
    for row in range(0,len(data)): ## LOOPS THROUGH EACH SET
        for column in range(0,2): ## LOOPS THROUGH SET NAME AND AUTHOR FOR THE CURRENT SET
            model.setData(model.index(row,column),str(data[row][column])) ## SETS THE MODEL DATA ACCORDING TO THE FETCHED DATA
    self.setstable.setModel(model) ## SETS THE MODEL TO THE TABLE VIEW
    self.setstable.setColumnWidth(0,500) ## SETS COLUMN WIDTH FOR SETS COLUMN
    self.setstable.setColumnWidth(1,265) ## SETS COLUMN WIDTH FOR AUTHOR COLUMN
    self.setstable.show() ## SHOWS THE TABLE
```



Test – Successful – When loading up the program, the three existing sets where displayed in the table. I then tabbed out and added a new set to the Sets table, wrote the changes, and then returned to the program. I then pressed the back button and then went back into the play game screen, and the table had been updated with the new set shown.

OBJECTIVES 51, AND 52 COMPLETED – ALL QUESTION SETS AND THEIR AUTHORS ARE SUCCESSUFLLY FETCHED FROM THE DATABASE, AND ARE DISPLAYED IN A TABLE

CREATING SEARCH BARS

In my program there are multiple times where a table is on screen, and once I take my program back to the school, hopefully will be filled with lots of question sets, and users. For example, when a user is trying to attempt a new test, they should be able to search through the sets in order to find one they want to attempt faster. In order to have a more intuitive program, I want to create a dynamic search bar, meaning that the table is searched as the user is typing, removing the need for an extra button on screen to press to search. In order to do this, I will be using the "textChanged" event for line edit, which executes a function whenever the text inside the text box is changed.

In order to make the search function work, instead of searching through all the fetched data from the database for sets with the same name as the user's entry, I will do all the filtering in SQLite using the

WHERE and LIKE conditions. The LIKE condition in SQLite means that only data that is identical to the chosen string will be fetched, as can be seen below in my test in the SQLite browser.

```
1 SELECT SetName,Author FROM Sets WHERE SetName LIKE "Entertainment";
```

	SetName	Author
1	Entertainment	theteacher

Although this is useful when you know the exact name of the set, it cannot be expected of every user to remember the exact name of every set. For this reason, I have decided to add % symbols to the statement, which means that any sets containing the string in any capacity will be fetched.

```
1 SELECT SetName,Author FROM Sets WHERE SetName LIKE "%s%";
```

	SetName	Author
1	Set	theteacher
2	TEST	TEST

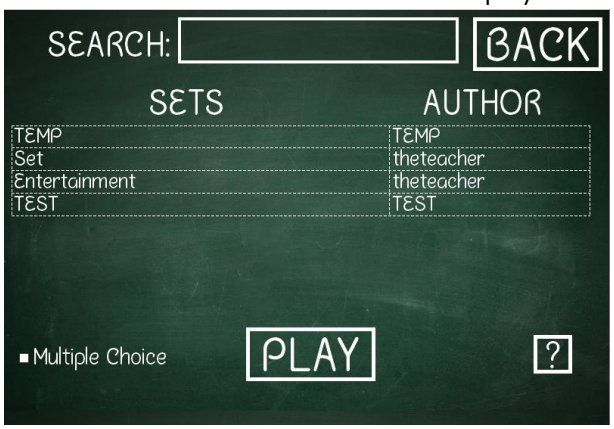
Now that I know the statement that I will be using to filter the database, I can add the functionality for searching into my program as can be seen below. The code is identical to the code for creating the table except for the SQLite statement.

```
def __init__(self,parent=None):
    QtGui.QMainWindow.__init__(self,parent)
    QtGui.QMainWindow.showEvent(self,parent) ## RUNS SHOWEVENT FUNCTION WHENEVER WINDOW IS SHOWN
    self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
    self.setupUi(self)
    self.backbutton.clicked.connect(self.Back) ## CHECKS IF BACK BUTTON IS PRESSED
    self.playbutton.clicked.connect(self.startgame) ## CHECKS IF PLAY BUTTON IS PRESSED
    self.randombutton.clicked.connect(self.randomquiz) ## CHECKS IF RANDOM BUTTON IS PRESSED
    self.searchedit.textChanged.connect(self.search) ## RUNS SEARCH FUNCTION WHENEVER SEARCH BAR CONTENTS CHANGE

def search(self):
    word = "%" + self.searchedit.text() + "%" ## GETS STRING FROM SEARCH BAR
    query = 'SELECT SetName,Author FROM Sets WHERE SetName LIKE "'+word+'";' ## QUERY TO FETCH SET NAMES WITH SIMILAR TITLTES TO USER'S SEARCH
    cur.execute(query) ## EXECUTES QUERY
    data = cur.fetchall() ## FETCHES DATA FROM DATABASE
    model = QtGui.QStandardItemModel(len(data),2) ## CREATES A MODEL TO PUT THE DATA IN
    for row in range(0,len(data)): ## LOOPS THROUGH EACH SET
        for column in range(0,2): ## LOOPS THROUGH SET NAME AND AUTHOR FOR THE CURRENT SET
            model.setData(model.index(row,column),str(data[row][column])) ## SETS THE MODEL DATA ACCORDING TO THE FETCHED DATA
    self.setstable.setModel(model) ## SETS THE MODEL TO THE TABLE VIEW
    self.setstable.setColumnWidth(0,500) ## SETS COLUMN WIDTH FOR SETS COLUMN
    self.setstable.setColumnWidth(1,265) ## SETS COLUMN WIDTH FOR AUTHOR COLUMN
    self.setstable.show() ## SHOWS THE TABLE
```

Test Number	Search Bar Entry	Test Purpose
1	[EMPTY]	To check that all sets are shown when user is not searching
2	FAKE	To check that no sets are shown when a non-existent set is searched for

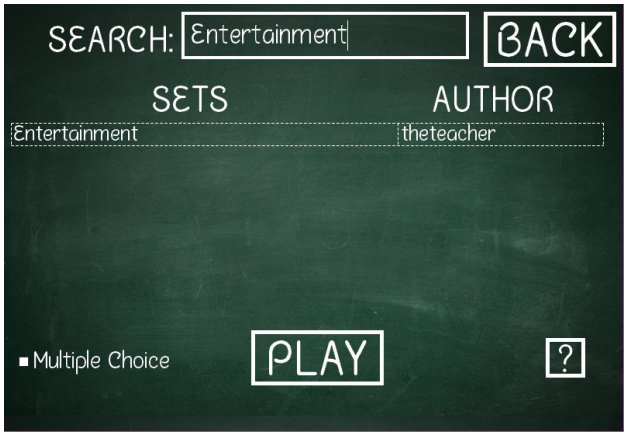
3	Entertainment	To check that the correct set is shown when user searches for it
4	S	To check that sets that are similar to the search are found

Test 1 – Successful – All four sets from the database were displayed in the table as expected

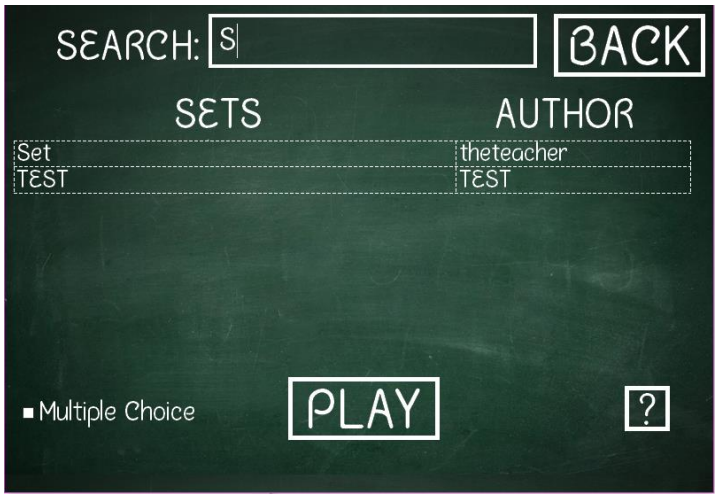


Test 2 – Successful – No sets were shown when a search that was similar to no sets was entered as expected

Test 3 – Successful – Only the entertainment set was shown in the table. If there are more than one set with the same name, the user will be able to differentiate the sets by author as it is a secondary key like the set name

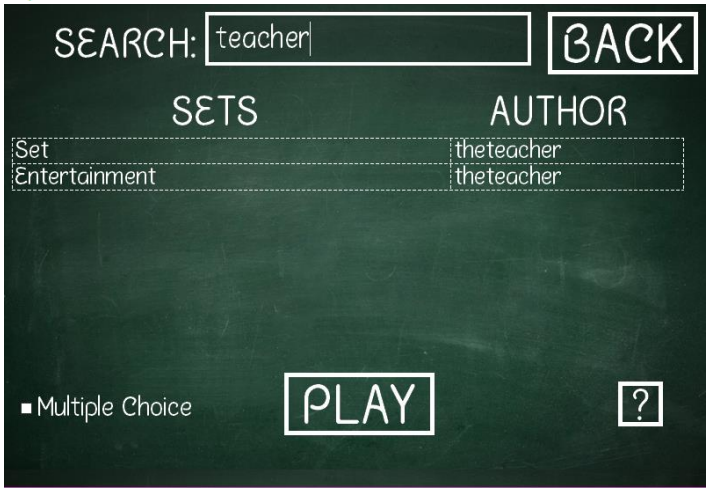


Test 4 – Successful – Both the "TEST" and "Arts & Literature" sets were shown as they contain an "s" in their title



After carrying out test three and realising that multiple tests with the same name could be created, but by different authors, I have decided to add an extra part to the SQLite statement, allowing the author section to be searched as well as set name, which can be seen below, alongside a quick test to show it works just the same.

```
query = 'SELECT SetName,Author FROM Sets WHERE SetName LIKE "'+word+'" OR Author LIKE "'+word+'";'
```

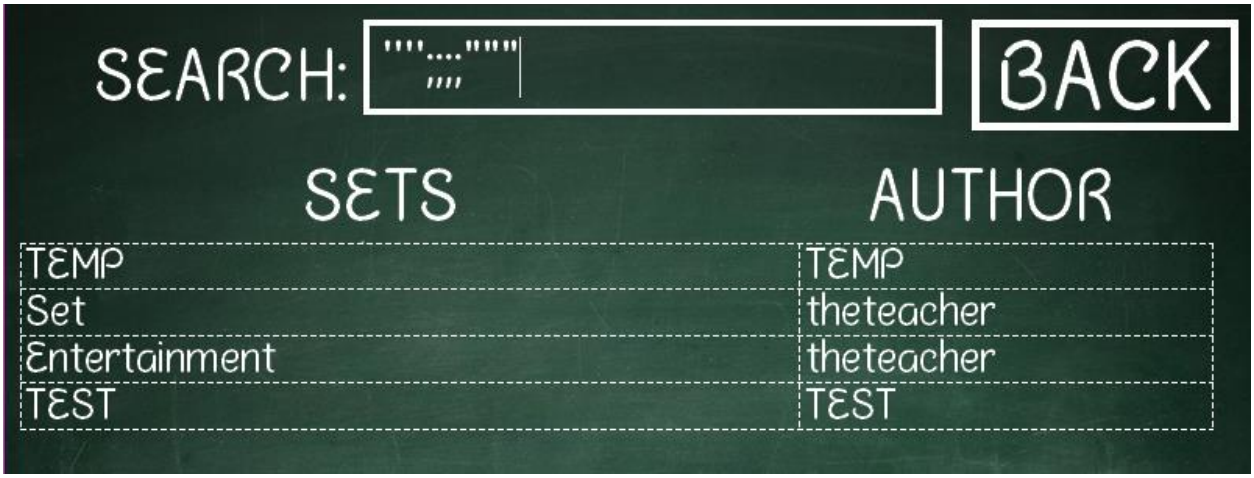


Additional Test – Successful – Only the two sets created by "theteacher" were shown when "teacher" was entered into the search bar

OBJECTIVE 11 COMPLETED – USERS CAN USE SEARCH BARS TO FILTER TABLES

Post-Comment: Having realised that I forgot to put in validation for this entry box in case a user tries to exploit the entry like I mentioned when creating the teacher registration screen, I have gone back and added and two lines of code that will strip the user's entry of any speech marks, apostrophes, or semicolons, which can be seen below.

```
for char in ["'",";",'"']: ## LOOPS THROUGH THE DISALLOWED CHARACTERS
    word = word.replace(char,"") ## REMOVES CHARACTERS FROM USER'S ENTRY
```



As can be seen above, all the characters do not affect the search function as they are removed from the user's entry before the SQLite statement is executed., and so the box now has correct validation.

THE RANDOM QUIZ BUTTON

The random quiz button is the small question mark in the bottom right of the play game screen, and it is designed to allow the user to press it and let them try a randomly selected question set. Its implementation shouldn't be too difficult to add as it just needs to be able to return the name and author of the randomly selected set.

In order for this functionality to work, I have imported the "random" module into the program, and have written the code for the program to print the randomly chosen set, which can be seen below.

```
def randomquiz(self):
    query = "SELECT SetName,Author FROM Sets;" ## QUERY TO FETCH ALL SETS FROM DATABASE
    cur.execute(query) ## EXECUTES QUERY
    data = cur.fetchall() ## FETCHES DATA FROM DATABASE
    randsetnum = random.randint(1,len(data)) ## CHOOSES RANDOM NUMBER BETWEEN 1 AND THE NUMBER OF SETS IN THE DATABASE
    playset = data[randsetnum-1] ## GETS THE SETNAME AND AUTHOR OF THE CHOSEN SET (-1 AS INDEXES START FROM 0)
    print(playset)
```

To test that the randomness of the function is working, I will press the button ten times and just check that the sets that are chosen are fairly "different". Randomness is difficult to test because the program could randomly select the same set four times in a row, so as long as it is selecting different sets at some point, that is random enough for me to be satisfied that it is working. Additionally, as more question sets are added, the probability of getting the same set twice in a row decreases.

Test – Successful – Each of the current sets were chosen at some point so I am satisfied with saying that the randomness of selecting a set is sufficient for the program.

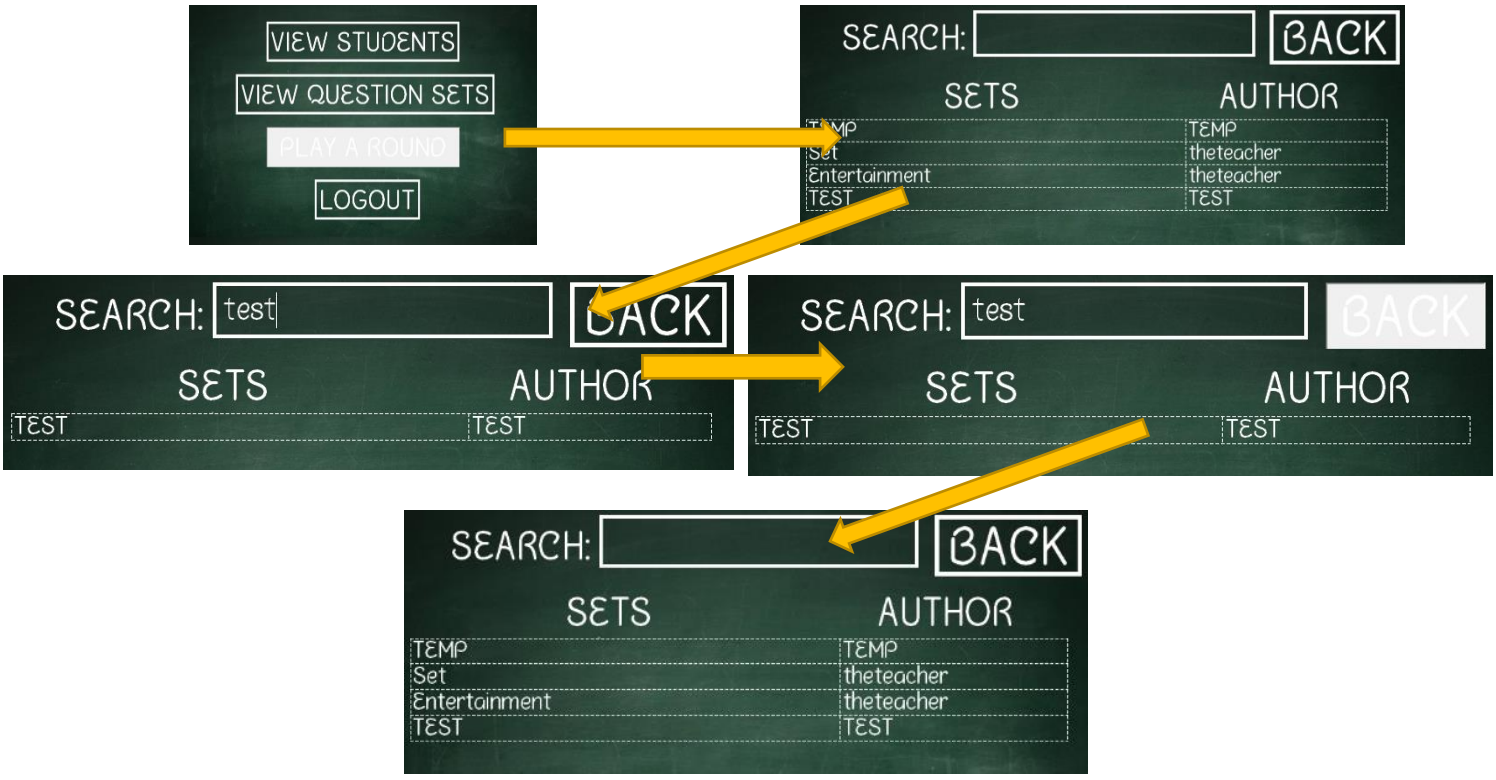
```
('TEMP', 'TEMP')
('TEMP', 'TEMP')
('Entertainment', 'theteacher')
('Entertainment', 'theteacher')
('TEST', 'TEST')
('TEMP', 'TEMP')
('TEST', 'TEST')
('Set', 'theteacher')
('Entertainment', 'theteacher')
('TEST', 'TEST')
```

BONUS OBJECTIVE COMPLETED – PROGRAM CAN RANDOMLY SELECT A SET FOR THE USER

EFFICIENCY FIX – TABLE CREATION

At the moment, I have the search function for when the user is trying to search through the table for a specific set, and also a function for when the window is shown that refreshes the table. Instead of having both functions that do practically the same thing, I am changing the showEvent function to just pass through to the search function. I have also added code to reset the search bar when the user exits this window, so that if they come back they have a fresh search bar. The code for this, and a test to show that the program works the same way can be seen below.

```
def showEvent(self,parent=None):
    self.search()

def Back(self):
    if access == "Teachers": ## CHECKS IF USER IS A TEACHER
        TeacherHome_Window.show() ## SHOWS THE TEACHER HOME WINDOW
    else:
        StudentHome_Window.show() ## SHOWS THE STUDENT HOME WINDOW
    self.searchedit.clear() ## CLEARS THE SEARCH BAR
    PlayGame_Window.hide() ## HIDES THE PLAY GAME WINDOW
```

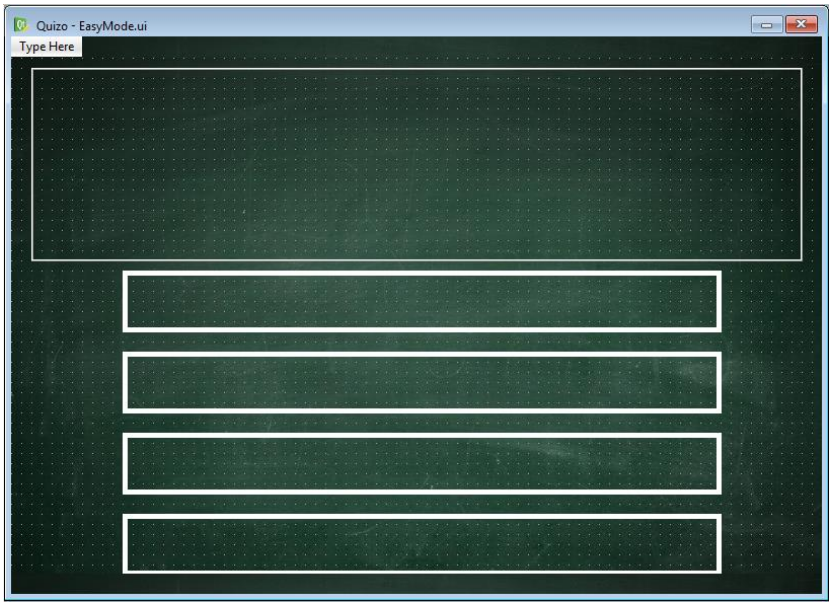


Test 1 – Successful – When showing the play game window for the first time, the table was successfully created without error, meaning the new code, passing the showEvent to the search function worked

Test 2 – Successful – When typing something into the search bar, upon leaving the window and returning, the bar was cleared and the table unfiltered

CREATING THE MULTIPLE CHOICE SCREEN

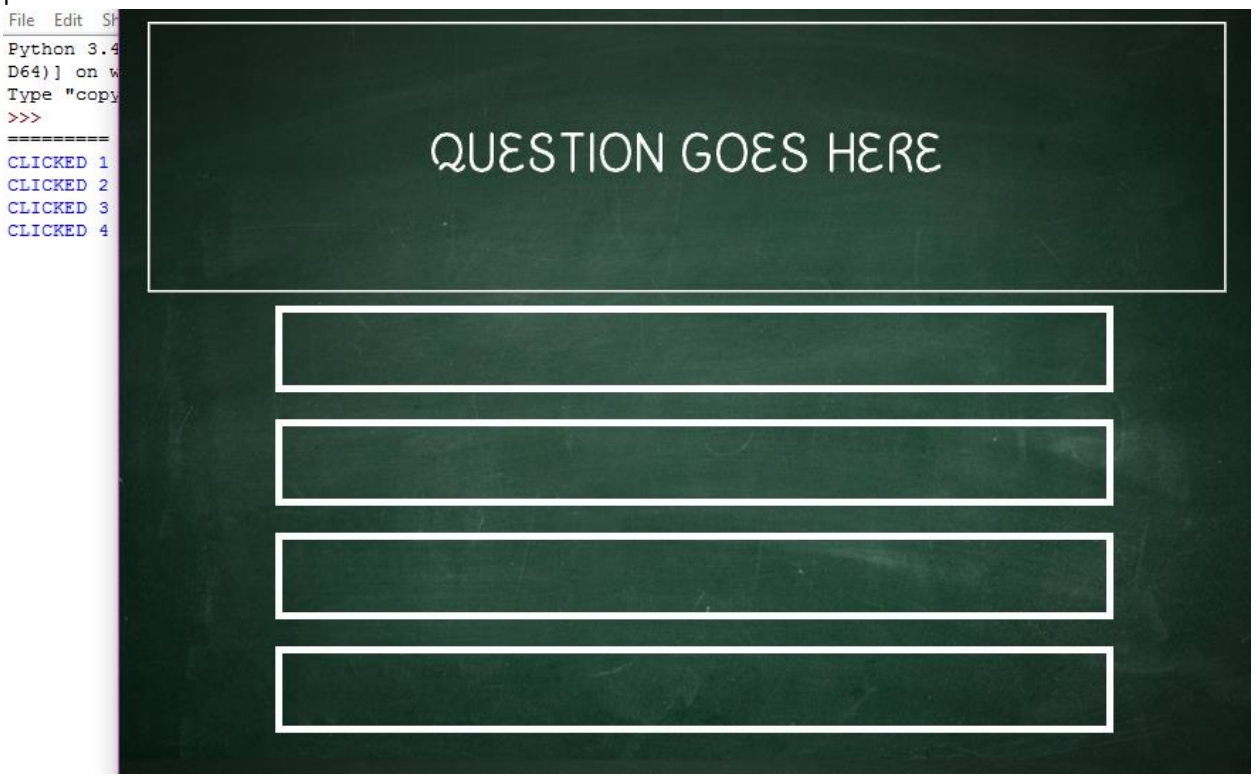
When using Quizo and attempting question sets, the user is able to choose whether they play with multiple choice or with typed out answers. The first of the modes I am going to create is multiple choice because I believe it will require more work to perfect. I began creating this window by following my initial design, and creating a PyQt window which can be seen to the right. For the sake of ease, I have decided to name multiple choice "easy mode" while programming, even though I realise it is subjective.



In the window is a text box at the top which is where questions will be displayed, and then four buttons for each of the possible answers. For this screen I have decided to code it in a separate file for now, as it allows me to access the screen faster for testing purposes, and because it's easier to find issues when the class is isolated, so the code for that is below.

```
## Multiple Choice Testing ##

## IMPORTS NECESSARY MODULES
import sys,os
import sqlite3 as lite
import random
from PyQt4 import QtCore,QtGui,uic

con = lite.connect('QuizoBase') ## CONNECTING TO THE DATABASE
cur = con.cursor()

Easy_class = uic.loadUiType("EasyMode.ui")[0] ## LOADS UI FILE

class Easy(QtGui.QMainWindow,Easy_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setupUi(self)
        self.ans1button.clicked.connect(self.click1) ## RUNS CLICK FUNCTION WHEN BUTTON 1 PRESSED
        self.ans2button.clicked.connect(self.click2) ## RUNS CLICK FUNCTION WHEN BUTTON 2 PRESSED
        self.ans3button.clicked.connect(self.click3) ## RUNS CLICK FUNCTION WHEN BUTTON 3 PRESSED
        self.ans4button.clicked.connect(self.click4) ## RUNS CLICK FUNCTION WHEN BUTTON 4 PRESSED
        self.questionlabel.setText("QUESTION GOES HERE") ## SETS TEXT FOR THE LABEL

    def click1(self):
        print("CLICKED 1")
    def click2(self):
        print("CLICKED 2")
    def click3(self):
        print("CLICKED 3")
    def click4(self):
        print("CLICKED 4")

app = QtGui.QApplication(sys.argv)
Easy_Window = Easy(None)
Easy_Window.show() ## SHOWS THE MULTIPLE CHOICE WINDOW
app.exec_()
```

The code that I have written simply adds some text to the label, and prints the button number that is pressed as can also be seen below.

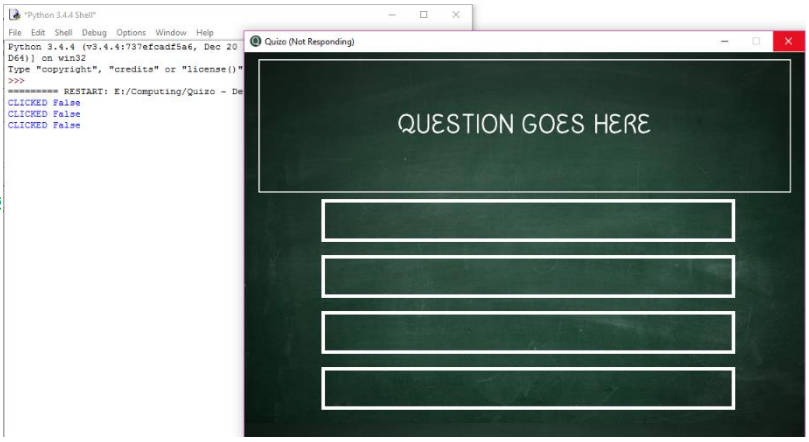


Although having a separate function for each button works as intended, as I start to add functionality for checking for correct answers, having four identical functions would be inefficient, and so instead of having four functions I need to find a way to pass parameters when pressing a button.

After some failed attempts that can be seen below, I decided to look online, and found a solution to my problem on a forum, where I found that I can use part of the "functools" module called partial, which will allow me to pass parameters when pressing a button as desired.

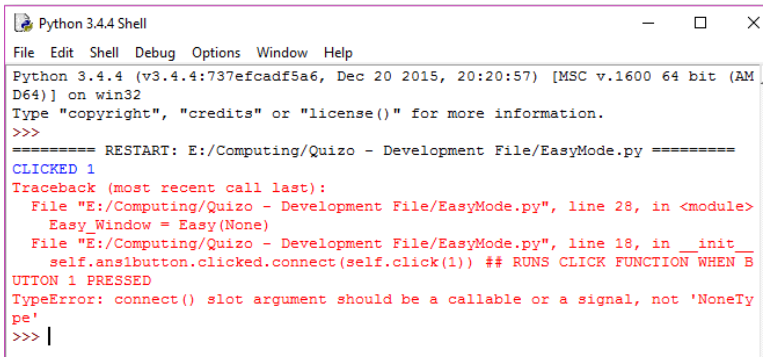
```
def __init__(self,parent=None):
    QtGui.QMainWindow.__init__(self,parent)
    self.setupUi(self)
    self.ans1button.clicked.connect(self.click,1)
    self.ans2button.clicked.connect(self.click,2)
    self.ans3button.clicked.connect(self.click,3)
    self.ans4button.clicked.connect(self.click,4)
    self.questionlabel.setText("QUESTION GOES HERE

def click(self,number):
    print("CLICKED",str(number))
```

Trying to pass parameters as above caused the program to crash when trying to press the fourth button.



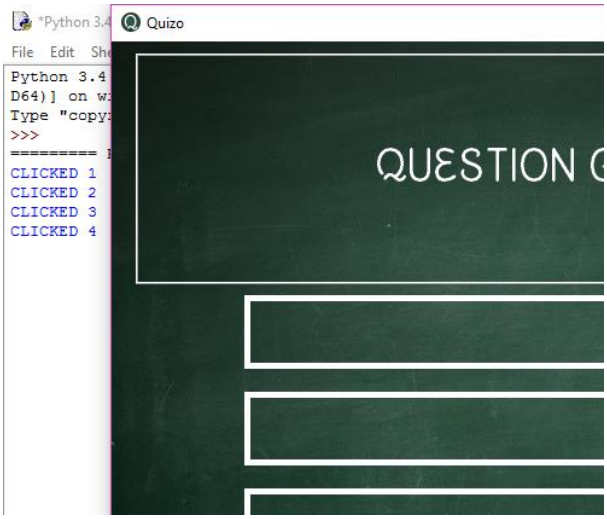
```
self.ans1button.clicked.connect(self.click(1))
self.ans2button.clicked.connect(self.click(2))
self.ans3button.clicked.connect(self.click(3))
self.ans4button.clicked.connect(self.click(4))
self.questionlabel.setText("QUESTION GOES HERE"
```

And then trying to pass parameters like this wouldn't work due to not being a callable or a signal.




```
from functools import partial
    self.ans1button.clicked.connect(partial(self.click,1))
    self.ans2button.clicked.connect(partial(self.click,2))
    self.ans3button.clicked.connect(partial(self.click,3))
    self.ans4button.clicked.connect(partial(self.click,4))
    self.questionlabel.setText("QUESTION GOES HERE") ## SET
```

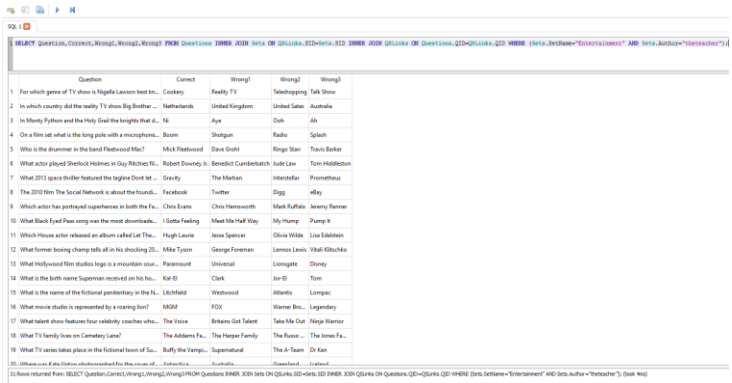
When importing the partial function from the functools module, the program worked as intended, successfully printing the correct number for each button without crashing.



FETCHING QUESTIONS FOR A QUESTION SET

Having created the screen for selecting which set to attempt, I decided that the details that would be fetched when selecting a set would be the setname and author, because although setnames can be used by multiple sets, and one author could make multiple sets, authors will not be allowed to create two of their own sets with the same name, thus creating a unique identifier. Having this knowledge, when the program moves to the multiple choice section, it must fetch all the questions from the given set, which will be achieved with the SQLite statement below, and I have shown an example of it working in the SQLite Browser.

SELECT Question,Correct,Wrong1,Wrong2,Wrong3 FROM Questions INNER JOIN Sets ON QSLinks.SID=Sets.SID INNER JOIN QSLinks ON Questions.QID=QSLinks.QID WHERE (Sets.SetName="'+setname+'" AND Sets.Author="'+author+'");



In order to know the setname and author for the set that the user wants to attempt, I will need to globalise these details from the play game screen when this part of the program is added to the main file. However, for now I will use temporary values for the variables when fetching data, for which the code can be seen below.

```
setname = "Entertainment" ## TEMPORARY
author = "theteacher" ## TEMPORARY

query = 'SELECT Question,Correct,Wrong1,Wrong2,Wrong3 FROM Questions INNER JOIN Sets ON QSLinks.SID=Sets.SID INNER JOIN QSLinks ON Questions.QID=QSLinks.QID WHERE (Sets.SetName="'+setname+'" AND Sets.Author="'+author+'");'
cur.execute(query) ## EXECUTES THE QUERY
questions = cur.fetchall() ## FETCHES ALL THE QUESTIONS FROM THE DATABASE
print(questions)
```

To show that this code fetches the same data as when the statement was executed in the database, I have set the program to print all the questions fetched, and can be seen below.

```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 20:20:57) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: E:/Computing/Quizo - Development File/EasyMode.py ==========
[('For which genre of TV show is Nigella Lawson best known? ', 'Cookery', 'Reality TV', 'Teleshopping', 'Talk Show')
, ('In which country did the reality TV show Big Brother originally air?', 'Netherlands', 'United Kingdom', 'United
States', 'Australia'), ('In Monty Python and the Holy Grail the knights that demanded a shrubbery are known to say wh
at?', 'Ni', 'Aye', 'Ooh', 'Ah'), ('On a film set what is the long pole with a microphone on the end of it called?',
'Boom', 'Shotgun', 'Radio', 'Splash'), ('Who is the drummer in the band Fleetwood Mac?', 'Mick Fleetwood', 'Dave Gro
hl', 'Ringo Starr', 'Travis Barker'), ('What actor played Sherlock Holmes in Guy Ritchies film of the same name?', '
Robert Downey Jr.', 'Benedict Cumberbatch', 'Jude Law', 'Tom Hiddleston'), ('What 2013 space thriller featured the t
agline Dont let go?', 'Gravity', 'The Martian', 'Interstellar', 'Prometheus'), ('The 2010 film The Social Network is
about the founding of what website?', 'Facebook', 'Twitter', 'Digg', 'eBay'), ('Which actor has portrayed superheroe
s in both the Fantastic Four and The Avengers on  the big screen as of 2015?', 'Chris Evans', 'Chris Hemsworth', 'Ma
rk Ruffalo', 'Jeremy Renner'), ('What Black Eyed Peas song was the most downloaded song of all time on iTunes when A
pple announced the list in 2010?', 'I Gotta Feeling', 'Meet Me Half Way', 'My Hump', 'Pump It'), ('Which House actor
released an album called Let Them Talk?', 'Hugh Laurie', 'Jesse Spencer', 'Olivia Wilde', 'Lisa Edelstein'), ('What
former boxing champ tells all in his shocking 2013 autobiography,  Undisputed Truth?', 'Mike Tyson', 'George Foreman
', 'Lennox Lewis', 'Vitali Klitschko'), ('What Hollywood film studios logo is a mountain sourrounded by 22 stars?',
'Paramount', 'Universal', 'Lionsgate', 'Disney'), ('What is the birth name Superman received on his home planet Kryp
ton?', 'Kal-El', 'Clark', 'Jor-El', 'Tom'), ('What is the name of the fictional penitentiary in the Netflix hit seri
es Orange is the New Black?', 'Litchfield', 'Westwood', 'Atlantis', 'Lompac'), ('What movie studio is represented by
a roaring lion?', 'MGM', 'FOX', 'Warner Brothers', 'Legendary'), ('What talent show features four celebrity coaches
who compete to make their proteges the eventual winners?', 'The Voice', 'Britains Got Talent', 'Take Me Out', 'Ninja
Warrior'), ('What TV family lives on Cemetery Lane?', 'The Addams Family', 'The Harper Family', 'The Russo Family',
'The Jones Family'), ('What TV series takes place in the fictional town of Sunnydale,California?', 'Buffy the Vampir
e Slayer', 'Supernatural', 'The A-Team', 'Dr Ken'), ('Where was Kate Upton photographed for the cover of Sport Illus
trateds 2013 Swimsuit edition?', 'Antarctica', 'Australia', 'Greenland', 'Iceland'), ('Which actor plays the lead ro
le in the film Bridget Joness Diary?', 'Renee Zellweger', 'Gemma Jones', 'Shirley Henderson', 'Sally Phillips'), ('W
hich musician received Frances highest cultural award, the Legion of Honor, in 2013?', 'Bob Dylan', 'Ray Charles', '
John Legend', 'Lou Reed'), ('What fictional mineral was sought by the Sky People in Avatar?', 'Unobtainium', 'Adaman
tium', 'Vibranium', 'Platinum'), ('What is Nicolas Cages real name?', 'Nicolas Kim Coppola', 'Nicky Coppola', 'Richa
rd Cage', 'Nicolas Fitz'), ('Which secret agent battled with Nick Nack, Oddjob, and Jaws?', 'James Bond', 'Indiana J
ones', 'Alex Rider', 'Johnny English'), ('Which two Beatles have kids that were born on the same day?', 'Paul and Ri
ngo', 'John and George', 'Paul and John', 'Ringo and George'), ('Which U.S. novelist was the creator of Jurassic Par
k and the TV medical drama series ER?', 'Michael Crichton', 'John Steinbeck', 'George Orwell', 'James Baldwin'), ('W
ho directed Slumdog Millionaire?', 'Danny Boyle', 'Edgar Wright', 'Ben Affleck', 'Ridley Scott'), ('Who has a treadm
ill on the International Space Station named for him?', 'Stephen Colbert', 'Jimmy Kimmel', 'Bill Nye', 'Jon Stewart'
), ('What is the name of Postman Pats black and white cat?', 'Jess', 'Bess', 'Peach', 'Em'), ('Who was the subject o
f Elton Johns original song Candle in the Wind?', 'Marilyn Monroe', 'Marilyn Manson', 'Aretha Franklin', 'Whoppi Gol
dberg')]
>
```

Now that all the data is being fetched correctly, the program needs to be able to take all the questions, randomly select one, and display all the details in the label and buttons. When selecting a question, it needs to make sure that it doesn't select a question that has previously been selected, otherwise the quiz would make the user answer the same question multiple times. In order to prevent this, instead of generating a random integer and then selecting that index from the list of questions like I had previously thought of, I will use the shuffle method in the random module to mix up the order of the questions, and then just iterate through all the questions, therefore also preventing the user from learning the order of the questions. The code for this can be seen below, alongside a test showing that using the shuffle function has in fact changed the order of the questions in the list.

```
query = 'SELECT Question,Correct,Wrong1,Wrong2,Wrong3 FROM Questions INNER JOI
cur.execute(query) ## EXECUTES THE QUERY
questions = cur.fetchall() ## FETCHES ALL THE QUESTIONS FROM THE DATABASE
random.shuffle(questions) ## SHUFFLES THE ORDER OF THE QUESTIONS IN THE LIST
print(questions)
```

```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 20:20:57) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: E:/Computing/Quizo - Development File/EasyMode.py ==========
[('Where was Kate Upton photographed for the cover of Sport Illustrateds 2013 Swimsuit edition?', 'Antarctica', 'Australia', 'Greenland', 'Iceland')
, ('In Monty Python and the Holy Grail the knights that demanded a shrubbery are known to say what?', 'Ni', 'Aye', 'Ooh', 'Ah'), ('What actor played
Sherlock Holmes in Guy Ritchies film of the same name?', 'Robert Downey Jr.', 'Benedict Cumberbatch', 'Jude Law', 'Tom Hiddleston'), ('Which musicia
n received Frances highest cultural award, the Legion of Honor, in 2013?', 'Bob Dylan', 'Ray Charles', 'John Legend', 'Lou Reed'), ('Who directed Sl
umdog Millionaire?', 'Danny Boyle', 'Edgar Wright', 'Ben Affleck', 'Ridley Scott'), ('Which actor has portrayed superheroes in both the Fantastic Fo
ur and The Avengers on  the big screen as of 2015?', 'Chris Evans', 'Chris Hemsworth', 'Mark Ruffalo', 'Jeremy Renner'), ('What talent show features
four celebrity coaches who compete to make their proteges the eventual winners?', 'The Voice', 'Britains Got Talent', 'Take Me Out', 'Ninja Warrior'
), ('What is Nicolas Cages real name?', 'Nicolas Kim Coppola', 'Nicky Coppola', 'Richard Cage', 'Nicolas Fitz'), ('What former boxing champ tells al
l in his shocking 2013 autobiography,  Undisputed Truth?', 'Mike Tyson', 'George Foreman', 'Lennox Lewis', 'Vitali Klitschko'), ('What Black Eyed Pe
as song was the most downloaded song of all time on iTunes when Apple announced the list in 2010?', 'I Gotta Feeling', 'Meet Me Half Way', 'My Hump'
, 'Pump It'), ('What 2013 space thriller featured the tagline Dont let go?', 'Gravity', 'The Martian', 'Interstellar', 'Prometheus'), ('Which U.S. n
ovelist was the creator of Jurassic Park and the TV medical drama series ER?', 'Michael Crichton', 'John Steinbeck', 'George Orwell', 'James Baldwin
'), ('For which genre of TV show is Nigella Lawson best known? ', 'Cookery', 'Reality TV', 'Teleshopping', 'Talk Show'), ('Which actor plays the lea
d role in the film Bridget Joness Diary?', 'Renee Zellweger', 'Gemma Jones', 'Shirley Henderson', 'Sally Phillips'), ('What fictional mineral was so
ught by the Sky People in Avatar?', 'Unobtainium', 'Adamantium', 'Vibranium', 'Platinum'), ('In which country did the reality TV show Big Brother or
iginally air?', 'Netherlands', 'United Kingdom', 'United Sates', 'Australia'), ('Which House actor released an album called Let Them Talk?', 'Hugh L
aurie', 'Jesse Spencer', 'Olivia Wilde', 'Lisa Edelstein'), ('Who was the subject of Elton Johns original song Candle in the Wind?', 'Marilyn Monroe
', 'Marilyn Manson', 'Aretha Franklin', 'Whoppi Goldberg'), ('The 2010 film The Social Network is about the founding of what website?', 'Facebook',
'Twitter', 'Digg', 'eBay'), ('What Hollywood film studios logo is a mountain sourrounded by 22 stars?', 'Paramount', 'Universal', 'Lionsgate', 'Disn
ey'), ('What TV family lives on Cemetery Lane?', 'The Addams Family', 'The Harper Family', 'The Russo Family', 'The Jones Family'), ('Which two Beat
les have kids that were born on the same day?', 'Paul and Ringo', 'John and George', 'Paul and John', 'Ringo and George'), ('Which secret agent batt
led with Nick Nack, Oddjob, and Jaws?', 'James Bond', 'Indiana Jones', 'Alex Rider', 'Johnny English'), ('Who has a treadmill on the International S
pace Station named for him?', 'Stephen Colbert', 'Jimmy Kimmel', 'Bill Nye', 'Jon Stewart'), ('What TV series takes place in the fictional town of S
unnydale,California?', 'Buffy the Vampire Slayer', 'Supernatural', 'The A-Team', 'Dr Ken'), ('What is the name of the fictional penitentiary in the
Netflix hit series Orange is the New Black?', 'Litchfield', 'Westwood', 'Atlantis', 'Lompac'), ('Who is the drummer in the band Fleetwood Mac?', 'Mi
ck Fleetwood', 'Dave Grohl', 'Ringo Starr', 'Travis Barker'), ('What is the name of Postman Pats black and white cat?', 'Jess', 'Bess', 'Peach', 'Em
'), ('What movie studio is represented by a roaring lion?', 'MGM', 'FOX', 'Warner Brothers', 'Legendary'), ('On a film set what is the long pole wit
h a microphone on the end of it called?', 'Boom', 'Shotgun', 'Radio', 'Splash'), ('What is the birth name Superman received on his home planet Krypt
on?', 'Kal-El', 'Clark', 'Jor-El', 'Tom')]
```

OBJECTIVE 55 COMPLETED – PROGRAM FETCHES ALL CORRECT QUESTIONS

USING TIMERS TO CALCULATE POINTS

When the user starts this section of the program, the idea is that they will be immediately given a question, and then once they answer, they will move onto the next question without seeing whether or not they got the first question correct. From my interviews with some of the pupils at the RGS, they said that a score made up of both time and accuracy would be ideal. In order to achieve this, at the start of every question, a timer needs to be started, and then the time taken before they answer the question needs to be added or removed from their total score depending on whether they answered the question correctly or not.

I have decided to import python's time module to achieve this, creating a new timer at the beginning of every question, and then using a formula to convert their elapsed time to a numerical score. After some debating about the formula, I decided that it actually didn't matter what the actual values were in the formula, because everyone will be scored the same way. However from an aesthetic standpoint, I decided that scores should be whole numbers, so I used large multiples to remove any decimals. My method for reaching this conclusion can be seen below.

Initial Test – Implementing the timer – After importing the time module, I set a total for the user and created a new timer when the window was initialised. Whenever the user then pressed any of the buttons I printed their elapsed time to see how the elapsed time was recorded in python.

```
import random,time
```

```
total = 0 ## SETS USER'S TOTAL SCORE TO 0
self.timer = QtCore.QElapsedTimer() ## CREATES A TIMER OBJECT
self.timer.start() ## STARTS THE TIMER


def click(self,number):
    points = self.timer.restart() ## SETS POINTS TO THE ELAPSED TIME
    print(points)
```

Time Taken To Click (Roughly)	Value of Points
1 second	994
1 second	963
2 seconds	1982
6 seconds	6279

From my very rough tests, I found that the elapsed time was recorded as an integer, and was equivalent to the number of milliseconds elapsed since the last press of a button. As my program requires users that take longer to have a smaller magnitude of points, I will need to use a formula such as 1/points to create values that would work with my program.

Using the values I received from my first test, I have created a table showing how they would change if I used my proposed 1/points formula. As can be seen, the values are incredibly small and decimal, which is not what I wanted, so I decided to add another constant to the formula to try and increase the values and make them integers.

Time Taken To Click	Points	1/Points	1/Points X 1000000	Rounded Value
1 second	994	0.001005036217	1005.036217	1005
1 second	963	0.0010384215991	1038.4215991	1038
2 seconds	1982	0.0005045408678	504.5408678	505
6 seconds	6279	0.0001592610288	159.2610288	159

From the table I have found that using a multiplier of one million allows the values to be large enough numbers, but as the values still had some decimals, I decided that I will use a round function to keep the values whole numbers. I have implemented my findings into my code to change the points scored by users when answering questions.

```
def click(self,number):
    points = round(1000000/self.timer.restart()) ## SETS POINTS TO THE ELAPSED TIME
```

OBJECTIVES 60,61, AND 62 COMPLETED – PROGRAM STARTS A TIMER WHEN A QUESTION IS ASKED AND THEN FINDS THE TIME TAKEN FOR THE USER TO ANSWER. THIS TIME IS THEN USED IN A FORMULA TO CALCULATE THE MAGNITUDE OF POINTS EARNED

DISPLAYING QUESTIONS TO THE USER

In order for the user to actually answer questions, they need to be given all the necessary information: the question, and the possible answers. The way that my database is formatted, it is very easy to display the actual question to the user, simply using a setText command for the label as can be seen below.

```
global questions
```

Globalises the questions list inside the initialise function

```
total = 0 ## SETS USER'S TOTAL SCORE TO 0
self.timer = QtCore.QElapsedTimer() ## CREATES A
self.timer.start() ## STARTS THE TIMER
self.ask() ## DISPLAYS THE QUESTION TO THE USER
```

After the window is initialised and the timer is created, the ask function is started

```
def ask(self):
    if len(questions) != 0: ## CHECKS THAT THERE ARE STILL QUESTIONS LEFT TO ASK
        current = questions[0] ## SETS THE CURRENT QUESTION
        self.questionlabel.setText(current[0]) ## DISPLAYS THE QUESTION TO THE USER
        questions.remove(current) ## REMOVES THE CURRENT QUESTION FROM THE POOL
    else:
        print("ALL QUESTIONS ANSWERED")
```

This is the new function where the code for displaying the questions will be written. Currently the list of questions is checked to see if there are any questions left, and if so it asks the next question, and then removes it from the list

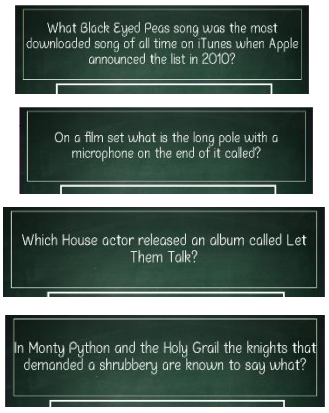
```
def click(self,number):
    points = round(1000000/self.timer.restart()) ## SETS POINTS TO THE E
    print(points)
    self.ask() ## AFTER USER CHOOSES AN ANSWER, NEXT QUESTION IS ASKED
```

Whenever a button is clicked, the points are calculated and then the next question is asked

Below are some screenshots showing this code working, going through each question and then printing that there are no more questions left to answer.

This question set has 31 questions, and as shown to the right, 31 scores were recorded before all questions had been answered as expected. There are also screenshots of a few of the questions that were displayed.

```
75
91
77
3322
6410
7143
6452
3906
7194
3472
5917
6757
6803
6757
6211
6452
6452
5882
7143
6803
7194
6173
6803
5814
6211
6410
5405
6098
5376
5848
5917
ALL QUESTIONS ANSWERED
```



OBJECTIVES 56 AND 57 COMPLETED – PROGRAM GOES THROUGH EACH QUESTION IN THE QUESTION SET, DISPLAYING THE QUESTION IN A TEXT BOX

Now that the questions are being correctly displayed, the possible answers now also need to be displayed. For the program to work, each question has four possible answers in the database; one correct answer and three wrong ones. The correct answer is always stored before the three wrong ones, and if I displayed all the options without swapping the positions, users would likely catch on and just always pick the first answer. Due to this, the position of the answers must be randomised before being placed on the buttons. To do this, I will create a list with the four buttons [1,2,3,4] and then shuffle their order. I will then go through each button and assign them an answer, and then when the user presses a button, the program will check to see if their answer is the correct one. This can be better seen in the code on the next page.

As a side note I have also realised that the code I have written in the initialise method should really be in the showEvent method because when I move this code over to the main file, this screen will not be the first thing the user sees, so I have amended that too.

```
def showEvent(self,parent=None):
    global questions
    setname = "Entertainment" ## TEMPORARY
    author = "theteacher" ## TEMPORARY

    query = 'SELECT Question,Correct,Wrong1,Wrong2,Wrong3 FROM Questions INNER JOIN Sets ON QSLinks.SID=Sets
    cur.execute(query) ## EXECUTES THE QUERY
    questions = cur.fetchall() ## FETCHES ALL THE QUESTIONS FROM THE DATABASE
    random.shuffle(questions) ## SHUFFLES THE ORDER OF THE QUESTIONS IN THE LIST

    total = 0 ## SETS USER'S TOTAL SCORE TO 0
    self.timer = QtCore.QElapsedTimer() ## CREATES A TIMER OBJECT
    self.timer.start() ## STARTS THE TIMER
    self.ask() ## DISPLAYS THE QUESTION TO THE USER

def ask(self):
    global choices,current
    if len(questions) != 0: ## CHECKS THAT THERE ARE STILL QUESTIONS LEFT TO ASK
        current = questions[0] ## SETS THE CURRENT QUESTION
        self.questionlabel.setText(current[0]) ## DISPLAYS THE QUESTION TO THE USER
        questions.remove(current) ## REMOVES THE CURRENT QUESTION FROM THE POOL
        choices = [1,2,3,4] ## THE POSSIBLE BUTTONS THAT CAN BE PRESSED
        random.shuffle(choices) ## SHUFFLES THE ORDER OF THE BUTTONS
        self.ans1button.setText(current[choices[0]]) ## SETS THE TEXT OF BUTTON 1
        self.ans2button.setText(current[choices[1]]) ## SETS THE TEXT OF BUTTON 2
        self.ans3button.setText(current[choices[2]]) ## SETS THE TEXT OF BUTTON 3
        self.ans4button.setText(current[choices[3]]) ## SETS THE TEXT OF BUTTON 4
    else:
        print("ALL QUESTIONS ANSWERED")

def click(self,number):
    correct = int(choices.index(1)) ## FINDS THE BUTTON NUMBER WITH THE CORRECT ANSWER
    print("YOU CHOSE ",current[choices[number]]) ## PRINTS THE USER'S ANSWER
    if number == correct: ## CHECKS IF THE BUTTON PRESSED IS THE SAME AS THE BUTTON WITH THE CORRECT ANSWER
        print("CORRECT")
    else:
        print("INCORRECT, THE CORRECT ANSWER IS",current[1]) ## PRINTS THE CORRECT ANSWER
    points = round(1000000/self.timer.restart()) ## SETS POINTS TO THE ELAPSED TIME
    print(points)
    self.ask() ## AFTER USER CHOOSES AN ANSWER, NEXT QUESTION IS ASKED
```

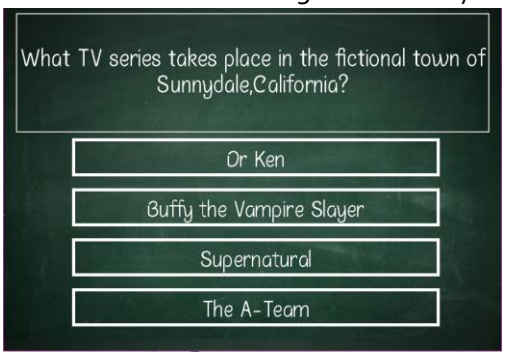
To test that this method of randomizing the order of answers is working, I have added code for the program to print the answer that the user chooses, aswell as the correct answer if they choose incorrectly. This code is purely for testing purposes and I will remove it when I am certain the program is working correctly.

I needed to globalise the choices and current lists so that they could also be used in the click function to check whether the user answered correctly.

To test that this code is working I will attempt the first four questions that I am given, and I will deliberately get two correct, and two incorrect to check that the program is correctly recognising which answer I pick, and whether or not it is correct. I will be able to do this deliberately because I will be using the database to check what the correct answers are for the questions I am given.

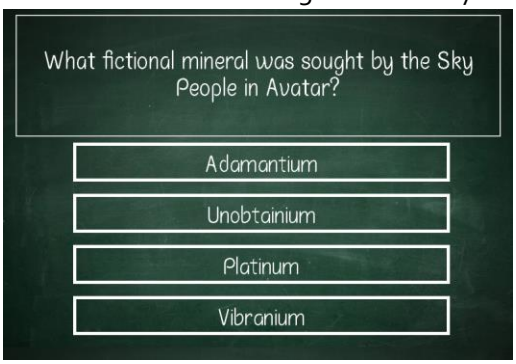
Test Number	Question	Correct Answer	My Answer
1	What TV series takes place in the fictional town of Sunnydale, California?	Buffy the Vampire Slayer	Buffy the Vampire Slayer
2	What fictional mineral was sought by the Sky People in Avatar?	Unobtainium	Unobtainium
3	What Black Eyed Peas song was the most downloaded song of all time on iTunes when Apple annouced the list in 2010?	I Gotta Feeling	Meet Me Half Way
4	Which secret agent battled with Nick Nack, Oddjob, and Jaws?	James Bond	Johnny English

Test 1 – Successful – Program correctly identified my answer as correct




```
YOU CHOSE  Buffy the Vampire Slayer
CORRECT
```

Test 2 – Successful – Program correctly identified my answer as correct

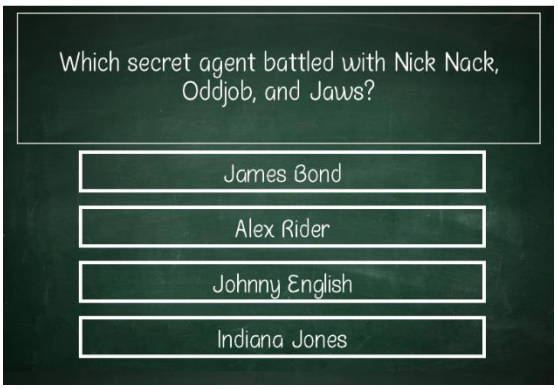



```
YOU CHOSE  Unobtainium
CORRECT
```

Test 3 – Successful – Program correctly identified my answer as incorrect, and showed the correct answer


```
YOU CHOSE  Meet Me Half Way
INCORRECT, THE CORRECT ANSWER IS I Gotta Feeling
```

Test 4 – Successful – Program correctly identified my answer as incorrect , and showed the correct answer




```
YOU CHOSE  Johnny English
INCORRECT, THE CORRECT ANSWER IS James Bond
```

All four tests were successful in identifying which button I had pressed and whether my answer was correct or not, and so I can add the final lines of code to this window of the program.

OBJECTIVE 59 COMPLETED – ANSWERS ARE DISPLAYED RANDOMLY ON BUTTONS

CALCULATING THE TOTAL

In order for this window to be fully complete, it needs to add the user's points if they get the answer correct, and if they get the answer incorrect, their points need to be deducted from their total, and the questions needs to be added to a list of incorrect questions, so that the user can review how they did at the end of the quiz.

I have added these additions to the program below, creating a new list called incorrect to store all the incorrect questions, along with the user's answers to them, which I had to globalise so that it could be used both in the click function, and also in the results window that I will create later in my development.

```
def showEvent(self,parent=None):
    global questions,incorrect,total
    setname = "Entertainment" ## TEMPORARY
    author = "theteacher" ## TEMPORARY

    query = 'SELECT Question,Correct,Wrong1,Wrong2,Wrong3 FROM Que
    cur.execute(query) ## EXECUTES THE QUERY
    questions = cur.fetchall() ## FETCHES ALL THE QUESTIONS FROM T
    random.shuffle(questions) ## SHUFFLES THE ORDER OF THE QUESTIC

    incorrect = [] ## RESETS THE LIST OF USER'S INCORRECT ANSWERS
    total = 0 ## SETS USER'S TOTAL SCORE TO 0
def click(self,number):
    correct = int(choices.index(1)) ## FINDS THE BUTTON NUMBER WITH THE COR
    points = round(1000000/self.timer.restart()) ## SETS POINTS TO THE ELAP
    if number == correct: ## CHECKS IF THE BUTTON PRESSED IS THE SAME AS TH
        total += points ## ADDS POINTS TO TOTAL
    else:
        total -= points ## DEDUCTS POINTS FROM TOTAL
        incorrect.append([current[0],current[choices[number]],current[1]])
    self.ask() ## AFTER USER CHOOSES AN ANSWER, NEXT QUESTION IS ASKED
```

When trying to test this new code, I encountered an issue with the "total" variable, which occurred when trying to add or subtract points to or from the total.

```
========= RESTART: E:/Computing/Quizo - Development File/EasyMode.py =========
Traceback (most recent call last):
  File "E:/Computing/Quizo - Development File/EasyMode.py", line 65, in click
    total -= points ## DEDUCTS POINTS FROM TOTAL
UnboundLocalError: local variable 'total' referenced before assignment
```

The issue stated that the local variable total was referenced before assignment, which seemed strange because I had globalised the variable in the showEvent function. To check whether this was an issue with just the total, I commented out the lines of code regarding the total to see if the program would have an issue with the new incorrect list that I had also globalised. Even though both variables were globalised and created in the same function, showEvent, and were both being used by the same function, click, the incorrect variable still managed to work.

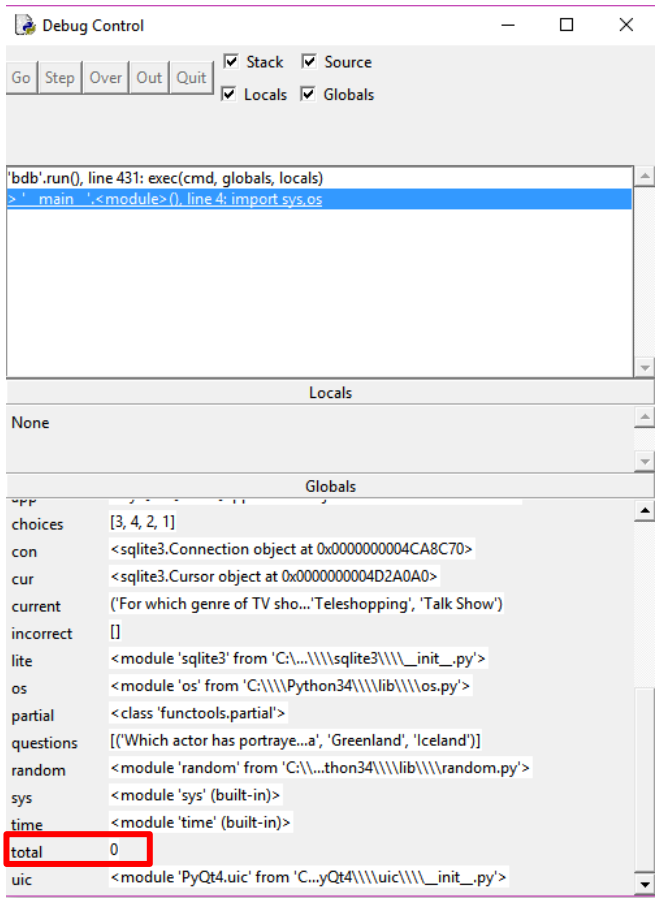
```
ALL QUESTIONS ANSWERED
[['In Monty Python and the Holy Grail the knights that demanded a shrubbery are known to say what?', 'Ah', 'Ni'], ['What TV family liv
es on Cemetery Lane?', 'The Russo Family', 'The Addams Family'], ['What talent show features four celebrity coaches who compete to mak
e their proteges the eventual winners?', 'Ninja Warrior', 'The Voice'], ['What Hollywood film studios logo is a mountain sourrounded b
y 22 stars?', 'Lionsgate', 'Paramount'], ['What former boxing champ tells all in his shocking 2013 autobiography,  Undisputed Truth?',
'Vitali Klitschko', 'Mike Tyson'], ['Which musician received Frances highest cultural award, the Legion of Honor, in 2013?', 'John Leg
end', 'Bob Dylan'], ['Who is the drummer in the band Fleetwood Mac?', 'Travis Barker', 'Mick Fleetwood'], ['In which country did the r
eality TV show Big Brother originally air?', 'United Kingdom', 'Netherlands'], ['What is the name of the fictional penitentiary in the
Netflix hit series Orange is the New Black?', 'Lompac', 'Litchfield'], ['Which U.S. novelist was the creator of Jurassic Park and the
TV medical drama series ER?', 'John Steinbeck', 'Michael Crichton'], ['What is the birth name Superman received on his home planet Kry
pton?', 'Jor-El', 'Kal-El'], ['Which actor has portrayed superheroes in both the Fantastic Four and The Avengers on  the big screen as
of 2015?', 'Jeremy Renner', 'Chris Evans'], ['What movie studio is represented by a roaring lion?', 'FOX', 'MGM'], ['Which actor plays
the lead role in the film Bridget Joness Diary?', 'Shirley Henderson', 'Renee Zellweger'], ['For which genre of TV show is Nigella Law
son best known? ', 'Teleshopping', 'Cookery'], ['Who directed Slumdog Millionaire?', 'Ridley Scott', 'Danny Boyle'], ['Which two Beatl
es have kids that were born on the same day?', 'Paul and John', 'Paul and Ringo'], ['Who has a treadmill on the International Space St
ation named for him?', 'Jimmy Kimmel', 'Stephen Colbert'], ['What is the name of Postman Pats black and white cat?', 'Em', 'Jess'], ['
What is Nicolas Cages real name?', 'Nicky Coppola', 'Nicolas Kim Coppola'], ['What Black Eyed Peas song was the most downloaded song o
f all time on iTunes when Apple announced the list in 2010?', 'Meet Me Half Way', 'I Gotta Feeling'], ['Which House actor released an
album called Let Them Talk?', 'Jesse Spencer', 'Hugh Laurie'], ['Which secret agent battled with Nick Nack, Oddjob, and Jaws?', 'Alex
Rider', 'James Bond']]
```

Incorrect variable was successfully printed to the IDLE when all questions had been answered.

I looked through my code multiple times, however I could not find where the issue was coming from.

I tried using python's built-in debugger to see if it was actually globalising the total variable and I found that it was, so that wasn't the issue.

Having also looked online for solutions and finding nothing that could help, I started just going through trying as many different ideas as possible. I tried creating the variable outside of the function, then outside of the class, and then I tried renaming the variable to see if that was the problem, but all of these ideas did nothing to help fix the problem.



Eventually, I tried declaring the total as a global variable inside both functions, showEvent and click, to see if that would work, and luckily that managed to fix the problem. The program was no longer producing errors and the total was being used by all three functions even though it was only globalised in two.

```
def click(self,number):
    global total
```

I am unsure of how or why what I did fixed the problem, but the fact the it fixed it was enough for me to be satisfied.

As I said I did some testing to check that it was working and this testing can be seen below. I went through the entire quiz twice, first getting every question correct, and then getting every question incorrect. I did this to make sure that the program was correctly adding and subtracting points when it needed to. I also added statements to print the number of points that were being added or subtracted for each question as a secondary check.

```
+ 244
+ 274
+ 671
+ 501
+ 521
+ 546
+ 491
+ 631
+ 682
+ 735
+ 858
+ 791
+ 766
+ 854
+ 799
+ 724
+ 428
+ 309
+ 682
+ 543
+ 735
+ 513
+ 482
+ 188
+ 562
+ 631
+ 870
+ 791
+ 954
+ 541
+ 602
ALL QUESTIONS ANSWERED
[]
18919
```

Test 1 – All Correct Answers – Successful – After getting every question correct, my score remained positive, and after adding all the individual scores up for each question, it did equal to the total that the program printed. Additionally, when the program printed the incorrect list, it was empty as expected.

Test 2 – All Incorrect Answers – Successful – After getting every question incorrect, my score remained negative, and after adding all the individual scores up for each question, it did equal to the total that the program printed. Additionally, when the program printed the incorrect list, it had every question in it, as expected.

```
- 600
- 397
- 960
- 864
- 795
- 887
- 901
- 946
- 607
- 864
- 816
- 893
- 417
- 961
- 634
- 777
- 887
- 758
- 511
- 517
- 560
- 731
- 511
- 692
- 864
- 588
- 543
- 864
- 499
- 809
- 825
ALL QUESTIONS ANSWERED
[['Which House actor released an album called Let Them Talk?', 'Je
ssic Park and the TV medical drama series ER?', 'George Orwell', '
e planet Krypton?', 'Tom', 'Kal-El'], ['What Black Eyed Peas song
cking 2013 autobiography,  Undisputed Truth?', 'Vitali Klitschko',
ches who compete to make their proteges the eventual winners?', 'B
the reality TV show Big Brother originally air?', 'United Kingdom'
'Teleshopping', 'Cookery'], ['Who is the drummer in the band Fleet
TV family lives on Cemetery Lane?', 'The Russo Family', 'The Addam
has a treadmill on the International Space Station named for him?'
ndle in the Wind?', 'Aretha Franklin', 'Marilyn Monroe'], ['On a f
'Universal', 'Paramount'], ['Who directed Slumdog Millionaire?', '
the fictional penitentiary in the Netflix hit series Orange is the
V series takes place in the fictional town of Sunnydale,California
d superheroes in both the Fantastic Four and The Avengers on  the
-22478
```

OBJECTIVES 63 AND 64 COMPLETED – POINTS ARE ADDED TO THE USER'S TOTAL IF THEY GET A QUESTION CORRECT, AND DEDUCTED FROM THE TOTAL IF THEY GET A QUESTION INCORRECT

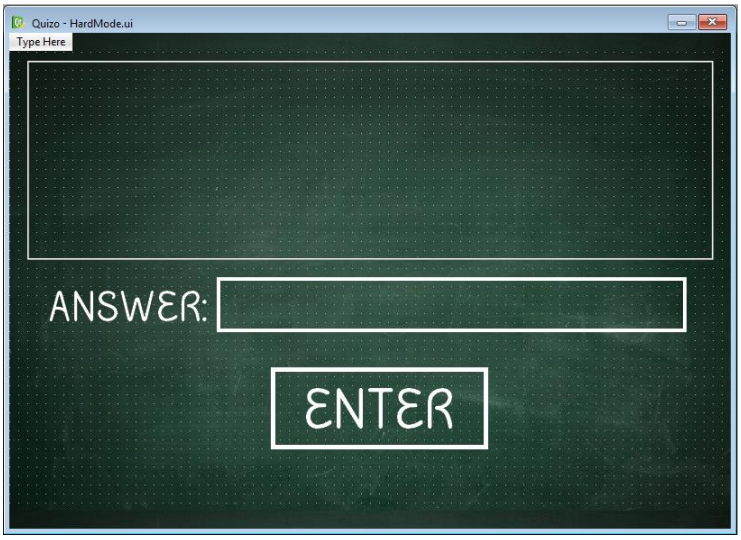
Until I create the results window for the program, I am unable to finish the final objective for this window, so now having created this window, I will move onto creating the "hard" mode.

As a side note, having looked back on this section I have realised that I missed out a crucial line of code for when I move this whole window to the main file. When the program was creating the timer to record the time taken for users to answer, it continues to run until it is closed. When the user finishes answering all the questions and moves onto the results window, if the program doesn't explicitly state to invalidate the timer, it will continue to run until the program is closed, which is going to slow down performance of the program. Due to this, I have gone back and added a line of code to invalidate the timer when the user finishes answering questions.

```
else:
    print("ALL QUESTIONS ANSWERED")
    print(incorrect)
    print(total)
    self.timer.invalidate() ## STOPS THE TIMER
```

CREATING THE NON-MULTIPLE CHOICE SCREEN

For users that find multiple choice too easy, or just like a challenge, there is also a mode in this program that lets you attempt quizzes by manually typing out the answers. Although a more difficult task when answering questions, from a programming stand-point, this window should be far easier to program, as I will only need to check whether what a user has entered is write or wrong, as opposed to multiple choice where I had to do all the shuffling.



As with all the previous screens, I have used my initial design for this window to create the actual window that will be used by the program in QtDesigner which can be seen to the right. Like the multiple choice screen, I will be programming this initially in its own file simply for ease, and then once it is complete, I will put both this and the multiple choice window into the main file together.

Since the code for the non-multiple choice screen is almost identical to the multiple choice screen, I have copy and pasted a lot of the code over as can be seen below, and I will explain the differences that I have made for this screen.

```
## Hard Mode Testing ##

import sys,os
import sqlite3 as lite
import random,time
from PyQt4 import QtCore,QtGui,uic

con = lite.connect('QuizoBase') ## CONNECTING TO THE DATABASE
cur = con.cursor()

Hard_class = uic.loadUiType("HardMode.ui")[0] ## LOADS UI FILE

class Hard(QtGui.QMainWindow,Hard_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        QtGui.QMainWindow.showEvent(self,parent)
        self.setupUi(self)
        self.enterbutton.clicked.connect(self.click)

    def showEvent(self,parent=None):
        global questions,incorrect,total
        setname = "Entertainment" ## TEMPORARY
        author = "theteacher" ## TEMPORARY

        query = 'SELECT Question,Correct,Wrong1,Wrong2,Wrong3 FROM Questions INNER JOIN Sets ON QSLinks.SID=Sets.SID INNER JOIN QSLinks ON Questions.QID=QSL
        cur.execute(query) ## EXECUTES THE QUERY
        questions = cur.fetchall() ## FETCHES ALL THE QUESTIONS FROM THE DATABASE
        random.shuffle(questions) ## SHUFFLES THE ORDER OF THE QUESTIONS IN THE LIST

        incorrect = [] ## RESETS THE LIST OF USER'S INCORRECT ANSWERS
        total = 0 ## SETS USER'S TOTAL SCORE TO 0
        self.timer = QtCore.QElapsedTimer() ## CREATES A TIMER OBJECT
        self.timer.start() ## STARTS THE TIMER
        self.ask() ## DISPLAYS THE QUESTION TO THE USER

    def ask(self):
        if len(questions)!=0: ## CHECKS IF THERE ARE ANY MORE QUESTIONS LEFT
            self.questionlabel.setText(questions[0][0]) ## DISPLAYS THE CURRENT QUESTION IN THE LABEL
        else:
            print("ALL QUESTIONS ANSWERED")
            print(incorrect)
            print(total)
            self.timer.invalidate() ## STOPS THE TIMER


    def click(self):
        global total
        answer = self.ansedit.text() ## GETS USER'S ANSWER FROM TEXT BOX
        correct = questions[0][1] ## CREATES VARIABLE TO STORE THE CORRECT ANSWER
        points = round(1000000/self.timer.restart()) ## SETS POINTS TO THE ELAPSED TIME
        if answer.lower() == correct.lower(): ## CHECKS IF THE USER'S ATTEMPT IS THE SAME AS THE ANSWER - NOT CASE SENSITIVE
            total += points ## ADDS POINTS TO TOTAL
        else:
            total -= points ## DEDUCTS POINTS FROM TOTAL
            incorrect.append([questions[0][0],answer,questions[0][1]]) ## ADDS QUESTION TO INCORRECT LIST, ALONG WITH USER'S ATTEMPT AND THE CORRECT ANSWER
        self.ansedit.clear() ## CLEARS THE TEXT BOX
        questions.pop(0) ## POPS THE FIRST QUESTION FROM THE QUESTION LIST
        self.ask() ## AFTER USER CHOOSES AN ANSWER, NEXT QUESTION IS ASKED

app = QtGui.QApplication(sys.argv)
Hard_Window = Hard(None)
Hard_Window.show() ## SHOWS THE NON-MULTIPLE CHOICE WINDOW
app.exec_()
```

Difference 1 – Instead of having to set the labels for all the buttons, only the question needs to be displayed to the user.

```
def ask(self):
    if len(questions)!=0: ## CHECKS IF THERE ARE ANY MORE QUESTIONS LEFT
        self.questionlabel.setText(questions[0][0]) ## DISPLAYS THE CURRENT QUESTION IN THE LABEL
```

Difference 2 – The only difference here is that it is the "HardMode" ui file that is being loaded and utilised for this screen. This is also the case with creating the window.

```
Hard_class = uic.loadUiType("HardMode.ui")[0] ## LOADS UI FILE

class Hard(QtGui.QMainWindow,Hard_class):

Hard_Window = Hard(None)
Hard_Window.show() ## SHOWS THE NON-MULTIPLE CHOICE WINDOW
```

Difference 3 – The biggest difference between the two windows is that with this window, the questions are removed from the question list when a question is answered rather than when a question is asked. This part makes no difference to the user experience, however does reduce the number of global variables used, which helps reduce memory slow downs. After testing if this works just as well as with the global variables, I will return to the multiple choice screen and change it to using this method of removing questions.

Also in this function, instead of having to check for the index of the button pressed, the program just takes the user's entry as a string and checks it against the correct answer. I have also decided that the program will not be case sensitive when it comes to answering as it does not have any impact on whether a student has the knowledge or not.

```
def click(self):
    global total
    answer = self.ansedit.text() ## GETS USER'S ANSWER FROM TEXT BOX
    correct = questions[0][1] ## CREATES VARIABLE TO STORE THE CORRECT A
    points = round(1000000/self.timer.restart()) ## SETS POINTS TO THE F
    if answer.lower() == correct.lower(): ## CHECKS IF THE USER'S ATTEMI
        total += points ## ADDS POINTS TO TOTAL
    else:
        total -= points ## DEDUCTS POINTS FROM TOTAL
        incorrect.append([questions[0][0],answer,questions[0][1]]) ## AI
    self.ansedit.clear() ## CLEARS THE TEXT BOX
    questions.pop(0) ## POPS THE FIRST QUESTION FROM THE QUESTION LIST
    self.ask() ## AFTER USER CHOOSES AN ANSWER, NEXT QUESTION IS ASKED
```

Now that I have got the code for this screen, I will test it in the same way I tested the multiple choice screen, first by answering all answers correctly, and then a second test with incorrect answers. I will also be printing the individual scores for each question so I can check the program is adding them up correctly.

```
+ 254
+ 146
+ 233
+ 312
+ 119
+ 177
+ 340
+ 202
+ 389
+ 271
+ 184
+ 191
+ 145
+ 260
+ 260
+ 193
+ 155
+ 355
+ 29
+ 211
+ 57
+ 252
+ 178
+ 334
+ 47
+ 148
+ 247
+ 397
+ 245
+ 53
+ 228
ALL QUESTIONS ANSWERED
[]
6612
```

Test 1 – All Correct Answers – Successful – After getting every question correct, my score remained positive, and after adding all the individual scores up for each question, it did equal to the total that the program printed. Additionally, when the program printed the incorrect list, it was empty as expected.

Test 2 – All Incorrect Answers – Successful – After getting every question incorrect, my score remained negative, and after adding all the individual scores up for each question, it did equal to the total that the program printed. Additionally, when the program printed the incorrect list, it had every question in it, as expected.

```
- 332
- 577
- 319
- 424
- 2075
- 398
- 484
- 358
- 610
- 860
- 815
- 429
- 288
- 405
- 870
- 528
- 631
- 343
- 199
- 246
- 899
- 754
- 343
- 147
- 439
- 97
- 409
- 447
- 847
- 585
- 2110
ALL QUESTIONS ANSWERED
[['What actor played Sh
ong was the most downlo
o?', '', 'Gravity'], ['W
', 'Mick Fleetwood'], [
gent battled with Nick
e Addams Family'], ['Wha
'What former boxing cha
ame?', '', 'Nicolas Kim
y air?', '', 'Netherlan
show is Nigella Lawson
the International Space
t demanded a shrubbery
the New Black?', '', 'L
-18268
```

Seeing as how both tests were successful, I can say that the method I used in this window for removing questions from the pool was just as good, and more efficient than the method used in the multiple choice window, and for this reason, I have gone back and changed the multiple choice window's method to this one. The edited code can be seen below.

```
def ask(self):
    global choices
    if len(questions) != 0: ## CHECKS THAT THERE ARE STILL QUESTIONS LEFT TO ASK
        current = questions[0] ## SETS THE CURRENT QUESTION
        self.questionlabel.setText(current[0]) ## DISPLAYS THE QUESTION TO THE USER
        choices = [1,2,3,4] ## THE POSSIBLE BUTTONS THAT CAN BE PRESSED
        random.shuffle(choices) ## SHUFFLES THE ORDER OF THE BUTTONS
        self.ans1button.setText(current[choices[0]]) ## SETS THE TEXT OF BUTTON 1
        self.ans2button.setText(current[choices[1]]) ## SETS THE TEXT OF BUTTON 2
        self.ans3button.setText(current[choices[2]]) ## SETS THE TEXT OF BUTTON 3
        self.ans4button.setText(current[choices[3]]) ## SETS THE TEXT OF BUTTON 4
    else:
        print("ALL QUESTIONS ANSWERED")
        print(incorrect)
        print(total)
        self.timer.invalidate() ## STOPS THE TIMER

def click(self,number):
    global total
    correct = int(choices.index(1)) ## FINDS THE BUTTON NUMBER WITH THE CORRECT ANSWER
    points = round(1000000/self.timer.restart()) ## SETS POINTS TO THE ELAPSED TIME
    if number == correct: ## CHECKS IF THE BUTTON PRESSED IS THE SAME AS THE BUTTON WITH THE CORRECT ANSWER
        total += points ## ADDS POINTS TO TOTAL
    else:
        total -= points ## DEDUCTS POINTS FROM TOTAL
        incorrect.append([questions[0][0],questions[0][choices[number]],questions[0][1]]) ## ADDS QUESTION TO INCORRECT LIST, ALONG WITH THE USER'S ATTEMPT AND THE ACTUAL ANSWER
    questions.pop(0) ## REMOVES THE FIRST QUESTION FROM THE QUESTION LIST
    self.ask() ## AFTER USER CHOOSES AN ANSWER, NEXT QUESTION IS ASKED
```

One thing I noticed when going through this test was how long it took to type an answer, and then press the enter button, and then click back on the text box to type another answer. After the fifth or so question it became laborious and long winded, and with a scoring system that takes speed into account, it didn't feel like it worked well enough for users that want to get the high scores. For this reason, I have added functionality to this part of the program to be able to just press the enter button on a keyboard to enter an answer. The code for this can be seen below, alongside a test showing that it works flawlessly.

```
self.ansedit.returnPressed.connect(self.click) ## RUNS THE CLICK FUNCTION WHEN ENTER IS PRESSED
```

Seeing as how well the enter button was working for entering data, I have decided to go back and add this functionality to the login screen, so that the user can login without having to use their mouse, making the whole program more intuitive and easy to use.

```
self.passedit.returnPressed.connect(self.Login) ## RUNS THE CLICK FUNCTION WHEN ENTER IS PRESSED
```

OBJECTIVE 58 COMPLETED – USERS CAN TYPE IN THEIR ANSWERS, AND THEN MOVE TO THE NEXT QUESTION BY EITHER USING THE BUILT IN ENTER BUTTON OR BY PRESSING THE ENTER KEY ON THEIR KEYBOARD

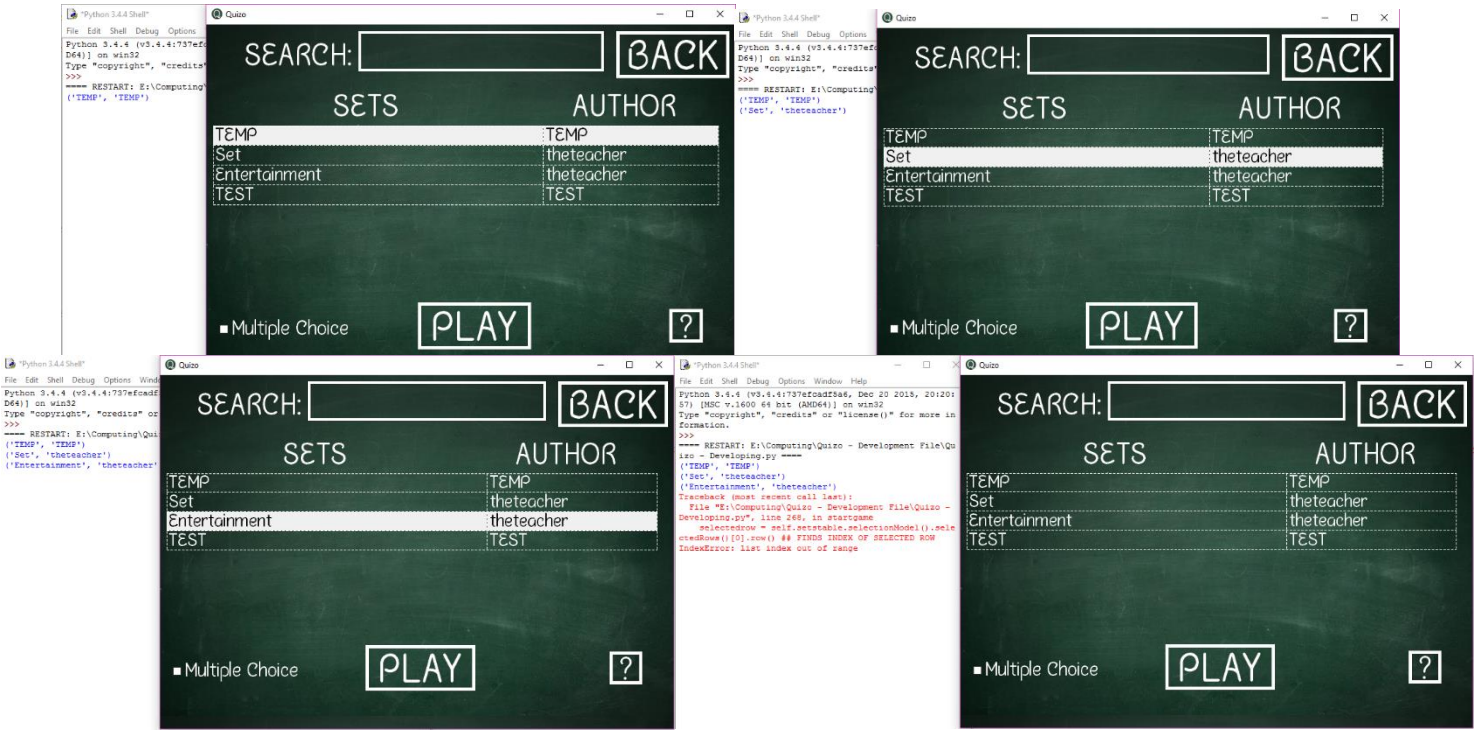
Now that both the multiple choice, and non-multiple choice modes are feature complete, I can go back to the play game screen and add the ability to select a question set and then attempt the quiz.

LINKING THE SET SELECTION AND QUIZ WINDOWS

In the Play Game screen, the user has the ability to check a box to decide whether they want to attempt the quiz as multiple choice or not. Having now created the windows for multiple choice, and non-multiple choice, I can link the play button to start the test when pressed. In order to make this work, I need to be able to check which set the user has selected from the table. Having done some research, I have written code below to print the details on a selected question set.

```
def startgame(self):
    selectedrow = self.setstable.selectionModel().selectedRows()[0].row() ## FINDS INDEX OF SELECTED ROW
    playset = allsets[selectedrow] ## SETS THE CURRENT PLAYSET TO THE SELECTED SET
    print(playset)
```

To test this is working I have selected several rows to check that the correct details are being collected, and I also tested pressing the play button when there is no row selected. When pressing the button without a row selected, an error occurs because there is no selected row to get an index for.

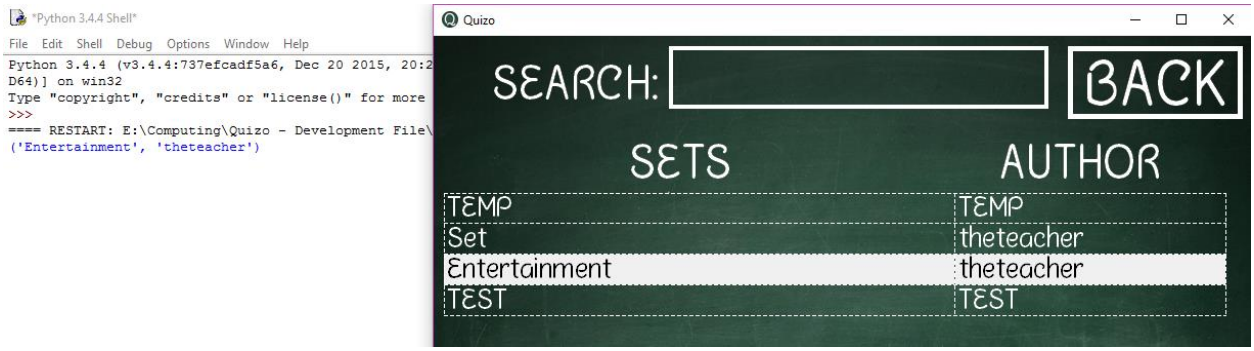


To fix this, I have added an if statement that checks whether the selection is empty, and if so, the program will provide a message box explaining that a set must be selected first.

```
def startgame(self):
    selection = self.setstable.selectionModel().selectedRows() ## FETCHES THE SELECTED ROWS FROM THE TABLE
    if selection == []: ## CHECKS IF ANY ROW HAS BEEN SELECTED
        CreateOutbox("Selection Not Found","You must select a question set to play!","",QtGui.QMessageBox.Critical)
    else:
        selectedrow = selection[0].row() ## FINDS INDEX OF SELECTED ROW
        playset = allsets[selectedrow] ## SETS THE CURRENT PLAYSET TO THE SELECTED SET
        print(playset)
```

Test 1 – Not selecting a question set – Successful – The correct error message was displayed when no set was selected

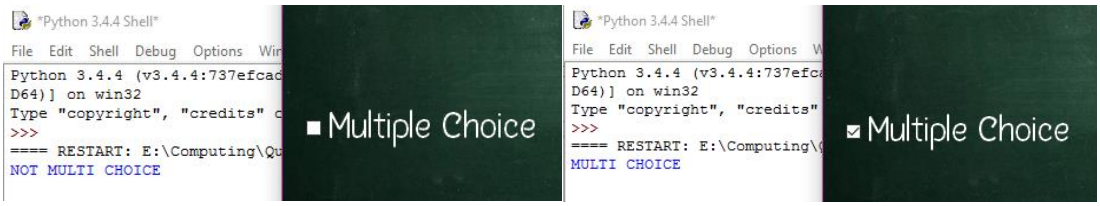
Test 2 – Selecting the "Entertainment" question set – Successful – No error message was shown, and the correct title and author were printed to the IDLE



Now that the program can successfully fetch the details for a set that the user wishes to attempt, I need to add the ability to check whether or not the user wants to play in multiple choice mode or not. In the window is a check box for multiple choice, which when clicked will enable the mode. Instead of changing variables whenever the box is checked and unchecked, I will write the code just to check the state of the box when the user attempts to start the quiz. The code for this can be seen below.

```
else:
    selectedrow = selection[0].row() ## FINDS INDEX OF SELECTED ROW
    playset = allsets[selectedrow] ## SETS THE CURRENT PLAYSET TO THE SELECTED SET
    if self.multichoice.isChecked() == True: ## CHECKS IF THE MULTIPLE CHOICE OPTION IS CHECKED
        print("MULTI CHOICE")

    else:
        print("NOT MULTI CHOICE")
```

To test that this code successfully checks if the user wants to use multiple choice, I have gone into the program and tried starting a quiz while in both states, and the results can be seen below.



As this test has worked successfully, I have now copied the multiple choice and non-multiple choice classes from their respective files to the main program so that I can try and link them to the quiz selection screen.

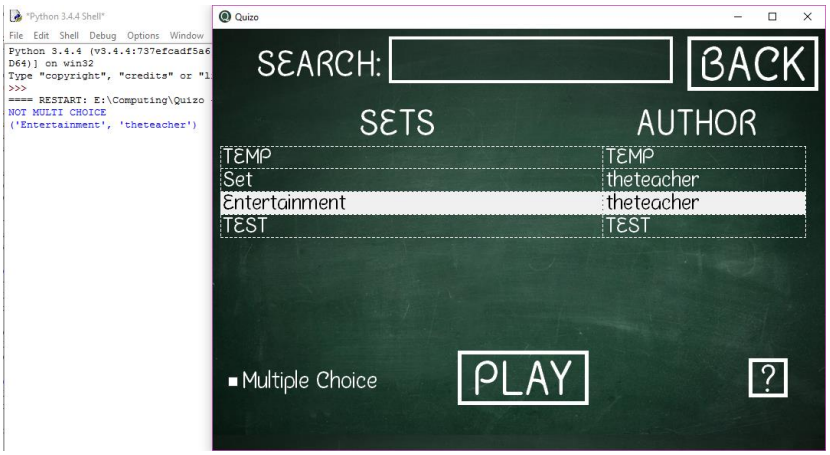
A small issue I've now come across having added the functionality for checking for multiple choice and the selected question set, is that I also have the random quiz function. Currently, as can be seen below, there is no easy way to pass from the random quiz function to the start game function without introducing another if condition and global variables.

```
def startgame(self):
    selection = self.setstable.selectionModel().selectedRows() ## FETCHES THE SELECTED ROWS FROM THE TABLE
    if selection == []: ## CHECKS IF ANY ROW HAS BEEN SELECTED
        CreateOutbox("Selection Not Found","You must select a question set to play!","",QtGui.QMessageBox.Critical)
    else:
        selectedrow = selection[0].row() ## FINDS INDEX OF SELECTED ROW
        playset = allsets[selectedrow] ## SETS THE CURRENT PLAYSET TO THE SELECTED SET
        if self.multichoice.isChecked() == True: ## CHECKS IF THE MULTIPLE CHOICE OPTION IS CHECKED
            print("MULTI CHOICE")

        else:
            print("NOT MULTI CHOICE")

def randomquiz(self):
    query = "SELECT SetName,Author FROM Sets;" ## QUERY TO FETCH ALL SETS FROM DATABASE
    cur.execute(query) ## EXECUTES QUERY
    data = cur.fetchall() ## FETCHES DATA FROM DATABASE
    randsetnum = random.randint(1,len(data)) ## CHOOSES RANDOM NUMBER BETWEEN 1 AND THE NUMBER OF SETS IN THE DATABASE
    playset = data[randsetnum-1] ## GETS THE SETNAME AND AUTHOR OF THE CHOSEN SET (-1 AS INDEXES START FROM 0)
    print(playset)
```

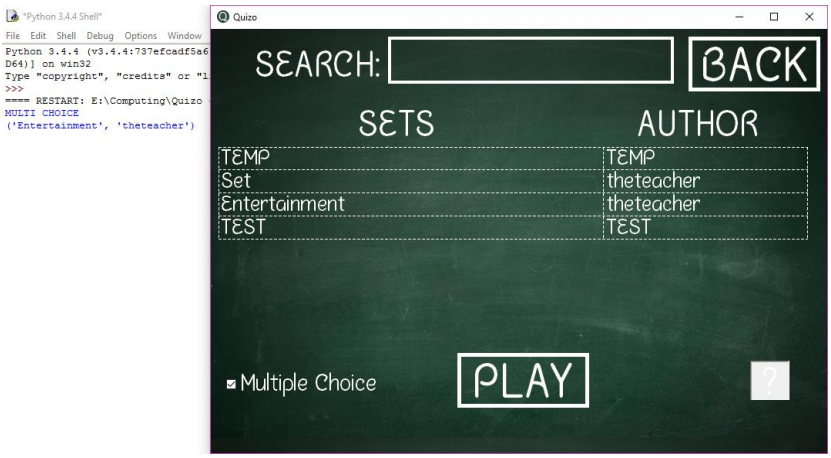
For this reason, I have slightly rewritten these two functions so that I can re-use the same code for checking for multiple choice and eventually moving onto the correct window. My rewritten functions can be seen below, where the new function "findselection" is used when the user presses the play button to find the selected set, and the "startgame" function now just checks if the user is playing with multiple choice, and then will take them to their desired window.

```
def findselection(self):
    selection = self.setstable.selectionModel().selectedRows() ## FETCHES THE SELECTED ROWS FROM THE TABLE
    if selection == []: ## CHECKS IF ANY ROW HAS BEEN SELECTED
        CreateOutbox("Selection Not Found","You must select a question set to play!","",QtGui.QMessageBox.Critical)
    else:
        selectedrow = selection[0].row() ## FINDS INDEX OF SELECTED ROW
        playset = allsets[selectedrow] ## SETS THE CURRENT PLAYSET TO THE SELECTED SET
        self.startgame(playset) ## PASSES THE SELECTED QUESTION SET TO THE START GAME FUNCTION

def startgame(self,playset):
    if self.multichoice.isChecked() == True: ## CHECKS IF THE MULTIPLE CHOICE OPTION IS CHECKED
        print("MULTI CHOICE")
    else:
        print("NOT MULTI CHOICE")
    print(playset)

def randomquiz(self):
    query = "SELECT SetName,Author FROM Sets;" ## QUERY TO FETCH ALL SETS FROM DATABASE
    cur.execute(query) ## EXECUTES QUERY
    data = cur.fetchall() ## FETCHES DATA FROM DATABASE
    randsetnum = random.randint(1,len(data)) ## CHOOSES RANDOM NUMBER BETWEEN 1 AND THE NUMBER OF SETS IN THE DATABASE
    playset = data[randsetnum-1] ## GETS THE SETNAME AND AUTHOR OF THE CHOSEN SET (-1 AS INDEXES START FROM 0)
    self.startgame(playset) ## PASSES THE CHOSEN QUESTION SET TO THE START GAME FUNCTION
```

Even though the base of the code is identical to before, I have introduced a new function, so I will test just to make sure that the chosen sets are being correctly passed to the new startgame function.



Test 1 – Selecting a set – Successful – The program successfully printed the chosen set



Test 2 – Pressing the random set button – Successful – The program successfully printed details for a random set

Now that my code is more efficient, I am able to add the code for actually moving to the next window depending on whether the user is playing multiple choice or not, and passing the chosen question set to that window. In order to pass the question set to the next window, I decided to change the showEvent function to a new function that would only run when called, rather than when the window was shown.

```
def startgame(self,playset):
    if self.multichoice.isChecked() == True: ## CHECKS IF THE MULTIPLE CHOICE OPTION IS CHECKED
        print("MULTI CHOICE")
        Easy_Window.show()
        Easy.startnew(self,playset)
def startnew(self,playset):
    global questions,incorrect,total
    setname = playset[0] ## GETS THE NAME OF THE QUESTION SET
    author = playset[1] ## GETS THE AUTHOR OF THE QUESTION SET

    query = 'SELECT Question,Correct,Wrong1,Wrong2,Wrong3 FROM Questions INNER JOI
    cur.execute(query) ## EXECUTES THE QUERY
    questions = cur.fetchall() ## FETCHES ALL THE QUESTIONS FROM THE DATABASE
    random.shuffle(questions) ## SHUFFLES THE ORDER OF THE QUESTIONS IN THE LIST

    incorrect = [] ## RESETS THE LIST OF USER'S INCORRECT ANSWERS
    total = 0 ## SETS USER'S TOTAL SCORE TO 0
    self.timer = QtCore.QElapsedTimer() ## CREATES A TIMER OBJECT
    self.timer.start() ## STARTS THE TIMER
    self.ask() ## DISPLAYS THE QUESTION TO THE USER
```

When trying to write this function however, I knew that I could not just pass "self" as a parameter to the new class, as it would be referencing the current window, rather than the one the function was being written in, and would cause an error as can be seen below.

```
Traceback (most recent call last):
  File "E:\Computing\Quizo - Development File\Quizo - Developing.py", line 277, in findselection
    self.startgame(playset) ## PASSES THE SELECTED QUESTION SET TO THE START GAME FUNCTION
  File "E:\Computing\Quizo - Development File\Quizo - Developing.py", line 283, in startgame
    Easy.startnew(self,playset)
  File "E:\Computing\Quizo - Development File\Quizo - Developing.py", line 324, in startnew
    self.ask() ## DISPLAYS THE QUESTION TO THE USER
AttributeError: 'PlayGame' object has no attribute 'ask'
```

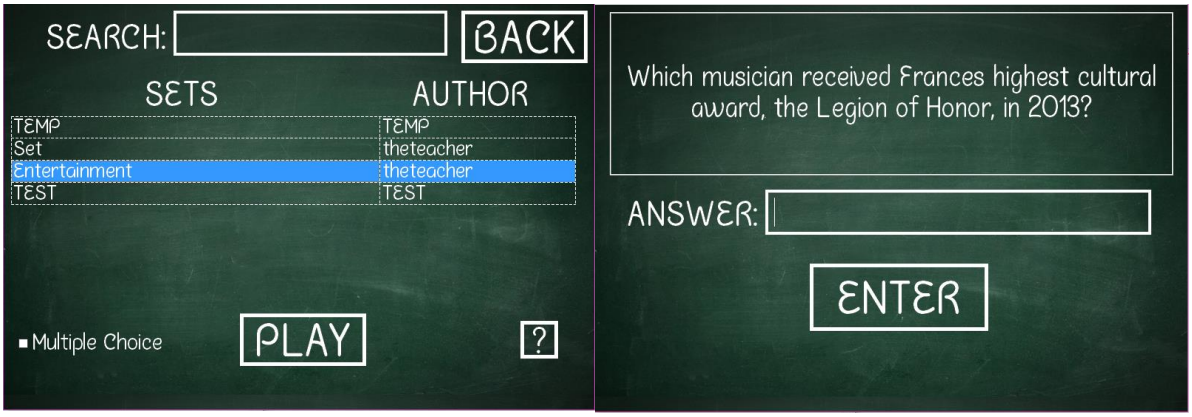
In order to circumnavigate this, I decided to pass the actual window instance as a parameter to the function, as can be seen below.

```
if self.multichoice.isChecked() == True: ## CHECKS IF THE MULTIPLE CHOICE OPTION IS CHECKED
    print("MULTI CHOICE")
    Easy_Window.show() ## SHOWS THE MULTI-CHOICE WINDOW
    Easy.startnew(Easy_Window,playset) ## PASSES THE WINDOW AND CHOSEN PLAYSET TO THE MULTI-CHOICE WINDOW
```

Having adjusted the parameters for the function, I then tested the program to see if it would now pass the playset successfully and begin the quiz, and as shown below, the new parameter fixed the issue and worked as intended.

Now that this method of passing the playset from one window to another was working for multiple choice, I added the functionality for non-multiple choice as well. The code for this can be seen below, alongside a test showing that it also worked the same way.

```
class Hard(QtGui.QMainWindow,Hard_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setupUi(self)
        self.enterbutton.clicked.connect(self.click)
        self.ansedit.returnPressed.connect(self.click) ## RUNS THE CLICK FUNCTION WHE

    def startnew(self,playset):
        global questions,incorrect,total
        setname = playset[0] ## GETS THE TITLE FOR THE QUESTION SET
        author = playset[1] ## GETS THE AUTHOR OF THE QUESTION SET

        query = 'SELECT Question,Correct,Wrong1,Wrong2,Wrong3 FROM Questions INNER JO
        cur.execute(query) ## EXECUTES THE QUERY
        questions = cur.fetchall() ## FETCHES ALL THE QUESTIONS FROM THE DATABASE
        random.shuffle(questions) ## SHUFFLES THE ORDER OF THE QUESTIONS IN THE LIST

        incorrect = [] ## RESETS THE LIST OF USER'S INCORRECT ANSWERS
        total = 0 ## SETS USER'S TOTAL SCORE TO 0
        self.timer = QtCore.QElapsedTimer() ## CREATES A TIMER OBJECT
        self.timer.start() ## STARTS THE TIMER
        self.ask() ## DISPLAYS THE QUESTION TO THE USER
def startgame(self,playset):
    if self.multichoice.isChecked() == True: ## CHECKS IF THE MULTIPLE CHOICE OPTION IS CHECKED
        Easy_Window.show() ## SHOWS THE MULTI-CHOICE WINDOW
        Easy.startnew(Easy_Window,playset) ## PASSES THE WINDOW AND CHOSEN PLAYSET TO THE MULTI-CHOICE WINDOW
    else:
        Hard_Window.show() ## SHOWS THE NON-MULTI-CHOICE WINDOW
        Hard.startnew(Hard_Window,playset) ## PASSES THE WINDOW AND CHOSEN PLAYSET TO THE MULTI-CHOICE WINDOW
    PlayGame_Window.hide() ## HIDES THE PLAY GAME WINDOW
```



Now that my quiz selection screen is feature complete, I can move onto creating the results screen for when a user finishes a quiz.

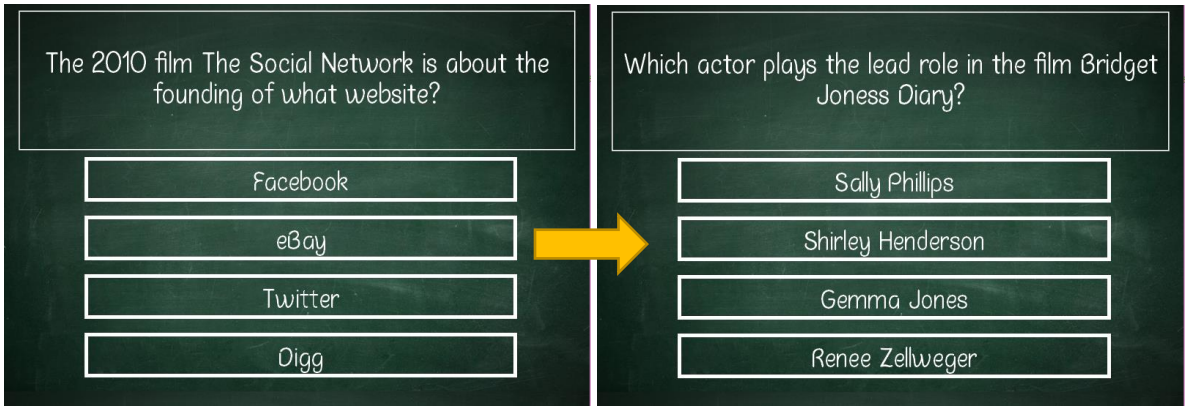
OBJECTIVES 53, AND 54 COMPLETED – USER CAN USE A CHECKBOX TO DETERMINE WHETHER OR NOT THEY WANT TO USE MULTIPLE CHOICE, AND THEN PRESS THE PLAY BUTTON TO START A QUIZ

REMOVING GLOBAL VARIABLES

Moving into the next screen, I have decided that I should try and remove as many global variables as I can now, so that I can use parameter passing to move necessary variables to the results screen.

In order to remove my current global variables, I have given "incorrect", "total", and "choices" a "self" header, so that they can be used by all functions within the class. The code for these changes can be seen below, as well as a small test that shows their functionality is identical, just without the need for global variables.

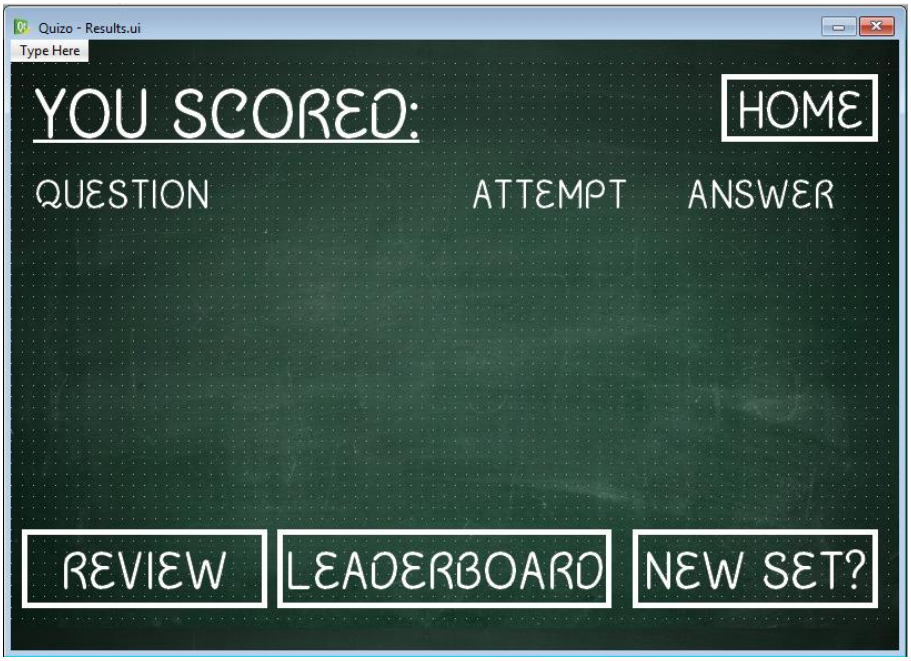
```
        self.incorrect = [] ## RESETS THE LIST OF USER'S INCORRECT ANSWERS
        self.total = 0 ## SETS USER'S TOTAL SCORE TO 0
def ask(self):
    if len(questions) != 0: ## CHECKS THAT THERE ARE STILL QUESTIONS LEFT TO ASK
        current = questions[0] ## SETS THE CURRENT QUESTION
        self.questionlabel.setText(current[0]) ## DISPLAYS THE QUESTION TO THE USER
        self.choices = [1,2,3,4] ## THE POSSIBLE BUTTONS THAT CAN BE PRESSED
        random.shuffle(self.choices) ## SHUFFLES THE ORDER OF THE BUTTONS
        self.ans1button.setText(current[self.choices[0]]) ## SETS THE TEXT OF BUTTON 1
        self.ans2button.setText(current[self.choices[1]]) ## SETS THE TEXT OF BUTTON 2
        self.ans3button.setText(current[self.choices[2]]) ## SETS THE TEXT OF BUTTON 3
        self.ans4button.setText(current[self.choices[3]]) ## SETS THE TEXT OF BUTTON 4
correct = int(self.choices.index(1)) ## FINDS THE BUTTON NUMBER WITH THE CORRECT ANSWER
points = round(1000000/self.timer.restart()) ## SETS POINTS TO THE ELAPSED TIME
if number == correct: ## CHECKS IF THE BUTTON PRESSED IS THE SAME AS THE BUTTON WITH THE CORRECT
    self.total += points ## ADDS POINTS TO TOTAL
else:
    self.total -= points ## DEDUCTS POINTS FROM TOTAL
    self.incorrect.append([questions[0][0],questions[0][self.choices[number]],questions[0][1]])
questions.pop(0) ## REMOVES THE FIRST QUESTION FROM THE QUESTION LIST
self.ask() ## AFTER USER CHOOSES AN ANSWER, NEXT QUESTION IS ASKED
```



As seen above, the program is able to work as it was before, without errors.

CREATING THE RESULTS SCREEN

In the results screen, the user has the ability to view their progress in the quiz they have just taken, seeing their score, and any questions that they answered incorrectly. When making the screens for attempting the quiz, I used two variables to keep track of the user's progress: incorrect, and total.



I will write the code that will pass the current set, list of incorrect questions, and total score, to the results window so that they can be used in creating the table, displaying the score, and eventually, writing scores to the database.

From my original design, I have used the QtDesigner to create the window design that can be seen to the right.

To generate the table, I have used the same code as I did for the question sets table, except this table has an extra column, and I have edited the column widths to better fit the screen.

Seeing as I was making a table with many columns, having a statement for each column to set its width seemed inefficient, for when I make tables with even more columns. To aid this, I created a list with all the widths and then used a loop to iterate through each column, setting its width.

The code for this can be seen below, alongside a quick test showing the user moving to this screen after completing the "Entertainment" quiz.

```
Results.newresults(Results_Window,self.playset,self.incorrect,self.total) ## PASSES THE CURRENT SET TO THE NEXT WINDOW
Results_Window.show() ## SHOWS THE RESULTS WINDOW
Easy_Window.hide() ## HIDES THE CURRENT WINDOW
```

Code that is run when the user answers their final question in multiple choice mode

```
Results.newresults(Results_Window,self.playset,self.incorrect,self.total) ## PASSES THE CURRENT SET TO THE NEXT WINDOW
Results_Window.show() ## SHOWS THE RESULTS WINDOW
Hard_Window.hide() ## HIDES THE CURRENT WINDOW
```

Code that is run when the user answers their final question in non-mulitple choice mode

```
Results_class = uic.loadUiType("Results.ui")[0] ## LOADS UI FILE
```

Written at the start of the program to load the QtDesigner file

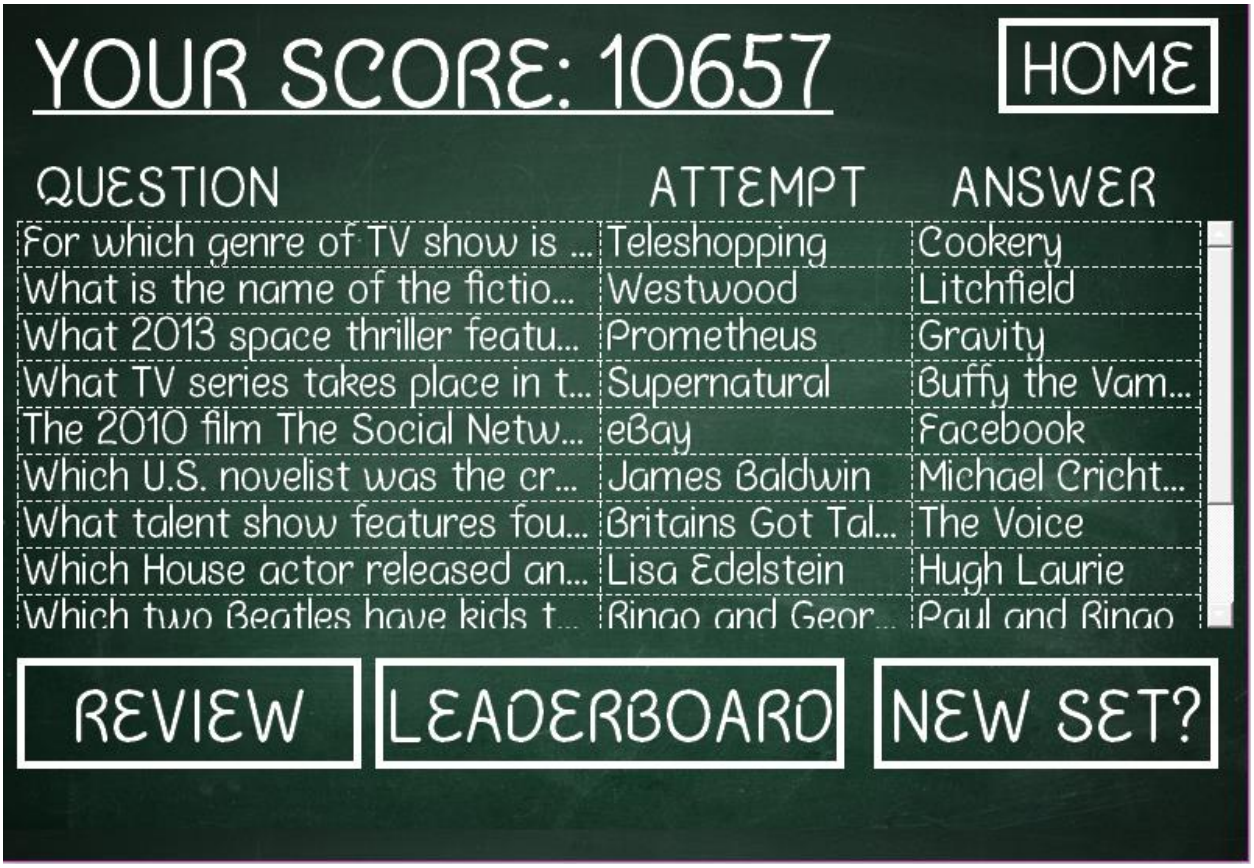
```
class Results(QtGui.QMainWindow,Results_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.setupUi(self)

    def newresults(self,playset,incorrect,total):
        self.scorelabel.setText("YOUR SCORE: "+str(total)) ## DISPLAYS USER'S TOTAL
        model = QtGui.QStandardItemModel(len(incorrect),3) ## CREATES MODEL FOR THE TABLE
        for x in range(0,len(incorrect)): ## LOOPS THROUGH EACH INCORRECT QUESTION
            for i in range(0,3): ## LOOPS THROUGH THE QUESTION, USER'S ANSWER AND CORRECT ANSWER FOR THE QUESTION
                model.setData(model.index(x,i),str(incorrect[x][i])) ## ADDS THE DATA TO THE TABLE MODEL
        self.incorrecttable.setModel(model) ## SETS THE TABLE MODEL
        tempwidth = [375,200,185] ## LIST OF COLUMN WIDTHS
        for i in range(3): ## ITERATES THROUGH EACH COLUMN
            self.incorrecttable.setColumnWidth(i,tempwidth[i]) ## SETS THE WIDTH OF THE COLUMN
        self.incorrecttable.show() ## DISPLAYS THE TABLE
```

The Results class which contains the code to create and utilise the window

```
Results_Window = Results(None)
```

Creates the window object

After answering the final question of the "Entertainment" question set, the results screen was displayed, with the user's score shown at the top, and the table of all their incorrect attempts shown below.



OBJECTIVES 65, 66, AND 67 COMPLETED – AFTER COMPLETING A QUIZ, THE USER IS SENT TO THE RESULTS SCREEN, WHERE THEY CAN SEE THEIR SCORE AND ANY INCORRECTLY ANSWERED QUESTIONS

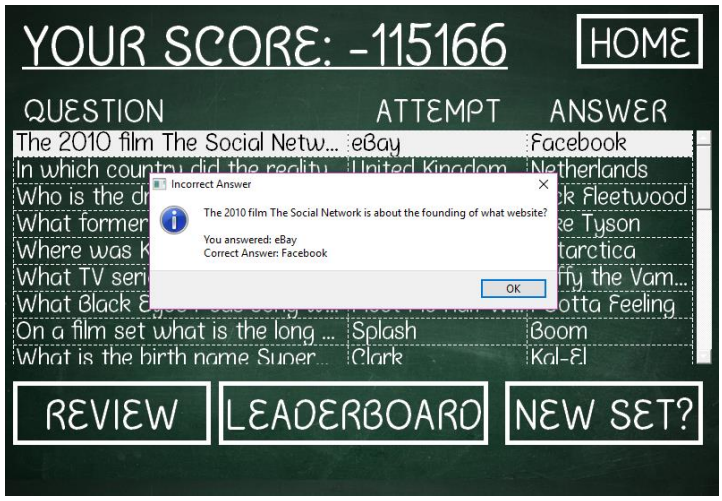
DOUBLE CLICK FUNCTIONALITY AND MORE DETAILED OUTPUT

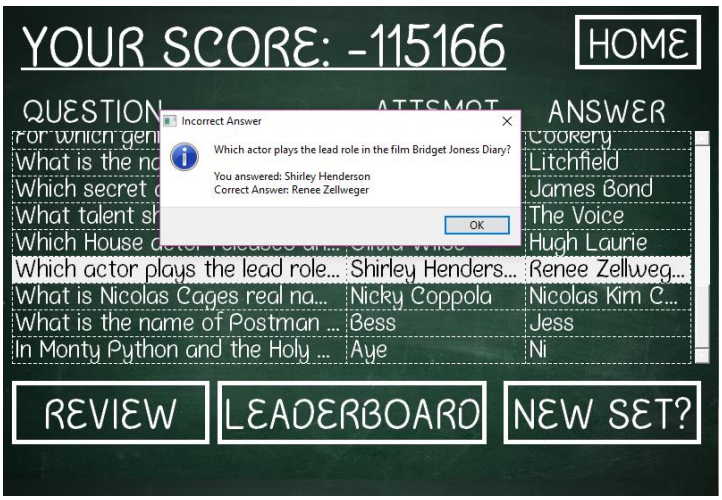
Although I am pleased that the table is being displayed as anticipated, I now realise that the actual questions and some of the answers that are displayed are too long to actually read the text. Without wanting to get into horizontal scroll bars, because they are unappealing in my opinion and aren't the easiest way to manoeuvre a table, I have decided that I will allow the user to double click a question they want to know more about, and it will create a message box that will display all the details.

The code for this addition can be seen below, alongside a test showing how this method looks and works.

```
    def newresults(self,playset,incorrect,total):
        self.incorrect = incorrect ## SETS THE INCORRECT QUESTIONS FOR THE CURRENT SET
```

```
def showincorrect(self,index):
    selection = self.incorrecttable.selectionModel().selectedRows() ## GETS THE SELECTED ROWS FROM THE TABLE
    selectedrow = selection[0].row() ## FINDS INDEX OF SELECTED ROW
    details = self.incorrect[selectedrow] ## GETS THE DETAILS OF THE SELECTED QUESTION
    detailmsg = ("You answered: "+details[1]+"\n"+"Correct Answer: "+details[2]) ## CREATES A MESSAGE SHOWING THE CORRECT ANSWER, AND THE USER'S ANSWER
    CreateOutbox("Incorrect Answer",details[0],detailmsg,QtGui.QMessageBox.Information) ## CREATES THE MESSAGE BOX
```





As shown above, the program is capable of displaying the message box with the correct details of the user's selected question. Being able to review this data is important to the students, as it will allow them to see where they are making mistakes, and encourage them to improve on their weaker areas.

OBJECTIVES 10 AND 68 COMPLETED – PROGRAM CAN RECOGNISE DOUBLE CLICKING A ROW, AND THIS PROVIDES A MESSAGE BOX TO SHOW THE FULL DETAILS OF A USER'S CHOSEN QUESTION

WRITING SCORES TO THE DATABASE

In order for the user to keep track of their progress over time, the program needs to keep a record of their scores as they achieve them. This is what the "Scores" table in the database is for; whenever the user finishes a quiz and goes to the result screen, their score along with the current time, their username, and the set id will be written as a new record to the table so that all scores can be used in the future.

In order to retrieve the set id for the current set, I have decided that instead of using another inner join in my statement, I will return to the play game window and fetch the selected set's id and store it in the playset variable alongside the title and author.

```
query = 'SELECT SetName,Author,SID FROM Sets WHERE SetName LIKE "'+word+'" OR Author LIKE "'+word+'";'
def randomquiz(self):
    query = "SELECT SetName,Author,SID FROM Sets;"
```

Additionally, I have gone back to the login window and globalised the logged in user's username, so that it can be accessed by this window, and eventually the windows surrounding viewing and editing students and question sets.

```
def Login(self):
    global access,cur_user ## GLOBALISES ACCESS LEVEL AND USERNAME SO ALL WINDOWS CAN USE IT
cur_user = username ## SETS CURRENT USER TO THE GIVEN USERNAME
```

After some research into SQLite I have found that there is a built in command for storing the current date and time when a record is stored, which I have shown below. Additionally, unlike previously where I have used string manipulation to create queries for SQLite, this time I have used formatting due to having more data to input. The code for adding new scores can be seen below.

```
query = 'INSERT INTO Scores (Username,SID,Score,Date) VALUES (?,?,?,datetime("now"));' ## QUERY FOR INSERTING SCORE INTO DATABASE
cur.execute(query,(cur_user,str(playset[2]),str(total),)) ## FORMATS THE QUERY WITH CORRECT VARIABLES AND EXECUTES
con.commit() ## WRITES CHANGES TO THE DATABASE
```

Test – Writing a new score to the database – Successful – With user "woodingmp" logged in, I completed the "Entertainment" quiz, and when I had finished the quiz, I took a screenshot of my computer's clock. I then opened up the database and checked the Scores table, to find that the new score had been written successfully, and with the correct time.

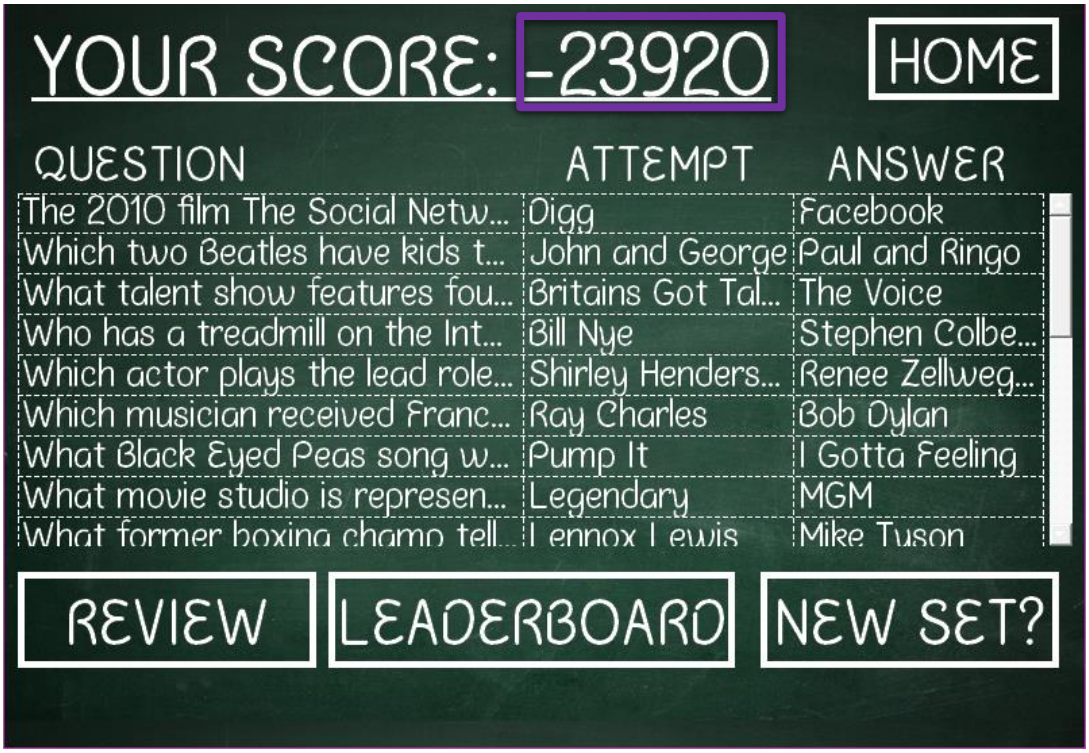


Table: Scores

	Username	SID	Score	Date
	Filter	Filter	Filter	Filter
1	WillTaylor	3	12315	2017-03-10 10:51:24
2	woodingmp	3	-79486	2017-03-10 10:51:47
3	woodingmp	3	14786	2017-03-10 10:53:01
4	woodingmp	3	-105075	2017-03-10 10:53:30
5	woodingmp	3	-23920	2017-03-28 15:42:37

OBJECTIVE 73 COMPLETED – PROGRAM CAN WRITE USERS' SCORES TO THE DATABASE

CREATING THE LEADERBOARD

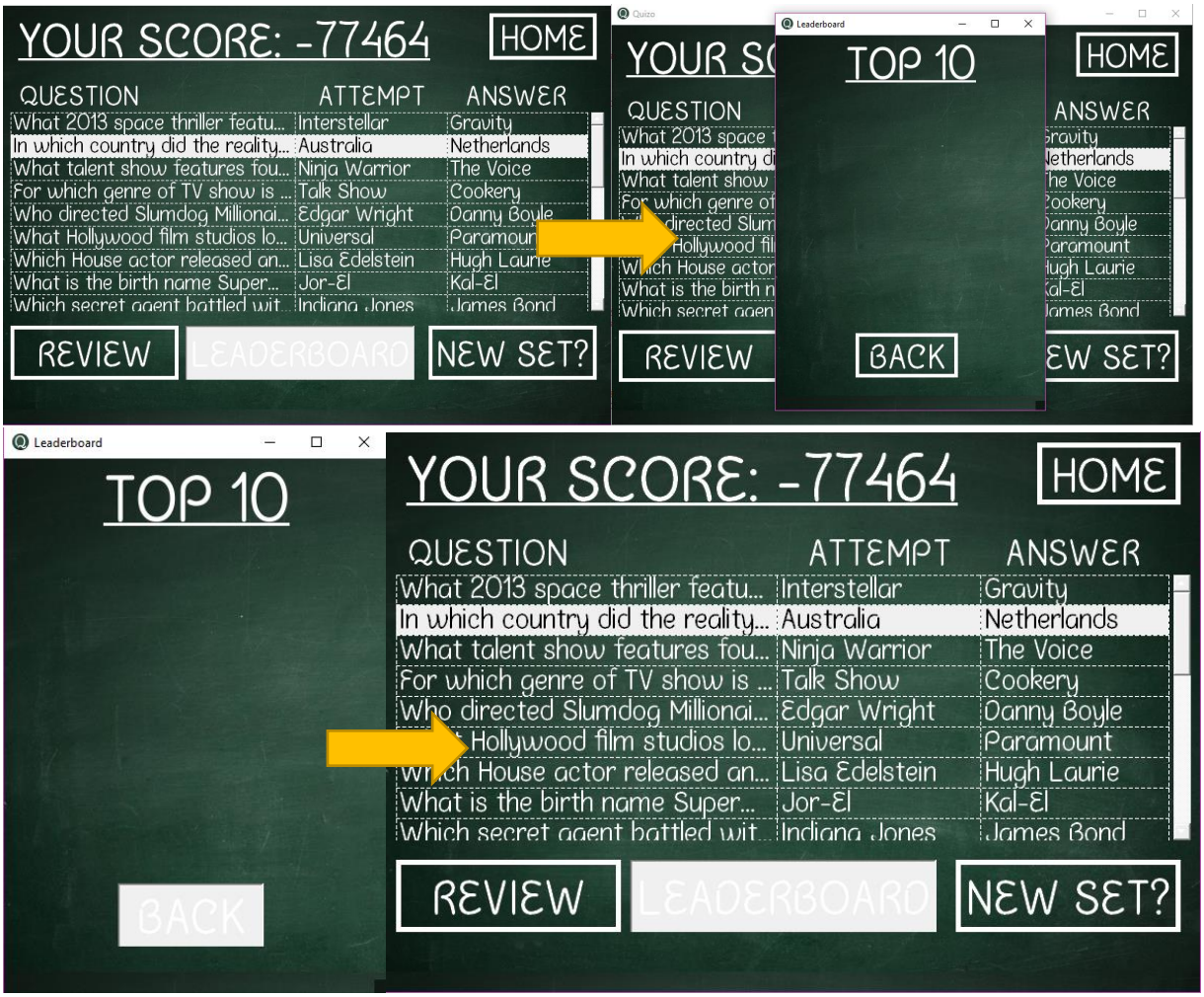
For every question set, my program will be able to produce a leader board to students, displaying the top ten students and their scores. Instead of trying to fit everything into one window, I have decided to create a new window for just displaying a leader board to the students, and its design can be seen to the side.

It is a smaller size window due to the fact that it will be shown over the top of the results window, rather than replacing it.

I have decided to use a table view to display the usernames and scores of the top scorers, and have included a back button to close the window if for what ever reason the user doesn't want to use the close button on the window.

The code for implementing this window can be seen below, followed by a quick test showing that it is viewable.

```
class Results(QtGui.QMainWindow,Results_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.setupUi(self)
        self.incorrecttable.doubleClicked.connect(self.showincorrect)
        self.leaderboardbutton.clicked.connect(self.showleaders)

    def showleaders(self):
        Leaderboard_Window.show() ## SHOWS THE LEADERBOARD

class Leaderboard(QtGui.QMainWindow,Leaderboard_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setupUi(self)
        self.backbutton.clicked.connect(self.Back)

    def Back(self):
        Leaderboard_Window.hide() ## HIDES THE LEADERBOARD
```



As can be seen above, when the leaderboard button is pressed, the leaderboard is shown above the results window, and when the back button is pressed, it is hidden again.

Now that the leaderboard can be displayed, it needs to be capable of showing data. At the moment, my database does not have enough users to show the top ten, and for this reason I have added 10 new accounts, along with a host of new scores for the "Entertainment" quiz. The new data can be seen in the database to the right.

Table: Students

	Username	Password	FirstName	Surname
	Filter	Filter	Filter	Filter
1	woodingmp	af808b049bef56...	Matthew	Wooding
2	WillTaylor	5e884898da280...	Will	Taylor
3	Student1	5e884898da280...	Student	One
4	Student2	5e884898da280...	Student	Two
5	Student3	5e884898da280...	Student	Three
6	Student4	5e884898da280...	Student	Four
7	Student5	5e884898da280...	Student	Five
8	Student6	5e884898da280...	Student	Six
9	Student7	5e884898da280...	Student	Seven
10	Student8	5e884898da280...	Student	Eight
11	Student9	5e884898da280...	Student	Nine
12	Student10	5e884898da280...	Student	10

Table: Scores

	Username	SID	Score	Date
	Filter	Filter	Filter	Filter
1	WillTaylor	3	12315	2017-03-10 10:5...
2	woodingmp	3	-79486	2017-03-10 10:5...
3	woodingmp	3	14786	2017-03-10 10:5...
4	woodingmp	3	-105075	2017-03-10 10:5...
5	woodingmp	3	-23920	2017-03-28 15:4...
6	theteacher	3	-77464	2017-03-28 16:0...
7	Student1	3	-230461	2017-03-28 16:1...
8	Student1	3	-134849	2017-03-28 16:1...
9	Student2	3	-65720	2017-03-28 16:1...
10	Student2	3	-64400	2017-03-28 16:1...
11	Student3	3	-124611	2017-03-28 16:1...
12	Student3	3	-68619	2017-03-28 16:1...
13	Student4	3	-70825	2017-03-28 16:1...
14	Student4	3	-32473	2017-03-28 16:1...
15	Student4	3	-63310	2017-03-28 16:1...
16	Student5	3	-92235	2017-03-28 16:1...
17	Student5	3	-156348	2017-03-28 16:1...
18	Student6	3	-66089	2017-03-28 16:1...
19	Student6	3	27316	2017-03-28 16:1...
20	Student7	3	-21895	2017-03-28 16:1...
21	Student8	3	-44223	2017-03-28 16:1...
22	Student8	3	-26279	2017-03-28 16:1...
23	Student9	3	25824	2017-03-28 16:1...
24	Student10	3	-4999	2017-03-28 16:1...

Now that I have a populated score table, I will write the code for selecting the top ten users in the given question set. Seeing as this is a leaderboard for students, any teacher account scores should not

be included, and users should only appear once in the top ten, even if they have for instance the two best scores. The code for this can be seen below, along with a test to show that the program is fetching the correct scores.

```
def fetchleaders(self,playset):
    query = "SELECT * FROM (SELECT Username,Score FROM Scores WHERE SID=? AND Username NOT IN (SELECT Username FROM Tea
    cur.execute(query,(playset[2],)) ## EXECUTES THE QUERY WITH THE CORRECT SET ID
    allscores = cur.fetchall() ## FETCHES THE SCORES
    model = QtGui.QStandardItemModel(10,2) ## CREATES A TABLE MODEL
    for x in range(10): ## LOOPS THROUGH EACH SCORE
        for i in range(2): ## LOOPS TROUGH EACH COLUMN
            model.setData(model.index(x,i),str(allscores[x][i])) ## SETS THE DATA FROM THE FETCHED SCORES TO THE MODEL
    self.leaderboard.setModel(model) ## SETS THE MODEL TO THE LEADERBOARD
    self.leaderboard.setColumnWidth(0,175) ## SETS THE FIRST COLUMN'S WIDTH
    self.leaderboard.setColumnWidth(1,175) # SETS THE SECOND COLUMN'S WIDTH
    self.leaderboard.show() ## SHOWS THE LEADERBOARD
```

SQLite Query:

```
"SELECT * FROM (SELECT Username,Score FROM Scores WHERE SID=? AND Username NOT IN (SELECT Username FROM Teachers) ORDER BY Score ASC) AS allscores GROUP BY Username ORDER BY Score DESC LIMIT 10;"
```

```
Username NOT IN (SELECT Username FROM Teachers)
```

First the query selects every Username and score for the given question set, but only for student accounts

```
(SELECT Username,Score FROM Scores WHERE SID=? AND Username NOT IN (SELECT Username FROM Teachers) ORDER BY Score ASC) AS allscores
```

The data fetched from this query is then ordered by their scores (ascending) used as a sub-table which I've labelled allscores. (All data is then fetched from this table)

```
GROUP BY Username
```

All the data from the sub-table is then grouped by username; the scores needed to originally be ordered in ascending order because the group by function prevents any duplicate usernames being fetched. The way SQLite does this is by only fetching the last given record with that username, meaning to find that user's best score, it had to be ordered in ascending order.

```
ORDER BY Score DESC
```

After all the scores have unique users, they are then ordered by their score, but descending now, so that the highest scores are at the top.

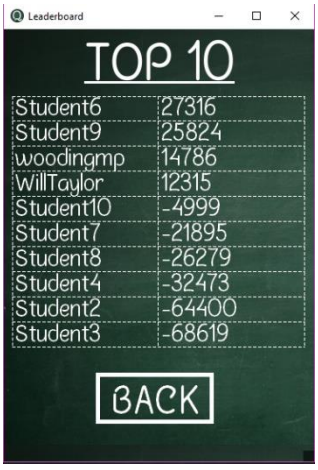
```
LIMIT 10;
```

The query is then limited to ten results, fetching the top ten records found.

Student6	27316
Student9	25824
woodingmp	14786
WillTaylor	12315
Student10	-4999
Student7	-21895
Student8	-26279
Student4	-32473
Student2	-64400
Student3	-68619

In order to check that the top ten results are in fact being fetched, I have used an excel spreadsheet to look through all the current results and find the top ten. From doing this I have found that the correct leaderboard is the one shown to the left.

Having gone into the program and looked at the leaderboard for the "Entertainment" set, I can see that it has in fact displayed the correct scores in the top ten list.



OBJECTIVES 69 AND 70 COMPLETED – THE TOP TEN UNIQUE USERS FOR A SPECIFIC SET CAN BE FETCHED FROM THE DATABASE AND DISPLAYED IN A LEADERBOARD

RETURNING TO THE HOME SCREEN AND PLAY GAME SCREEN

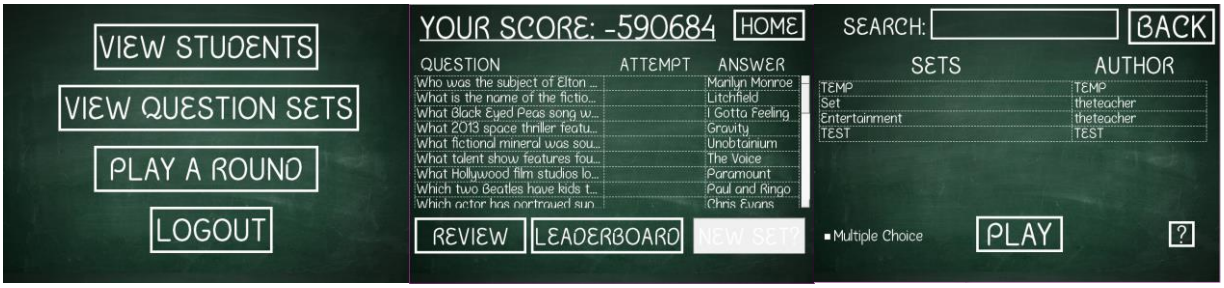
When a user is finished viewing their results, they need to be able to return to a previous window in the program. The options from the results screen are to either go all the way back to their home screen, or to the play game screen, so that they can attempt another quiz.

In the results window there are two buttons for these; the home, and new set buttons. The new set button will do the same for every user, whereas the home button needs to take users to their respective home screen, meaning that the user's access level needs to be checked.

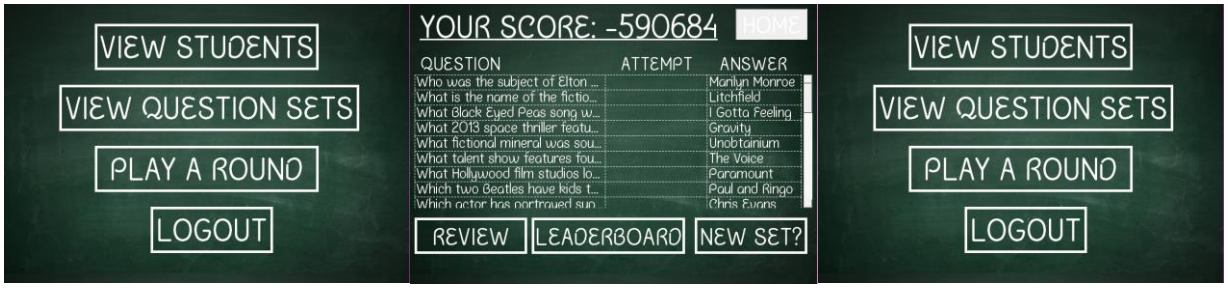
The code for these two buttons can be seen below, as well as tests showing that both work as intended.

```
self.homebutton.clicked.connect(self.home)
self.newbutton.clicked.connect(self.playagain)

def home(self):
    if access == "Students":
        StudentHome_Window.show() ## RETURNS USER TO THE STUDENT HOME
    else:
        TeacherHome_Window.show() ## RETURNS USER TO THE TEACHER HOME
    Results_Window.hide() ## HIDES THE RESULTS WINDOW

def playagain(self):
    PlayGame_Window.show() ## RETURNS USER TO THE PLAY GAME SCREEN
    Results_Window.hide() ## HIDES THE RESULTS WINDOW
```

Test 1 – Pressing the New Set button – Successful – Upon pressing this button, the program sent me back to the Play Game Screen as intended



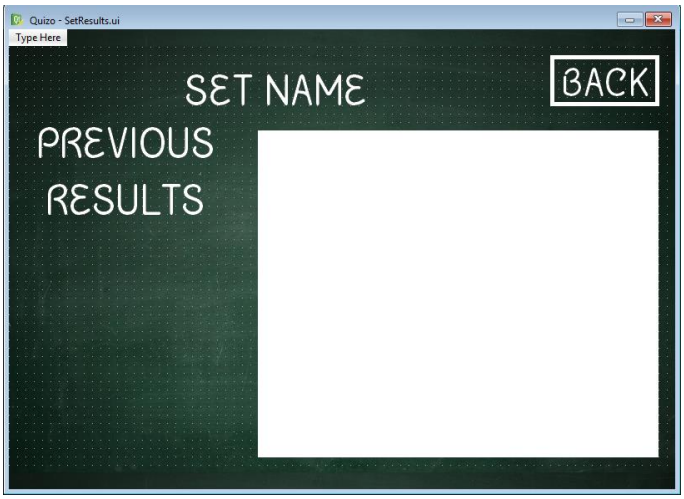
Test 2 – Pressing the Home button as a Teacher - Successful – Upon pressing this button, the program sent me back to the Teacher Home Screen as intended



Test 3 – Pressing the Home button as a Student – Successful – Upon pressing this button, the program sent me back to the Student Home Screen as intended

OBJECTIVE 72 COMPLETED – USER CAN RETURN TO THEIR HOME SCREEN OR THE PLAY GAME SCREEN

CREATING THE ANALYSIS SCREEN



Now that the user is able to attempt sets multiple times, they need to be able to track how they are progressing. Following my initial design for this screen, I have used QtDesigner to create the window which will house a list of all the users scores in a given question set, as well as a graph tracking their scores over time.

The first part of making this screen work, is to fetch all the previous results from the database of the current user. Once all the results are fetched, I will then be able to display them in a list under the previous results heading. The code for this can be seen below.

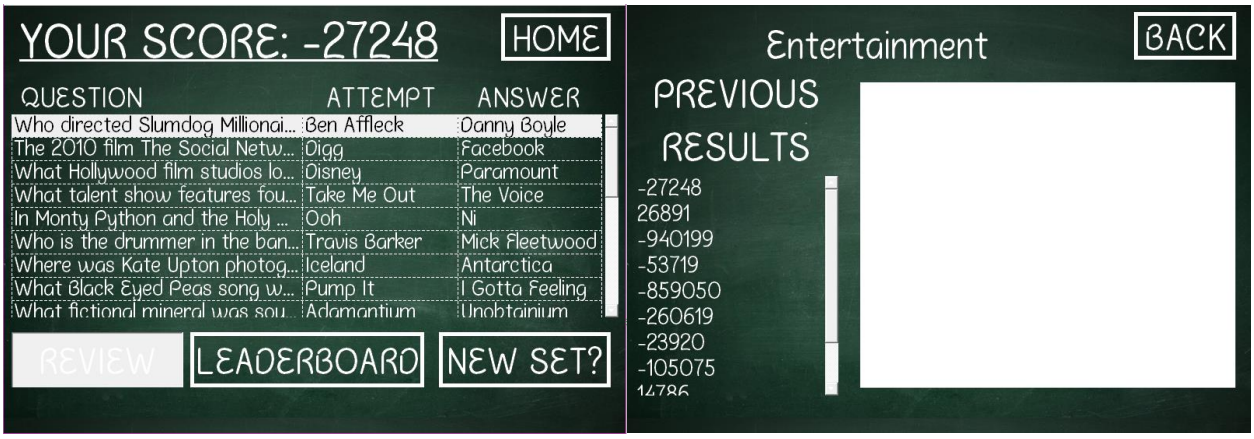
```
class SetAnalysis(QtGui.QMainWindow,SetAnalysis_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.setupUi(self)
        self.backbutton.clicked.connect(self.Back)

    def fetchresults(self,playset,user):
        self.setlabel.setText(playset[0]) ## SETS LABEL TEXT TO THE NAME OF THE SET
        query = "SELECT Score FROM Scores WHERE SID=? AND Username=? ORDER BY Date DESC;" ## FETCHES ALL THE USER'S RESULTS IN THE CURRENT SET
        cur.execute(query,(playset[2],user,)) ## EXECUTES THE QUERY
        allscores = cur.fetchall() ## FETCHES THE DATA
        for score in allscores: ## LOOPS THROUGH EACH SCORE
            item = QtGui.QListWidgetItem() ## CREATES A NEW ITEM IN THE LIST
            item.setText(str(score[0])) ## SETS THE TEXT OF THE ITEM TO THE SCORE
            self.resultstable.addItem(item) ## ADDS THE ITEM TO THE LIST WIDGET

    def Back(self):
        SetAnalysis_Window.hide() ## HIDES THE SET ANALYSIS WINDOW
        Results_Window.show() ## SHOWS THE RESULTS WINDOW
```

In the Results class:

```
def review(self):
    SetAnalysis.fetchresults(SetAnalysis_Window,self.playset,cur_user) ## PASSES THE NECESSARY PARAMETERS TO START FETCHING RESULTS FOR ANALYSIS
    SetAnalysis_Window.show() ## SHOWS THE ANALYSIS WINDOW
    Results_Window.hide() ## HIDES THE RESULTS WINDOW
```

Now that the class has been coded, I can test its functionality, which can be seen below.

Test – Review results as "woodingmp" – Successful – To test that the correct results are being fetched from the database, and displayed in date order, I have logged in as "woodingmp" as it currently has the most scores in the database. Once I reached the results screen I pressed the review button, and checked the list of scores against the scores from the database, and as hoped the results were the same and in the correct date order, with the latest results at the top.




```
1 SELECT Score,Date FROM Scores WHERE SID=3 AND Username="woodingmp" ORDER BY Date DESC;
```

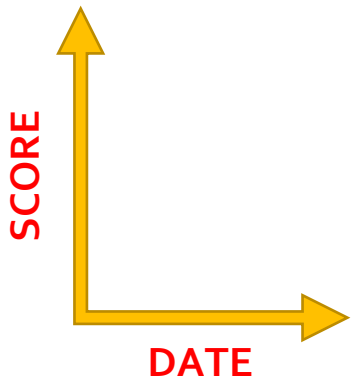
	Score	Date
1	-27248	2017-03-28 18:52:18
2	26891	2017-03-28 18:47:11
3	-940199	2017-03-28 18:41:44
4	-53719	2017-03-28 18:04:31
5	-859050	2017-03-28 18:04:22
6	-260619	2017-03-28 16:53:43
7	-23920	2017-03-28 15:42:37
8	-105075	2017-03-10 10:53:30
9	14786	2017-03-10 10:53:01
10	-79486	2017-03-10 10:51:47

Shown to the left is all the scores posted by "woodingmp" in the "Entertainment" quiz, ordered by date, and as can be seen when compared to the analysis window above, they are the same results, in the correct order.

OBJECTIVES 38 AND 39 COMPLETED – USERS CAN NOW ACCESS IN DEPTH ANALYSIS OF THEIR RESULTS, SEEING EVERY SCORE THEY'VE EVER HAD IN A SET, SHOWN IN DATE ORDER

GENERATING THE LINE GRAPH

Now that the user is able to see a list of all their scores, they next need to be able to view a graph of these results, allowing them to see how their scores have progressed over time. The axis for this graph will be date and score, as shown in the diagram.
Due to the fact that my program can dispense both positive and negative scores, the graph will need to have a line at zero to show users what side of the graph they are on.



In order to create this graph, I will be writing in HTML and then using a PyQt webview to display the graph inside my program. Before starting my program, I had done a bit of work with HTML and JavaScript so that I had a base knowledge of what I would need for this part of the program.

Before I started coding straight into Python, I have used JSBin to design the graph and make sure it looks acceptable before I move it into the window.

HTML

```
<!DOCTYPE html>
<html>
<body>

<canvas id="myCanvas" width="460" height="365"
style="border:1px solid #d3d3d3;">
</canvas>

</body>
</html>
```

JavaScript

```
var canvas = document.getElementById("myCanvas"); // CREATES THE CANVAS
var ctx = canvas.getContext("2d"); // SETS 2D CANVAS
var points=new Array(0,20,2,18,4,16,6,14,8,12,10,5,6,7,8,10,5); // LIST OF Y CO-ORDINATES FOR POINTS
var zerolocation=10; // SETS THE Y CO-ORDINATE OF THE ZERO LINE
var xoffset=25; // SETS HORIZONTAL SPACING BETWEEN EACH POINT
var yoffset=30; // SETS VERTICAL STRETCH OF EACH POINT
var old_x=xoffset; // SETS STARTING X VALUE
var old_y=canvas.height-yoffset*points[0]; // SETS STARTING Y VALUE

for (var i = 0; i < points.length; i++)  // LOOPS THROUGH EACH POINT IN ARRAY
{if (i>0){ctx.moveTo(old_x,old_y);} // IF THERE ARE POINTS TO GO TO, STARTS DRAWING A LINE FROM CURRENT POSITION
old_x=xoffset+xoffset*(i); // CALCULATES X CO-ORDINATE OF POINT
old_y=canvas.height-yoffset*points[i]*.5; // CALCULATES Y CO-ORDINATE OF POINT
ctx.lineTo(old_x,old_y); // DRAWS A LINE FROM PREVIOUS POINT TO NEW POINT
}

for (var x = 0; x < 47; x++) // LOOPS THROUGH NUMBER OF DASHES
{
ctx.moveTo(x*10,canvas.height-yoffset*zerolocation*.5); // GOES TO START OF LINE
ctx.lineTo((x*10)+5,canvas.height-yoffset*zerolocation*.5); // DRAWS HORIZONTAL DASH
}

ctx.font = "16px Arial"; // SETS FONT FOR GRAPH
ctx.fillStyle = "black"; // SETS FILL STYLE
ctx.textAlign = "left"; // ALLIGNS TEXT LEFT
ctx.fillText(0,0,(canvas.height-yoffset*zerolocation*.5)); // WRITES IN THE 0

ctx.stroke();
```

The code above draws out a simple two dimensional graph, that when given a list of points is able to plot them, and draw a line between them. The code achieves this by using offsets for both the x and y directions, making sure that the scale of the graph remains constant.

The issue with this graph when I initially started creating it, is that any negative points will be drawn below the actual canvas.

The way to show the negative points on the graph is to scale all the points up, so that they are all positive, which I achieved simply by adding the same constant to each point's y co-ordinate. This makes the graph actually fit vertically on the canvas, which is why the zero line is necessary to show where the positive scores start and the negatives end.

Another issue with my graph at the moment is that it can only take a certain number of points before it starts going off the screen. When I move the graph into python, I will need to code in some form of scalability so that it can continue to show every result, even when there are more than would normally fit on the graph.

In terms of aesthetic, I am happy with how the graph looks, as it needs to be clean and easy to interpret, rather than being very colourful and animated; while those things may look nice, they don't help when trying to actually read the information from the graph.

Given that I am happy with the graph except for the scaling issues, I have copied my code into a text document, rather than copying it straight into python. I have also replaced the array, zero location, and

x offset with unique text that I will be able to find and replace when I want to use points from my program.

```
<!DOCTYPE html>
<html>
<body>

<canvas id="myCanvas" width="460" height="365"
style="border:1px solid #d3d3d3;">
</canvas>

<script>

var canvas = document.getElementById("myCanvas"); // CREATES THE CANVAS
var ctx = canvas.getContext("2d"); // SETS 2D CANVAS
var points=new Array(POINTS-LIST-HERE); // LIST OF Y CO-ORDINATES FOR POINTS
var zerolocation=ZERO-COORDS-HERE; // SETS THE Y CO-ORDINATE OF THE ZERO LINE
var xoffset=X-OFFSET-HERE; // SETS HORIZONTAL SPACING BETWEEN EACH POINT
var yoffset=30; // SETS VERTICAL STRETCH OF EACH POINT
var old_x=xoffset; // SETS STARTING X VALUE
var old_y=canvas.height-yoffset*points[0]; // SETS STARTING Y VALUE

for (var i = 0; i < points.length; i++)  // LOOPS THROUGH EACH POINT IN ARRAY
{if (i>0){ctx.moveTo(old_x,old_y);} // IF THERE ARE POINTS TO GO TO, STARTS DRAWING A LINE FROM CURRENT POSITION
old_x=xoffset+xoffset*(i); // CALCULATES X CO-ORDINATE OF POINT
old_y=canvas.height-yoffset*points[i]*.5; // CALCULATES Y CO-ORDINATE OF POINT
ctx.lineTo(old_x,old_y); // DRAWS A LINE FROM PREVIOUS POINT TO NEW POINT
}

for (var x = 0; x < 47; x++) // LOOPS THROUGH NUMBER OF DASHES
{
ctx.moveTo(x*10,canvas.height-yoffset*zerolocation*.5); // GOES TO START OF LINE
ctx.lineTo((x*10)+5,canvas.height-yoffset*zerolocation*.5); // DRAWS HORIZONTAL DASH
}

ctx.font = "16px Arial"; // SETS FONT FOR GRAPH
ctx.fillStyle = "black"; // SETS FILL STYLE
ctx.textAlign = "left"; // ALLIGNS TEXT LEFT
ctx.fillText(0,0,(canvas.height-yoffset*zerolocation*.5)); // WRITES IN THE 0

ctx.stroke();

</script>

</body>
</html>
```

In order to allow for scaling of my graph, I will need to first get all of the scores in a list. The best way to scale is to first get both the greatest and smallest scores for scaling vertically, and also getting the number of scores, for scaling horizontally.

```
plotlist = [] ## CONTAINS THE Y CO-ORDINATES OF POINTS TO BE PLOTTED

for score in allscores: ## LOOPS THROUGH EACH SCORE
    item = QtGui.QListWidgetItem() ## CREATES A NEW ITEM IN THE LIST
    item.setText(str(score[0])) ## SETS THE TEXT OF THE ITEM TO THE SCORE
    self.resultstable.addItem(item) ## ADDS THE ITEM TO THE LIST WIDGET
    plotlist.append(score[0]) ## ADDS THE CURRENT SCORE TO THE PLOTLIST
```

Scaling horizontally is simple enough, as I just need to take the number of points that I have, and distribute them evenly across the width of the graph. To do this, I have created a formula that takes the width of the canvas in pixels, and divided it by the number of points to be plotted.

```
xoffset = (450/len(plotlist)) ## GETS THE HORIZONTAL SPACING BETWEEN EACH POINT
```

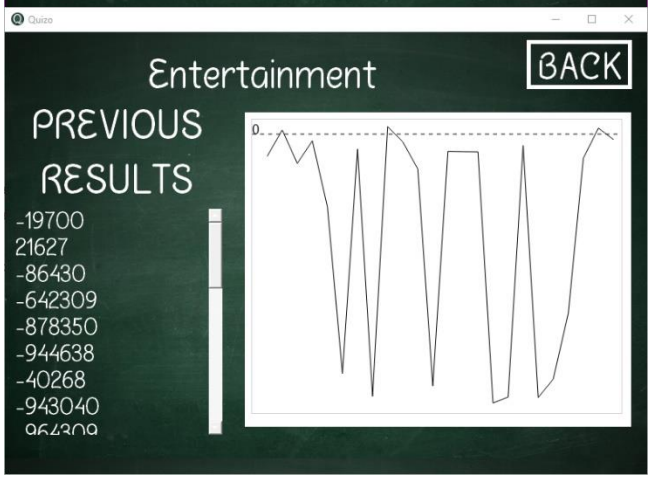
Scaling vertically is a bit more challenging, as I need to take the maximum and minimum points, find the range of values between them, and then use a formula to create a scale that will display all values. To create the formula, I have done a little bit of trial and error to find what the limits of the floor and ceiling of the graph's canvas is.

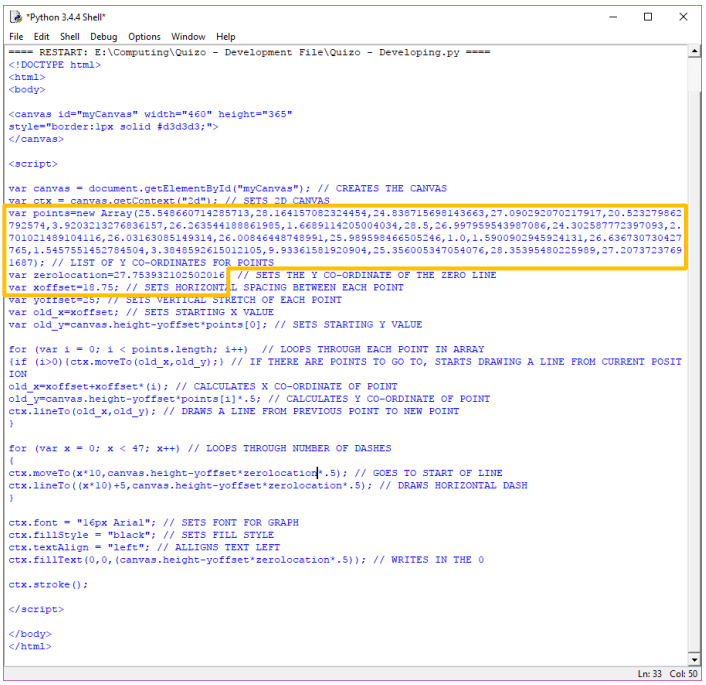
```
largest = max(plotlist) ## FINDS THE HIGHEST SCORE
smallest = min(plotlist) ## FINDS THE LOWEST SCORE
FLOOR = 1 ## MINIMUM Y CO-ORDINATE OF POINT - CAPITALISED FOR CONSTANT
CEILING = 28.5 ## MAXIMUM Y CO-ORDINATE OF POINT - CAPITALISED FOR CONSTANT
zeroline = (((CEILING-FLOOR)*(0-smallest))/(largest-smallest))+FLOOR ## FINDS THE Y CO-ORDINATE OF THE ZERO LINE
for i in range(0,len(plotlist)): ## LOOPS THROUGH EACH SCORE IN THE PLOTLIST
    plotlist[i] = (((CEILING-FLOOR)*(plotlist[i]-smallest))/(largest-smallest))+FLOOR ## ADJUSTS THE VALUE OF THE Y CO-ORDINATE SO THAT IT FITS ON THE GRAPH
```

Now that I have all the values required to draw the graph in my program, I can load the graph file that I created, and replace the values I had set, with the new plotlist, zeroline, and xoffset.

```
file = open("GraphFile.txt","rt") ## OPENS THE GRAPH FILE
contents = file.read() ## SAVES THE CONTENTS OF THE FILE TO A VARIABLE
pointslist=",".join([str(x) for x in plotlist]) ## JOINS TOGETHER ALL THE POINTS INTO ONE STRING SO IT'S FORMATTED CORRECTLY
contents = contents.replace("POINTS-LIST-HERE",pointslist) ## ADDS THE POINTSLIST TO THE HTML FILE
contents = contents.replace("ZERO-COORDS-HERE",str(zeroline)) ## ADDS THE ZEROLINE CO-ORDINATE TO THE HTML FILE
contents = contents.replace("X-OFFSET-HERE",str(xoffset)) ## ADDS THE XOFFSET TO THE HTML FILE
print(contents)
self.webView.setHtml(contents) ## SETS THE VIEW OF THE WEB VIEWER IN THE WINDOW
```

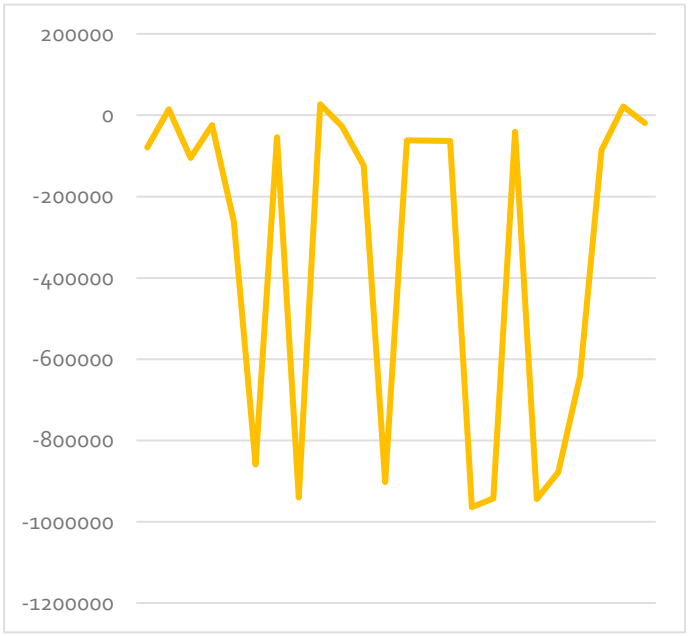
With the code now implemented to draw and scale the graph in the window, I will test this new functionality by logging in as "woodingmp" and finishing the "Entertainment" quiz. Once in the results screen I will try to go to the analysis screen and see whether or not the graph has been drawn correctly. I have also added a print command into the code to check whether the plotlist, zeroline, and xoffset are added to the HTML code correctly.

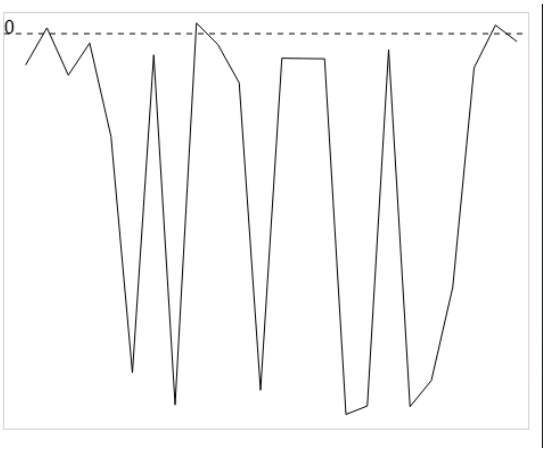




The correct variables were successfully added to the HTML code, however to make sure the graph is correct, I have taken the values and made a graph in Excel to check that the graph drawn in my program is correct.

-79486
14786
-105075
-23920
-260619
-859050
-53719
-940199
26891
-27248
-124399
-902998
-62078
-62913
-63593
-964309
-943040
-40268
-944638
-878350
-642309
-86430
21627
-19700





As can be seen above, the graph that was generated in Excel using the same data as my program has generated a graph that is identical to the one that my program is capable of creating. This shows that the scalability and drawing of graphs in my program has worked successfully.

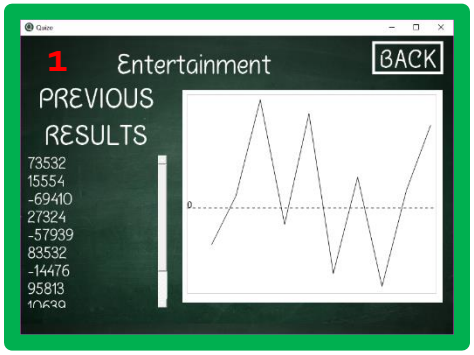
Before deciding that this feature is fully working, I wanted to do some extreme testing to make sure that the graph drawing had the ability to scale in size if a user had completed the same quiz a large number of times. In order to generate all these scores, I have used the random module and a for loop to generate the number of results I needed for each test.

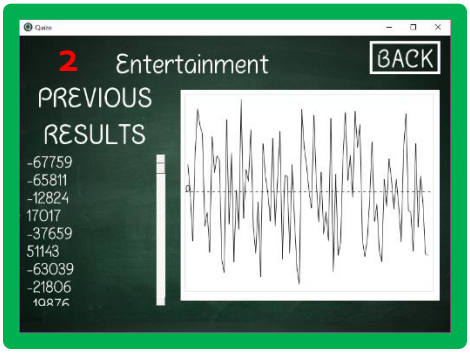
Additionally, in order to see how well the program scales in terms of time taken to process all these results, I have added a timer, that will print the elapsed time when the program has finished drawing the graph. Using this, I will able to see whether the time complexity that I expect the program to be is accurate; from the code that I have written, I expect the time complexity of this section of the program to be exponential due to the multiple for loops which have to iterate through each result in order to scale it for the graph.

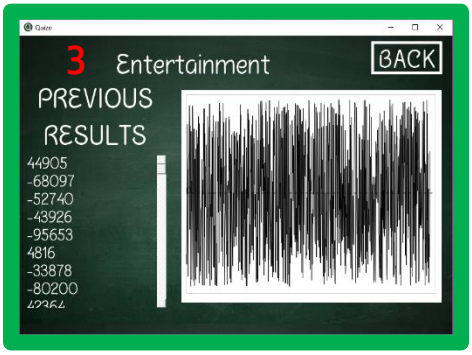
The code for these temporary changes can be seen below, along with my test table for documenting my findings.

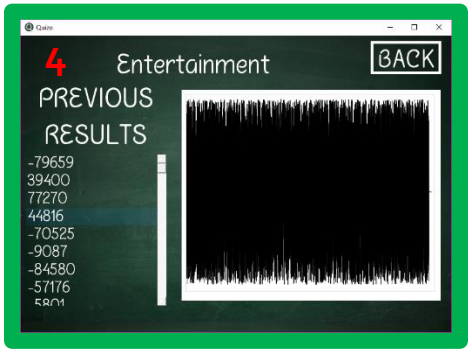
Test Number	Number of Results	Elapsed Time (Seconds)
1	10	0.022
2	100	0.021
3	1000	0.231
4	20000	48.856
5	50000	Program Crashed

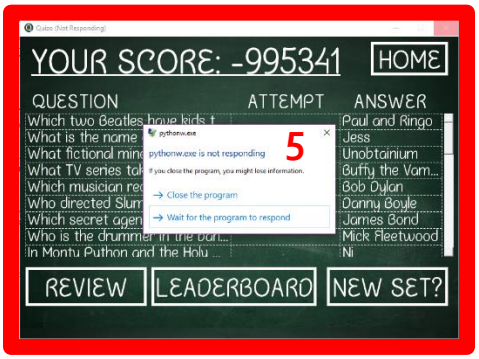
```
self.timer = QtCore.QElapsedTimer() ## CREATES A TIMER OBJECT
self.timer.start() ## STARTS THE TIMER

self.setlabel.setText(playset[0]) ## SETS LABEL TEXT TO THE NAME OF THE SET
self.resultstable.clear() ## CLEARS THE CURRENT LIST
#query = "SELECT Score FROM Scores WHERE SID=? AND Username=? ORDER BY Date DESC;
#cur.execute(query,(playset[2],user,)) ## EXECUTES THE QUERY
#allscores = cur.fetchall() ## FETCHES THE DATA

plotlist = [] ## CONTAINS THE Y CO-ORDINATES OF POINTS TO BE PLOTTED

#for score in allscores: ## LOOPS THROUGH EACH SCORE
#    item = QtGui.QListWidgetItem() ## CREATES A NEW ITEM IN THE LIST
#    item.setText(str(score[0])) ## SETS THE TEXT OF THE ITEM TO THE SCORE
#    self.resultstable.addItem(item) ## ADDS THE ITEM TO THE LIST WIDGET
#for score in reversed(allscores):
#    plotlist.append(score[0]) ## ADDS THE CURRENT SCORE TO THE PLOTLIST

allscores = [] ## RESETS LIST OF SCORES
for i in range(10): ## LOOPS THROUGH EACH NEW RESULT
        score = random.randint(-100000,100000) ## GENERATES A RANDOM RESULT
        allscores.append(score) ## ADDS IT TO THE SCORE LIST
        item = QtGui.QListWidgetItem() ## CREATES AN ITEM
        item.setText(str(score)) ## SETS THE TEXT OF THE ITEM
        self.resultstable.addItem(item) ## ADDS THE ITEM TO THE LIST WIDGET
for score in reversed(allscores): ## LOOPS THROUGH EACH SCORE IN REVERSE
    plotlist.append(score) ## ADDS THE SCORE TO THE PLOTLIST

xoffset = (450/len(plotlist)) ## GETS THE HORIZONTAL SPACING BETWEEN EACH POINT

largest = max(plotlist) ## FINDS THE HIGHEST SCORE
smallest = min(plotlist) ## FINDS THE LOWEST SCORE
FLOOR = 1 ## MINIMUM Y CO-ORDINATE OF POINT - CAPITALISED FOR CONSTANT
CEILING = 28.5 ## MAXIMUM Y CO-ORDINATE OF POINT - CAPITALISED FOR CONSTANT
zeroline = (((CEILING-FLOOR)*(0-smallest))/(largest-smallest))+FLOOR ## FINDS THE
for i in range(0,len(plotlist)): ## LOOPS THROUGH EACH SCORE IN THE PLOTLIST
    plotlist[i] = (((CEILING-FLOOR)*(plotlist[i]-smallest))/(largest-smallest))+F

file = open("GraphFile.txt","rt") ## OPENS THE GRAPH FILE
contents = file.read() ## SAVES THE CONTENTS OF THE FILE TO A VARIABLE
pointslist=",".join([str(x) for x in plotlist]) ## JOINS TOGETHER ALL THE POINTS
contents = contents.replace("POINTS-LIST-HERE",pointslist) ## ADDS THE POINTSLIST
contents = contents.replace("ZERO-COORDS-HERE",str(zeroline)) ## ADDS THE ZEROLIN
contents = contents.replace("X-OFFSET-HERE",str(xoffset)) ## ADDS THE XOFFSET TO
print(contents)
self.webView.setHtml(contents) ## SETS THE VIEW OF THE WEB VIEWER IN THE WINDOW

elapsed = self.timer.restart() ## GETS THE ELAPSED TIME
self.timer.invalidate() ## STOPS THE TIMER
print(elapsed)
```











For the 5th test, the program crashed while trying to process all the data, for this reason I will need to have some sort of warning to users if they attempt to use too much data.

From my testing, I can see that starting with a fairly low number of results doesn't cause any issue with a time delay, and the program is able to efficiently process all the information. However, as soon as the program was getting to lists with tens of thousands of scores, it began to slow down massively, taking nearly a minute to process 20,000 results, and then crashing at 50,000.

From this I can safely say that the time complexity of this section of the program is exponential as I had originally thought, and that I will need to inform users with large numbers of scores that there could be issues when trying to process so much data. Although I never expect a single user to log anywhere near a thousand results in the same question set, I need to put a system in place for extreme purposes.

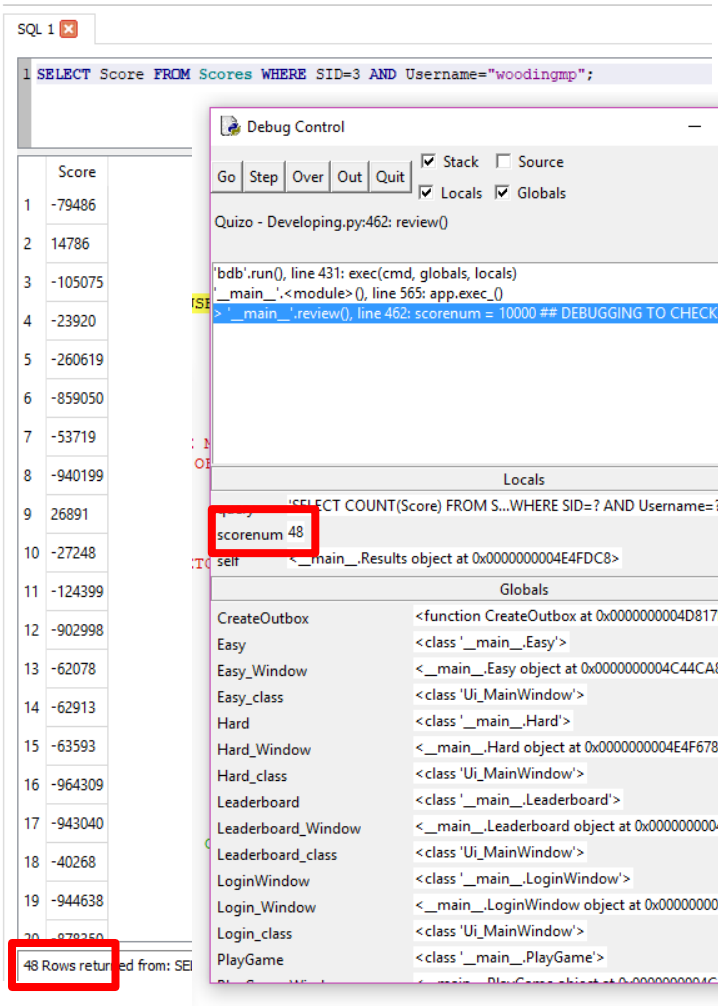
In order to implement this, I will simply check the length of the list of scores before the program begins to process all the scores, informing the user that it could take some time, and allowing them to either continue or back out. In order to allow for multiple options in the message box that appears, I have decided not to use the CreateOutbox function, as this is a specialised case, and if I were to use the function, I would have to change the number of inputs it could take, as well as adapt all the times it is used within the program to adhere to this.

```
def review(self):
    query = "SELECT COUNT(Score) FROM Scores WHERE SID=? AND Username=?;" ## QUERY THAT FETCHES THE NUMBER OF SCORES BY THE USER IN A SET
    cur.execute(query,(self.playset[2],cur_user,)) ## EXECUTES THE QUERY WITH THE CORRECT PARAMETERS
    scorenum = cur.fetchone()[0] ## FETCHES THE ACTUAL NUMBER OF SCORES
    scorenum = 10000 ## DEBUGGING TO CHECK THE MESSAGE BOX WORKS
    if scorenum > 1000:
        outbox = QtGui.QMessageBox() ## CREATES THE MESSAGE BOX
        outbox.setWindowTitle("Set Analysis") ## SETS THE TITLE OF THE MESSAGE BOX
        outbox.setText("You are trying to analysis a set where you have more than 1000 scores.") ## SETS THE MAIN TEXT OF THE MESSAGE BOX
        outbox.setInformativeText("This may take some time to process, would you like to continue?") ## SETS ADDITIONAL TEXT OF THE MESSAGE BOX
        outbox.setIcon(QtGui.QMessageBox.Information) ## SETS THE ICON OF THE MESSAGE BOX
        outbox.setStandardButtons(QtGui.QMessageBox.Ok | QtGui.QMessageBox.Cancel) ## SETS THE BUTTONS FOR THE MESSAGE BOX
        choice = outbox.exec() ## EXECUTES THE MESSAGE BOX AND RETRIEVES THE BUTTON PRESSED
        if choice == QtGui.QMessageBox.Ok: ## CHECKS IF OK WAS PRESSED
            SetAnalysis.fetchresults(SetAnalysis_Window,self.playset,cur_user) ## PASSES THE NECESSARY PARAMETERS TO START FETCHING RESULTS FOR ANALYSIS
            SetAnalysis_Window.show() ## SHOWS THE ANALYSIS WINDOW
            Results_Window.hide() ## HIDES THE RESULTS WINDOW
```

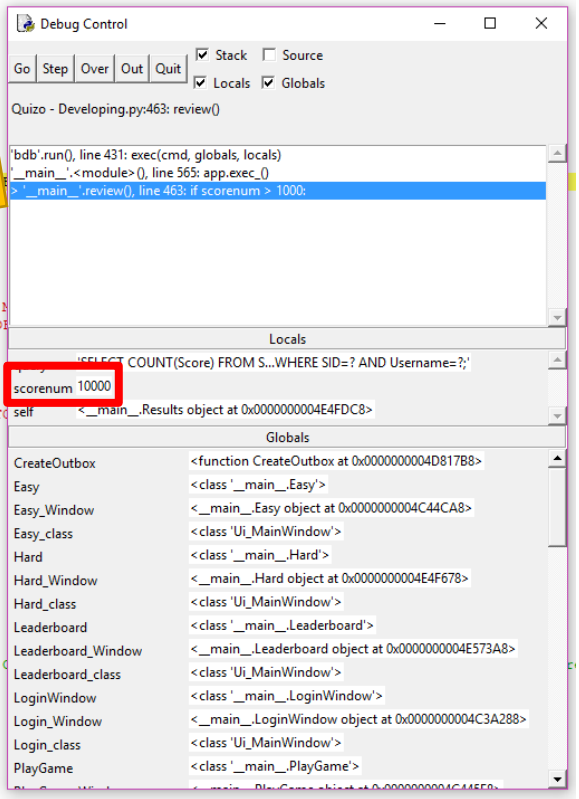
In order to test that the query is fetching the correct value I will be using the python debugger along with some breakpoints that I have set (highlighted in yellow) to step through this part of the program, so that when I reach the execution of the query, the first value of "scorenum" should be the actual number of scores achieved by "woodingmp".

```
def review(self):
    query = "SELECT COUNT(Score) FROM Scores WHERE SID=? AND Username=?;" ## QUERY THAT FETCHES THE NUMBER OF SCORES BY THE USER IN A SET
    cur.execute(query,(self.playset[2],cur_user,)) ## EXECUTES THE QUERY WITH THE CORRECT PARAMETERS
    scorenum = cur.fetchone()[0] ## FETCHES THE ACTUAL NUMBER OF SCORES
    scorenum = 10000 ## DEBUGGING TO CHECK THE MESSAGE BOX WORKS
    if scorenum > 1000:
        outbox = QtGui.QMessageBox() ## CREATES THE MESSAGE BOX
        outbox.setWindowTitle("Set Analysis") ## SETS THE TITLE OF THE MESSAGE BOX
        outbox.setText("You are trying to analysis a set where you have more than 1000 scores.") ## SETS THE MAIN TEXT OF THE MESSAGE BOX
        outbox.setInformativeText("This may take some time to process, would you like to continue?") ## SETS ADDITIONAL TEXT OF THE MESSAG
        outbox.setIcon(QtGui.QMessageBox.Information) ## SETS THE ICON OF THE MESSAGE BOX
        outbox.setStandardButtons(QtGui.QMessageBox.Ok | QtGui.QMessageBox.Cancel) ## SETS THE BUTTONS FOR THE MESSAGE BOX
        choice = outbox.exec() ## EXECUTES THE MESSAGE BOX AND RETRIEVES THE BUTTON PRESSED
        if choice == QtGui.QMessageBox.Ok: ## CHECKS IF OK WAS PRESSED
            SetAnalysis.fetchresults(SetAnalysis_Window,self.playset,cur_user) ## PASSES THE NECESSARY PARAMETERS TO START FETCHING RESULT
            SetAnalysis_Window.show() ## SHOWS THE ANALYSIS WINDOW
            Results_Window.hide() ## HIDES THE RESULTS WINDOW
```

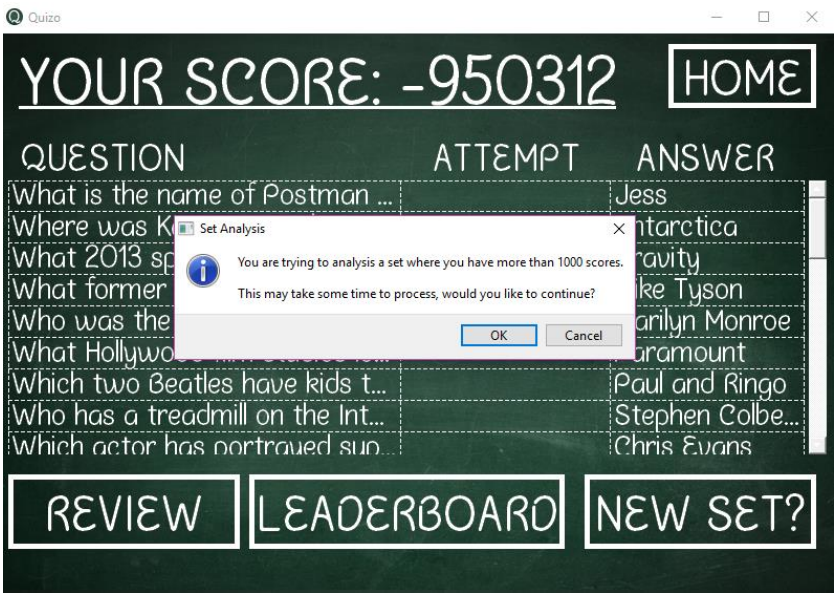
I have then also added a line of code for checking that the message box will appear correctly when the number of questions exceed 1000. The testing for both of these things can be seen below alongside some screenshots.



Once reaching the results screen, I pressed the "Review" button and stepped through the code until I reached the line shown below, fetching and setting the number of scores achieved by "woodingmp" on the "Entertainment" quiz to the "scorenum" variable, and it was successfully set to 48, as can be shown in this SQL window.

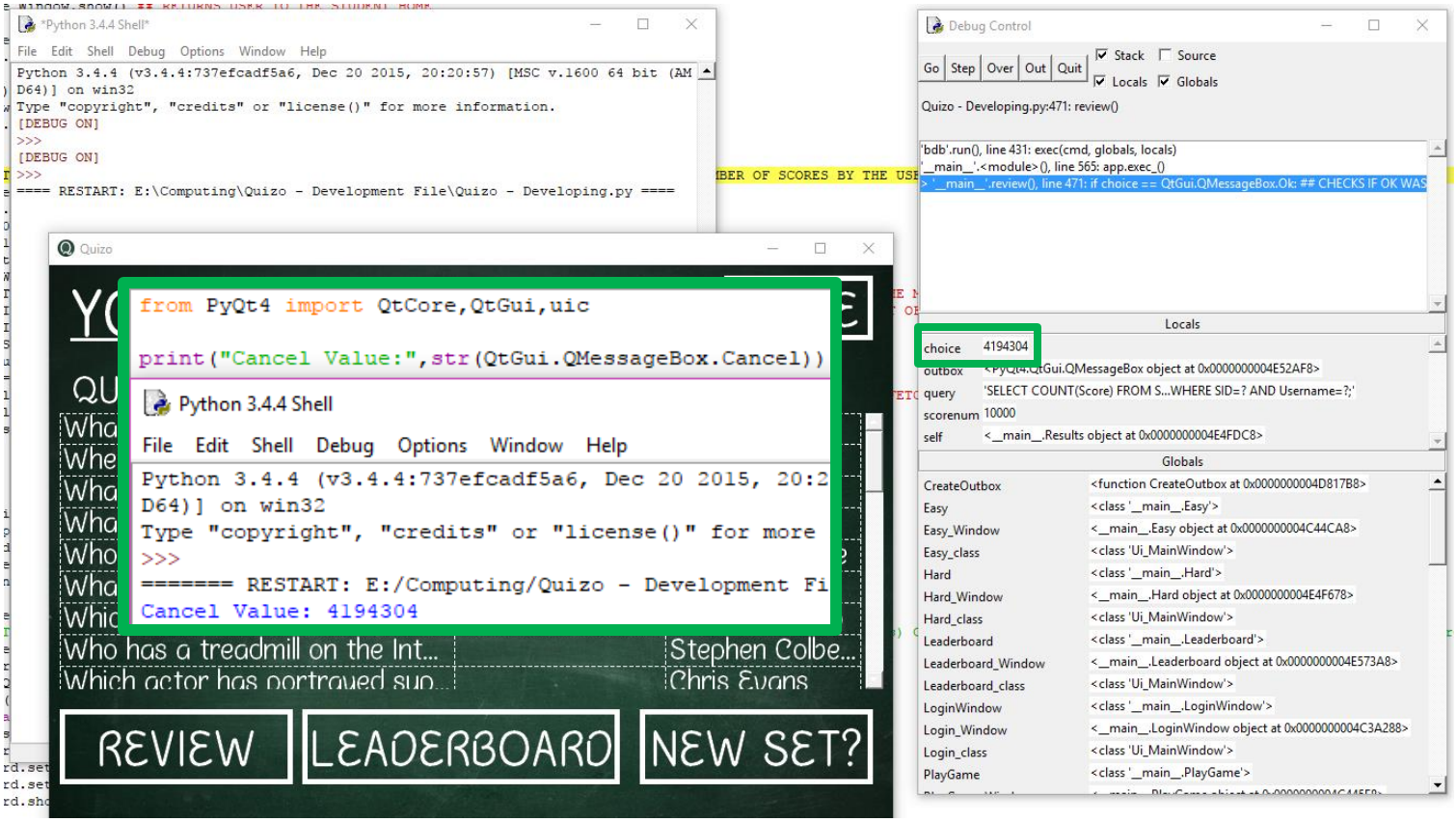


Stepping through the program another line shows that the program then successfully set the value of "scorenum" to 10000 in order for me to check whether the message box would appear correctly.

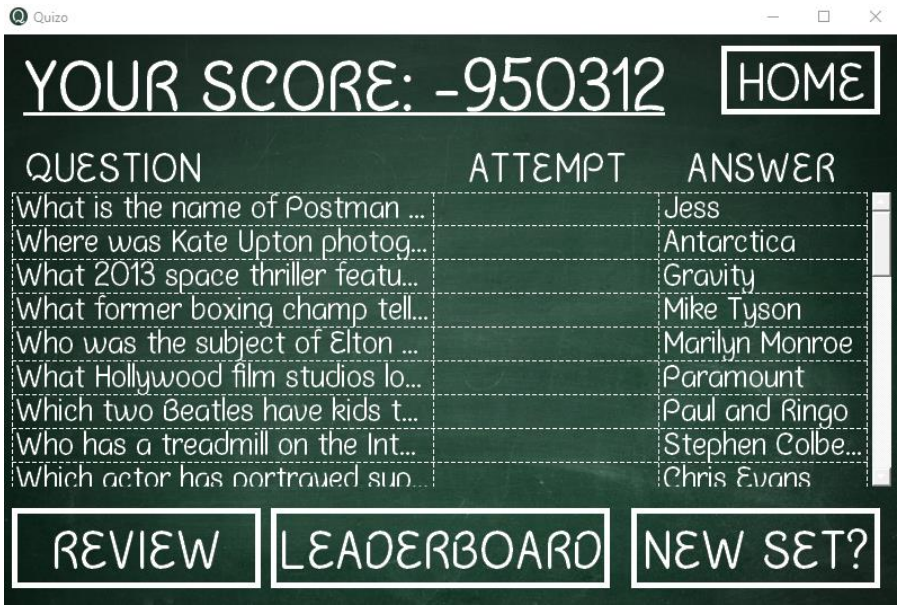


After this, the program then correctly displayed the message box asking if I wanted to continue.

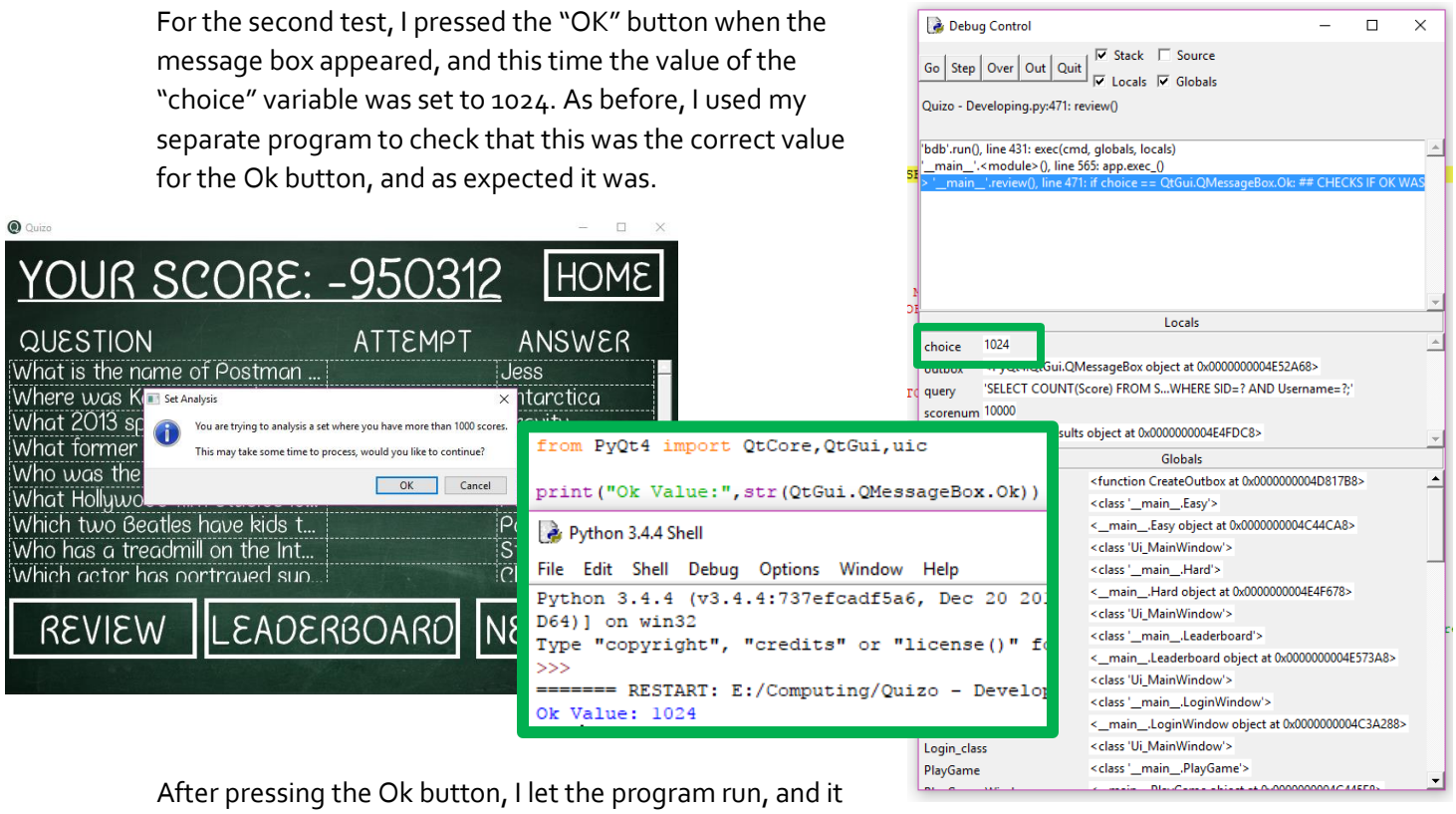
For the first test I decided that I would press the cancel button, and as can be seen below, the "choice" variable was given the value 4194304. To make sure that this is the correct value for the cancel button, I made a separate file just to print the value of "QtGui.QMessageBox.Cancel" and as can be seen, it was infact the correct value.



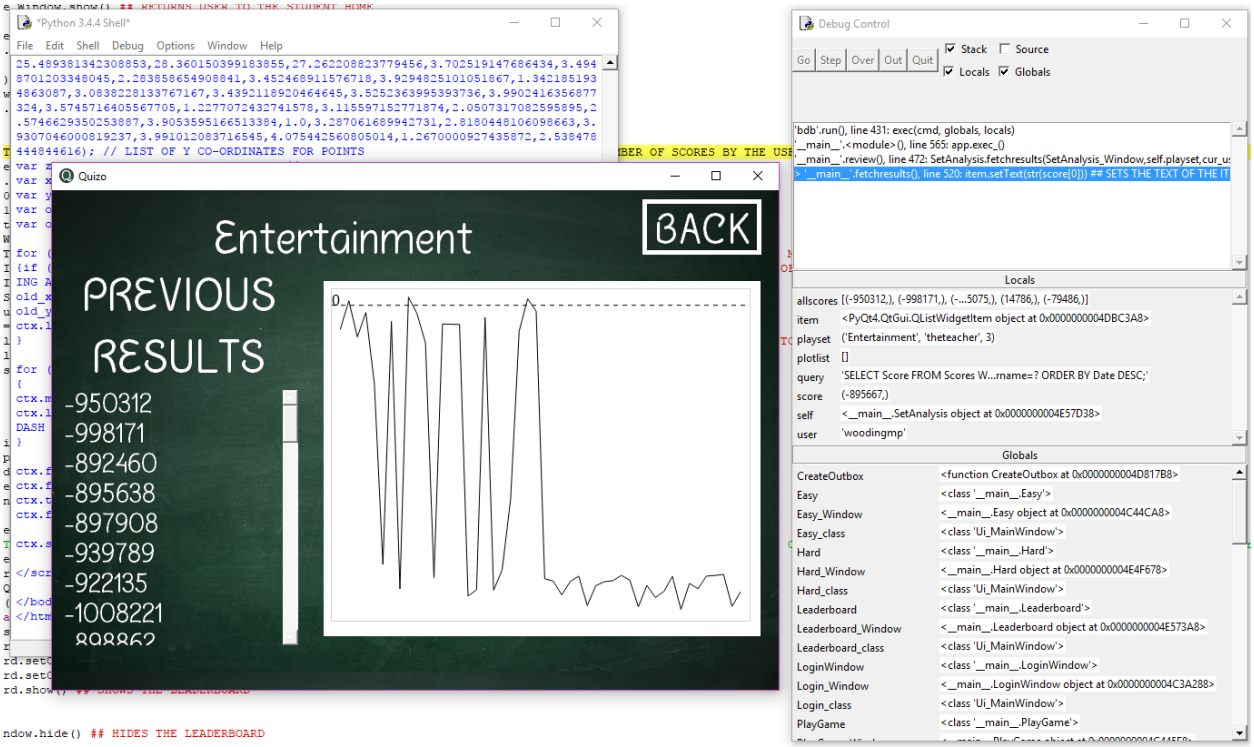
As can be seen, the program correctly just closed the message box when the cancel button was pressed.



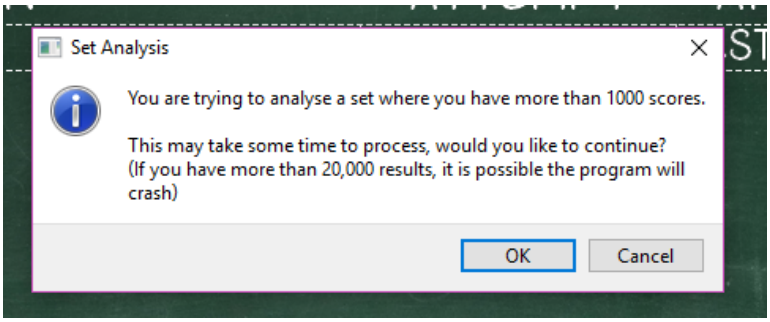
For the second test, I pressed the "OK" button when the message box appeared, and this time the value of the "choice" variable was set to 1024. As before, I used my separate program to check that this was the correct value for the Ok button, and as expected it was.



After pressing the Ok button, I let the program run, and it successfully came to the analysis screen and displayed the graph as expected.



Having just spot a spelling mistake in my message box, I thought it would also be a good idea to mention that too many scores, as in above 20,000 may crash the program, just in case anyone were to achieve that many.



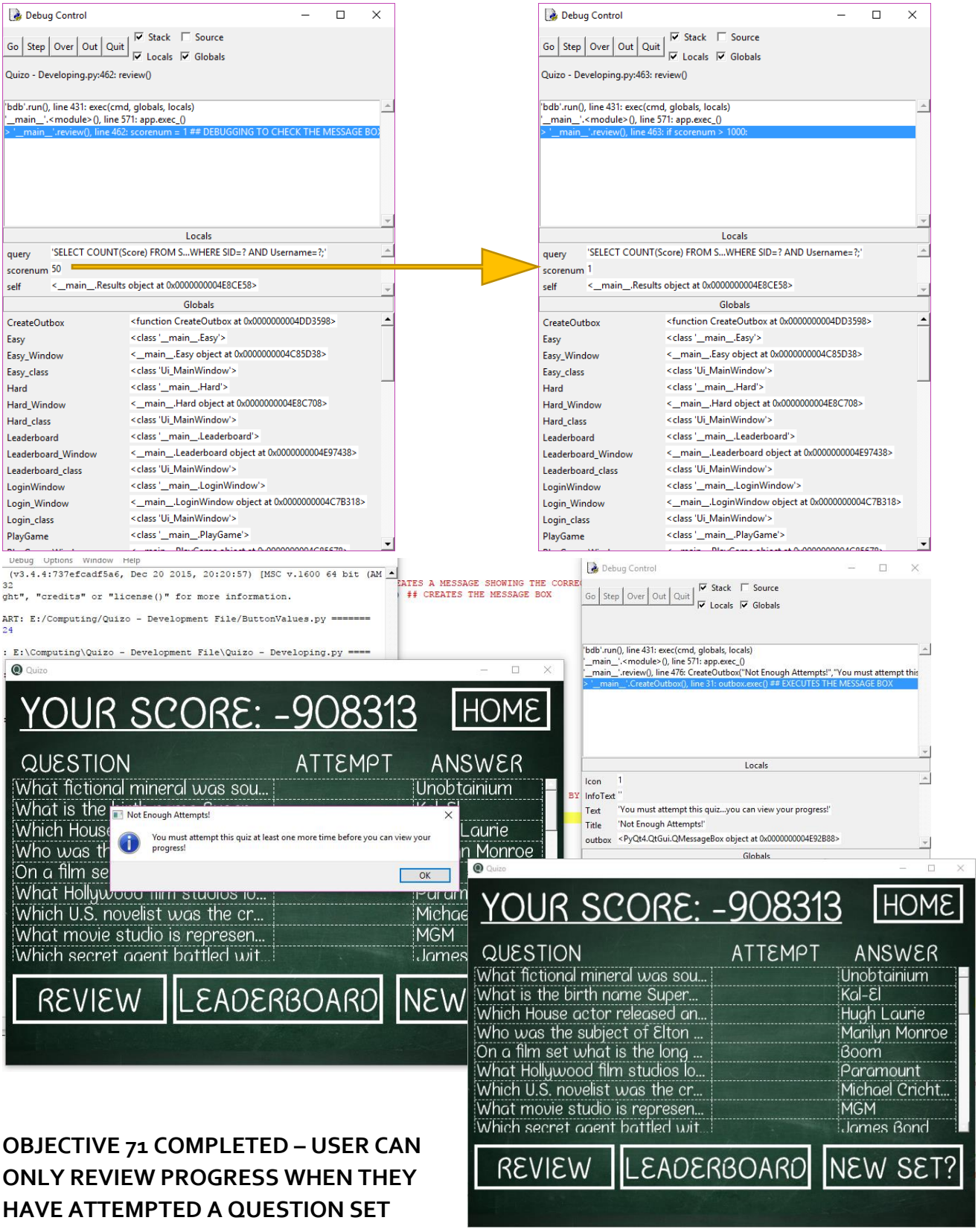
Although the crashing of the program is not ideal, the likelihood of any one user scoring more than 20,000 results is highly unlikely, let alone 1000 in order to reach this message. However, hopefully by putting this message box in place, if any user were to reach this many, they will have been thoroughly warned.

OBJECTIVES 40 AND 41 COMPLETED – USER IS ABLE TO VIEW A GRAPH OF ALL THEIR RESULTS IN A GIVEN SET THAT THEY HAVE COMPLETED, AND THIS GRAPH CAN SCALE TO ACCOMMODATE ALL THEIR RESULTS

Similar to how the program needs to warn users with too many scores about viewing graphs, it would be impossible to generate a graph with only one score ever recorded in the database. For this reason, I have added an extra if statement into my code that will check whether the user has only ever attempted the question set once, and if so, will display a message box that explains to them that they need to attempt the set again in order to review their progress. The code for this can be seen below, along with the breakpoint I have added in order to test this portion of the program, as I have set the value of "scorenum" to 1 instead of 10000, to check that this message box will appear after attempting a set only once.

```
def review(self):
    query = "SELECT COUNT(Score) FROM Scores WHERE SID=? AND Username=?;" ## QUERY THAT FETCHES THE NUMBER OF SCORES BY THE USER IN A SET
    cur.execute(query,(self.playset[2],cur_user,)) ## EXECUTES THE QUERY WITH THE CORRECT PARAMETERS
    scorenum = cur.fetchone()[0] ## FETCHES THE ACTUAL NUMBER OF SCORES
    scorenum = 1 ## DEBUGGING TO CHECK THE MESSAGE BOX WORKS
    if scorenum > 1000:
        outbox = QtGui.QMessageBox() ## CREATES THE MESSAGE BOX
        outbox.setWindowTitle("Set Analysis") ## SETS THE TITLE OF THE MESSAGE BOX
        outbox.setText("You are trying to analyse a set where you have more than 1000 scores.") ## SETS THE MAIN TEXT OF THE MESSAGE BOX
        outbox.setInformativeText("This may take some time to process, would you like to continue?\n(If you have more than 20,000 results, it is possible the prograr
        outbox.setIcon(QtGui.QMessageBox.Information) ## SETS THE ICON OF THE MESSAGE BOX
        outbox.setStandardButtons(QtGui.QMessageBox.Ok | QtGui.QMessageBox.Cancel) ## SETS THE BUTTONS FOR THE MESSAGE BOX
        choice = outbox.exec() ## EXECUTES THE MESSAGE BOX AND RETRIEVES THE BUTTON PRESSED
        if choice == QtGui.QMessageBox.Ok: ## CHECKS IF OK WAS PRESSED
            SetAnalysis.fetchresults(SetAnalysis_Window,self.playset,cur_user) ## PASSES THE NECESSARY PARAMETERS TO START FETCHING RESULTS FOR ANALYSIS
            SetAnalysis_Window.show() ## SHOWS THE ANALYSIS WINDOW
            Results_Window.hide() ## HIDES THE RESULTS WINDOW
    elif scorenum == 1: ## CHECKS IF USER HAS ONLY ATTEMPTED THE SET ONCE
        CreateOutbox("Not Enough Attempts!","You must attempt this quiz at least one more time before you can view your progress!","",QtGui.QMessageBox.Information)
    else:
       SetAnalysis.fetchresults(SetAnalysis_Window,self.playset,cur_user) ## PASSES THE NECESSARY PARAMETERS TO START FETCHING RESULTS FOR ANALYSIS
        SetAnalysis_Window.show() ## SHOWS THE ANALYSIS WINDOW
        Results_Window.hide() ## HIDES THE RESULTS WINDOW
```

As can be seen below, as I stepped through the program, the value of "scorenum" changed from 50 to 1 as expected, meaning that the program displayed the message box to the user, stating that they have to attempt the set again in order to review progress. The program then successfully closed the message box and remained on this window as hoped.



OBJECTIVE 71 COMPLETED – USER CAN ONLY REVIEW PROGRESS WHEN THEY HAVE ATTEMPTED A QUESTION SET MORE THAN ONCE

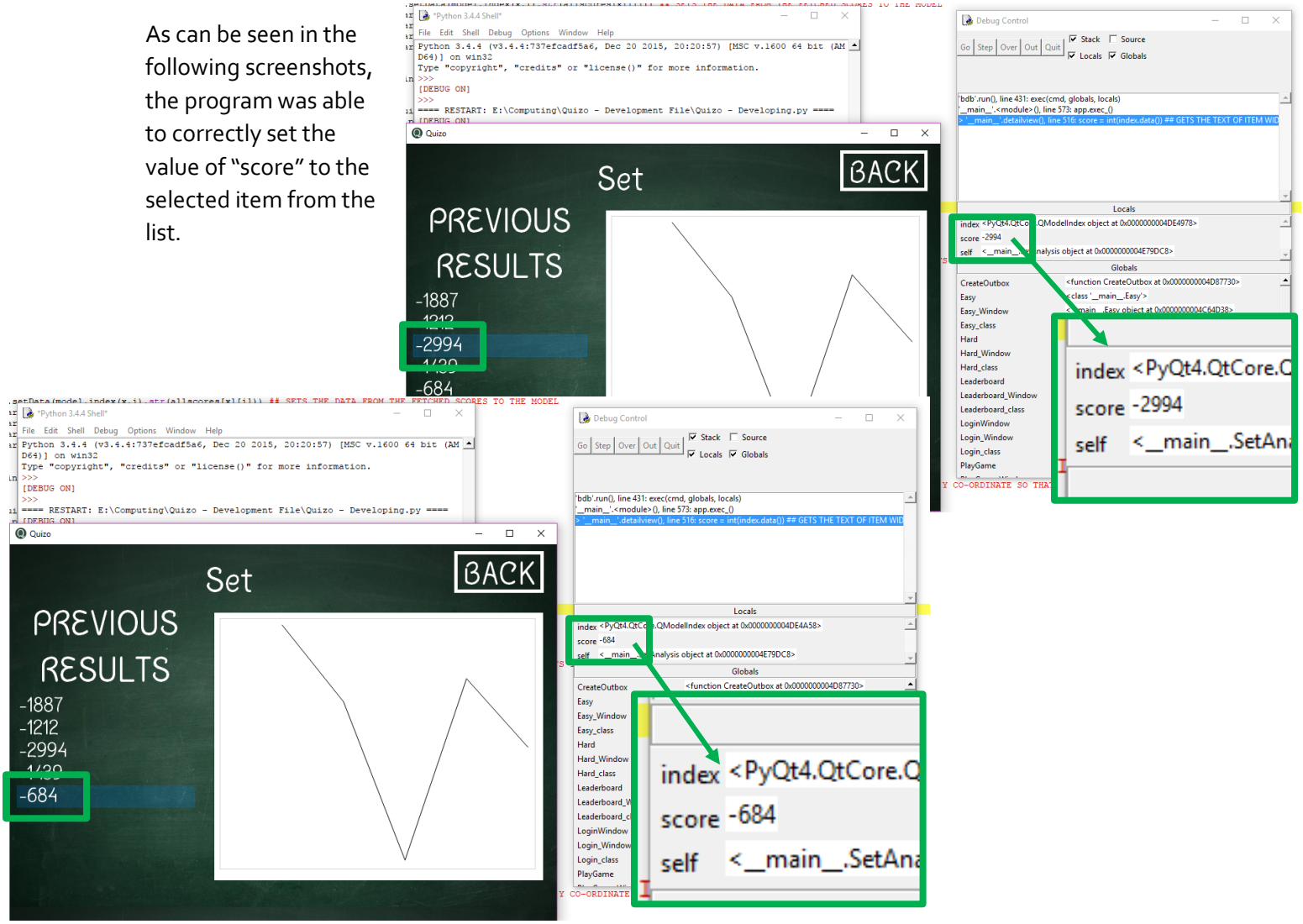
VIEWING INDIVIDUAL SCORES

While being able to view a graph is useful to see progress over time, the user should also be able to find a specific score on the graph from the list on the left of the window. I plan to implement this in the same way I have implemented the zero line: a horizontal dashed line across the graph at the appropriate y value.

In order to fetch the actual score of the selected score in the table, I will be using a clicked method for the table, and because the table is actually a list widget, I am able to pass the list widget item to the function. The code for this can be seen below, alongside a test showing that the correct scores are being set to the "score" variable in the debugger.

```
class SetAnalysis(QtGui.QMainWindow,SetAnalysis_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.setupUi(self)
        self.backbutton.clicked.connect(self.Back)
        self.resultstable.clicked.connect(self.detailview) ## PASSES SELECTED ITEM TO DETAILVIEW FUNCTION

    def detailview(self,index): ## INDEX IS THE ACTUAL ITEM WIDGET SELECTED
        score = int(index.data()) ## GETS THE TEXT OF ITEM WIDGET SELECTED (THE SCORE)
```

As can be seen in the following screenshots, the program was able to correctly set the value of "score" to the selected item from the list.



Now that the program is able to get the value of the item selected, it needs to be able to add the horizontal line onto the graph. In order to do this, I have adapted the code for writing the graph slightly. Now instead of all the drawing and fetching being in the same function, I have created a new "drawgraph" function which is now dedicated to the drawing part of making the graph. As shown below,it takes 6 arguments, so that it can take all the necessary values and lists to replace data in the HTML code.

```
def drawgraph(self,points,zero,xset,scoreline,score):
    file = open("GraphFile.txt","rt") ## OPENS THE GRAPH FILE
    contents = file.read() ## SAVES THE CONTENTS OF THE FILE TO A VARIABLE
    contents = contents.replace("POINTS-LIST-HERE",points) ## ADDS THE POINTSLIST TO THE HTML FILE
    contents = contents.replace("ZERO-COORDS-HERE",str(zero)) ## ADDS THE ZEROLINE CO-ORDINATE TO THE HTML FILE
    contents = contents.replace("X-OFFSET-HERE",str(xset)) ## ADDS THE XOFFSET TO THE HTML FILE
    self.webView.setHtml(contents) ## SETS THE VIEW OF THE WEB VIEWER IN THE WINDOW
```

In addition to this, I have also added the "self" prefix to the following variables: xoffset, largest, smallest, FLOOR, CEILING, zeroline, and pointslist. The new code for the "fetchresults" function can be seen below, showing these changes.

```
def fetchresults(self,playset,user):
    self.setlabel.setText(playset[0]) ## SETS LABEL TEXT TO THE NAME OF THE SET
    self.resultstable.clear() ## CLEARS THE CURRENT LIST
    query = "SELECT Score FROM Scores WHERE SID=? AND Username=? ORDER BY Date DESC;" ## FETCHES ALL THE USER'S RESULTS
    cur.execute(query,(playset[2],user,)) ## EXECUTES THE QUERY
    allscores = cur.fetchall() ## FETCHES THE DATA

    plotlist = [] ## CONTAINS THE Y CO-ORDINATES OF POINTS TO BE PLOTTED

    for score in allscores: ## LOOPS THROUGH EACH SCORE
        item = QtGui.QListWidgetItem() ## CREATES A NEW ITEM IN THE LIST
        item.setText(str(score[0])) ## SETS THE TEXT OF THE ITEM TO THE SCORE
        self.resultstable.addItem(item) ## ADDS THE ITEM TO THE LIST WIDGET
    for score in reversed(allscores):
       plotlist.append(score[0]) ## ADDS THE CURRENT SCORE TO THE PLOTLIST

    self.xoffset = (450/len(plotlist)) ## GETS THE HORIZONTAL SPACING BETWEEN EACH POINT

    self.largest = max(plotlist) ## FINDS THE HIGHEST SCORE
    self.smallest = min(plotlist) ## FINDS THE LOWEST SCORE
    self.FLOOR = 1 ## MINIMUM Y CO-ORDINATE OF POINT - CAPITALISED FOR CONSTANT
    self.CEILING = 28.5 ## MAXIMUM Y CO-ORDINATE OF POINT - CAPITALISED FOR CONSTANT

    self.zeroline = (((self.CEILING-self.FLOOR)*(0-self.smallest))/(self.largest-self.smallest))+self.FLOOR ## FINDS THE
    for i in range(0,len(plotlist)): ## LOOPS THROUGH EACH SCORE IN THE PLOTLIST
        plotlist[i] = (((self.CEILING-self.FLOOR)*(plotlist[i]-self.smallest))/(self.largest-self.smallest))+self.FLOOR

    self.pointslist=",".join([str(x) for x in plotlist]) ## JOINS TOGETHER ALL THE POINTS INTO ONE STRING SO IT'S FORMAT
    self.drawgraph(self.pointslist,self.zeroline,self.xoffset,0,0)
```

I have made these changes so that the "drawgraph" can be called from multiple functions, without the need for redoing all the processing to get all the co-ordinates for the points on the graph, and to find the zero line location. Additonally, I have called the "drawgraph" function at the end of this function but with two zeros for the final two parameters. This is so that when the initial graph is plotted, there is no extra horizontal line drawn in.

```
def detailview(self,index): ## INDEX IS THE ACTUAL ITEM WIDGET SELECTED
    score = int(index.data()) ## GETS THE TEXT OF ITEM WIDGET SELECTED (THE SCORE)
    scoreline = (((self.CEILING-self.FLOOR)*(score-self.smallest))/(self.largest-self.smallest))+self.FLOOR ## FINDS THE Y CO-ORDINATE OF THE SCORE LINE
    self.drawgraph(self.pointslist,self.zeroline,self.xoffset,scoreline,score)
```

Above is the code for the "detailview" function, which generates the y value for the selected score so that it can be drawn onto the graph. So that the exact same graph is displayed but with this extra line ontop, I have called the "drawgraph" function with the same parameters as before, but with the final two actually representing the new horizontal line to draw.

In order for the new line to be drawn, I have had to edit the HTML code so that there is a place to insert the new data and draw the line, which can be seen below, alongside the additional code in the "drawgraph" function.

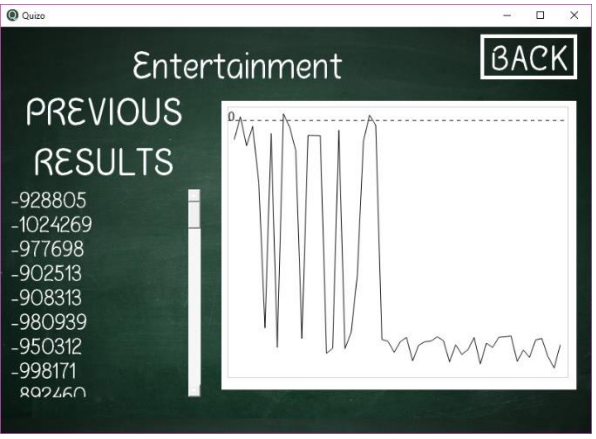
The additional code in the "drawgraph" function just checks whether or not the function is being called from the "fetchresults" or "detailview" function by checking to see whether the value of score and scoreline is 0, which will be the case if coming from "fetchresults" function.

In the event that the user gets a score of zero, and its position on the graph is at y=0, then the user won't need the extra line anyway as there will always be a zero line on the graph.

Below is a test to check whether or not the code is working as intended to draw the new line.

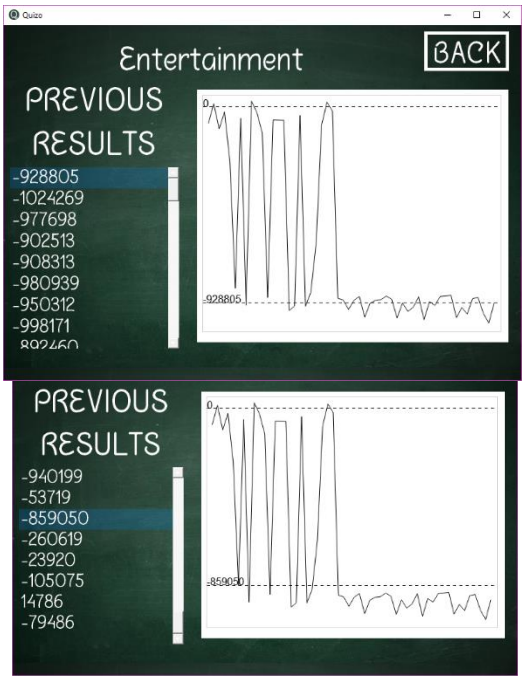
```
var canvas = document.getElementById("myCanvas"); // CREATES
var ctx = canvas.getContext("2d"); // SETS 2D CANVAS
var points=new Array(POINTS-LIST-HERE); // LIST OF Y CO-ORD
var zerolocation=ZERO-COORDS-HERE; // SETS THE Y CO-ORDINATE
// SCORE-COORDS-HERE
// SCORE-HERE
var xoffset=X-OFFSET-HERE; // SETS HORIZONTAL SPACING BETWEE
var yoffset=25; // SETS VERTICAL STRETCH OF EACH POINT
var old_x=xoffset; // SETS STARTING X VALUE
var old_y=canvas.height-yoffset*points[0]; // SETS STARTING

for (var i = 0; i < points.length; i++)  // LOOPS THROUGH EA
{if (i>0){ctx.moveTo(old_x,old_y);} // IF THERE ARE POINTS
old_x=xoffset+xoffset*(i); // CALCULATES X CO-ORDINATE OF PO
old_y=canvas.height-yoffset*points[i]*.5; // CALCULATES Y CO
ctx.lineTo(old_x,old_y); // DRAWS A LINE FROM PREVIOUS POINT
}

for (var x = 0; x < 47; x++) // LOOPS THROUGH NUMBER OF DASH
{
ctx.moveTo(x*10,canvas.height-yoffset*zerolocation*.5); //
ctx.lineTo((x*10)+5,canvas.height-yoffset*zerolocation*.5);

// DRAW-SCORE-LINE
}

ctx.font = "16px Arial"; // SETS FONT FOR GRAPH
ctx.fillStyle = "black"; // SETS FILL STYLE
ctx.textAlign = "left"; // ALLIGNS TEXT LEFT
ctx.fillText(0,0,(canvas.height-yoffset*zerolocation*.5));
// WRITE-SCORE
```

```
def drawgraph(self,points,zero,xset,scoreline,score):
    file = open("GraphFile.txt","rt") ## OPENS THE GRAPH FILE
    contents = file.read() ## SAVES THE CONTENTS OF THE FILE TO A VARIABLE
    contents = contents.replace("POINTS-LIST-HERE",points) ## ADDS THE POINTSLIST TO THE HTML FILE
    contents = contents.replace("ZERO-COORDS-HERE",str(zero)) ## ADDS THE ZEROLINE CO-ORDINATE TO THE HTML FILE
    contents = contents.replace("X-OFFSET-HERE",str(xset)) ## ADDS THE XOFFSET TO THE HTML FILE
    if score != 0 and scoreline != 0:
        contents = contents.replace("// SCORE-COORDS-HERE","var scorelocation="+str(scoreline)+";") ## INSERTS THE Y-VALUE FOR THE SCORE
        contents = contents.replace("// SCORE-HERE","var score="+str(score)+";") ## INSERTS THE SCORE VALUE
        contents = contents.replace("// DRAW-SCORE-LINE","ctx.moveTo(x*10,canvas.height-yoffset*scorelocation*.5); \n ctx.lineTo((x*10)+5,canvas.height-yoffset*scorelocation*.5);") ## INSERTS THE CODE FOR DRAWING THE LINE
        contents = contents.replace("// WRITE-SCORE","ctx.fillText(score,0,(canvas.height-yoffset*scorelocation*.5));") ## INSERTS THE CODE FOR WRITING THE SCORE
    self.webView.setHtml(contents) ## SETS THE VIEW OF THE WEB VIEWER IN THE WINDOW
```



After loading the analysis screen, the graph was displayed correctly, as seen on the left.

After clicking some of the scores from the list, their horizontal line was drawn onto the graph immediately as can be seen to the right.

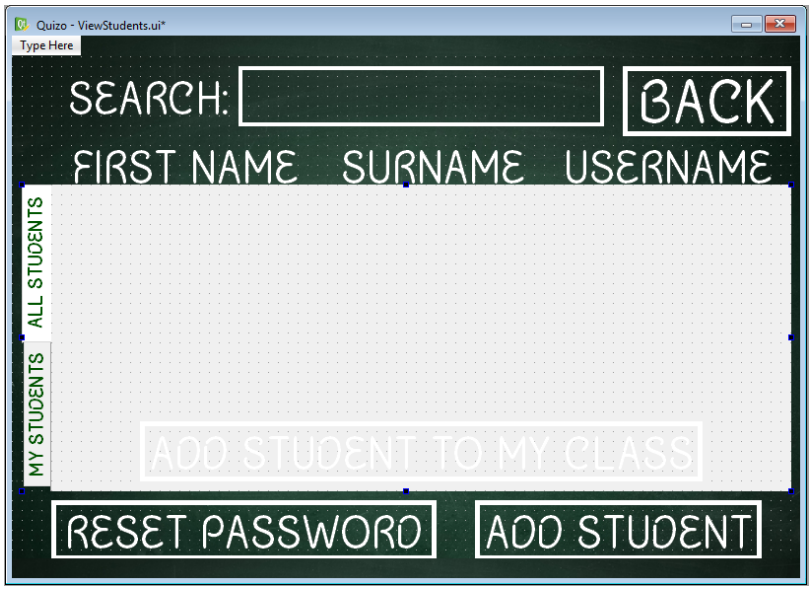


Seeing as this code is working as expected, I have decided that this window is feature complete, and I can move onto creating the next window for the program.

OBJECTIVE 42 COMPLETED – USERS CAN SELECT A SCORE AND VIEW IT ON THE GRAPH

CREATING THE VIEW STUDENTS WINDOW

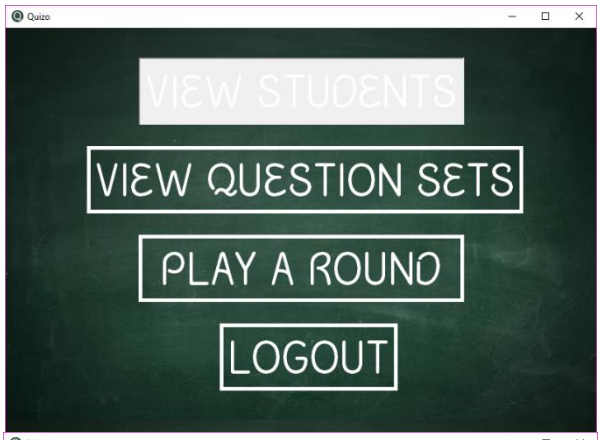
Now that the quiz portion of the program is complete, and allows users to track their progress after completing a quiz, I need to add the ability for teachers to view the progress of their students. In order to achieve this, I first need to create a window that allows teachers to view all students within the database, as well as perform some basic functions.

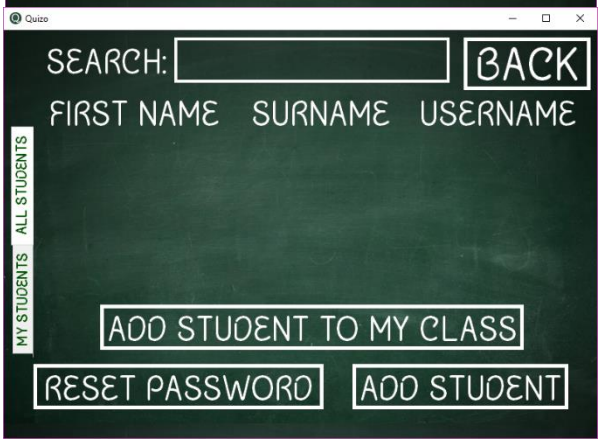


To the right is the design I have created in QtDesigner based on the initial design that I created. It has two tabs, one for looking at all students, and the other for looking at just the current user's students. There are also buttons, allowing the teacher to reset users' passwords, add or remove students to and from their class, and register new students to the program.

I have created a new class for this window, and have written the necessary code for it to be opened in the program when a teacher presses the "View Students" button. This code, and a demonstration of this working can be seen below.

```
class ViewStudents(QtGui.QMainWindow,ViewStudents_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.setupUi(self)
        self.backbutton.clicked.connect(self.Back)

    def Back(self):
        TeacherHome_Window.show()
        ViewStudents_Window.hide()
```





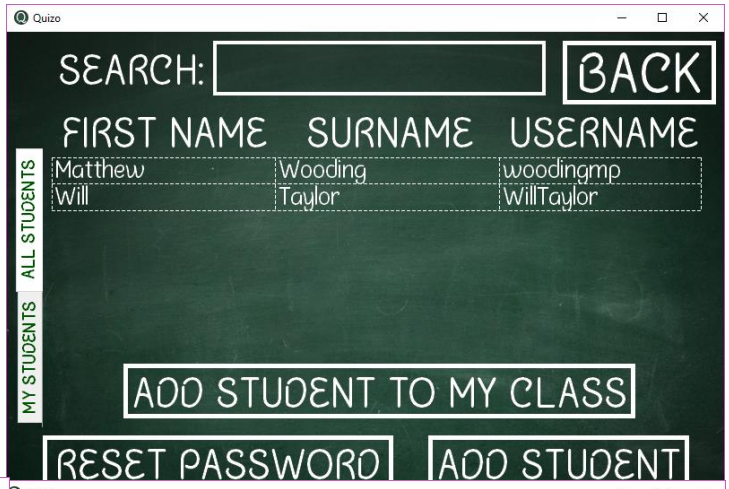
As I have decided to use tabs and two tables within this window that are almost identical, I will create a function that can be used by both tables, that will take the following parameters:
- The current table
- The teacher's username
- The entered search term

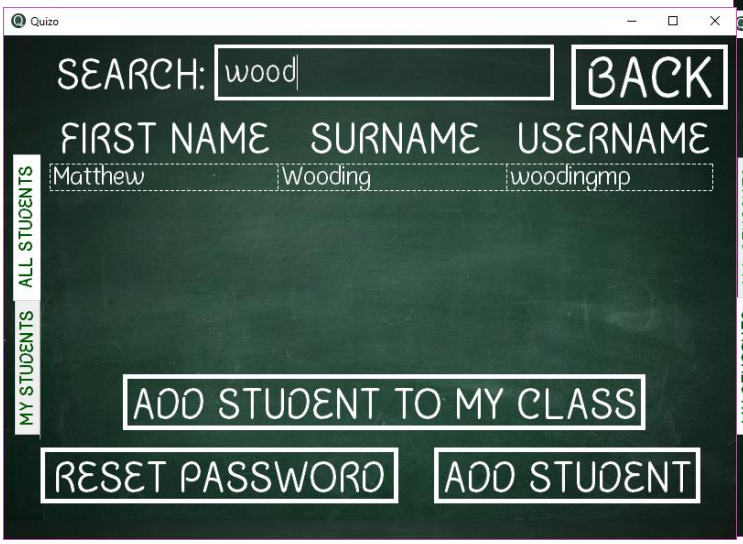
When this window is opened, the program will automatically display the tables, and the whenever the search bar is used, the tables will update accordingly.

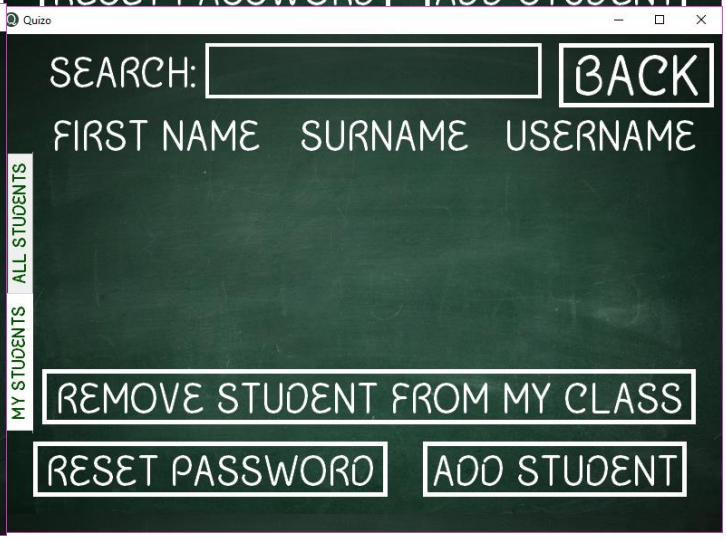
The code for how I have decided to do this can be seen below, alongside some screenshots of what happened in my initial testing.

```
class ViewStudents(QtGui.QMainWindow,ViewStudents_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.setupUi(self)
        self.backbutton.clicked.connect(self.Back)
        self.searchedit.textChanged.connect(self.search)

    def viewtable(self,table,searchword,teacher):
        teacher = "%"+teacher+"%" ## FORMATS THE TEACHER USERNAME FOR USE IN THE SQL STATEMENT
        searchword = "%"+searchword+"%" ## FORMATS THE SEARCH WORD FOR USE IN THE SQL STATEMENT
        query = '''SELECT Students.FirstName,Students.Surname,Students.Username FROM Students INNER JOIN STLinks on Students.Username=STLinks.SUser WHERE STLinks.TUser LIKE ?
and (Students.Username LIKE ? or Students.FirstName LIKE ? or Students.Surname LIKE ?);''' ## QUERY TO FETCH ALL STUDENTS FROM DATABASE
        cur.execute(query,(teacher,searchword,searchword,searchword,)) ## EXECUTES QUERY
        allstudents = cur.fetchall() ## FETCHES DATA FROM DATABASE
        model = QtGui.QStandardItemModel(len(allstudents),3) ## CREATES A MODEL TO PUT THE DATA IN
        for row in range(0,len(allstudents)): ## LOOPS THROUGH EACH SET
            for column in range(0,3): ## LOOPS THROUGH SET NAME AND AUTHOR FOR THE CURRENT SET
                model.setData(model.index(row,column),str(allstudents[row][column])) ## SETS THE MODEL DATA ACCORDING TO THE FETCHED DATA
        table.setModel(model) ## SETS THE MODEL TO THE TABLE VIEW
        tempwidth = [250,250,225] ## COLUMN WIDTHS FOR THE TABLE
        for i in range(3): ## LOOPS THROUGH EACH COLUMN
            table.setColumnWidth(i,tempwidth[i]) ## SETS THE WIDTH OF THE COLUMN
        table.show() ## SHOWS THE TABLE

    def search(self):
        self.viewtable(self.mystudstable,self.searchedit.text(),cur_user) ## DISPLAYS THE MY STUDENTS TABLE
        self.viewtable(self.allstudstable,self.searchedit.text(),"") ## DISPLAYS THE ALL STUDENTS TABLE

    def Back(self):
        TeacherHome_Window.show()
        ViewStudents_Window.hide()
```

As can be seen in these screenshots, when the window is initially opened, it correctly displays the all students table with the results it has fetched. Additionally, when logged in as "MathsTeacher" the "mystudents" table correctly displays no data, as this teacher does not have any students linked. Below is also a screenshot displaying that the search function works as intended.







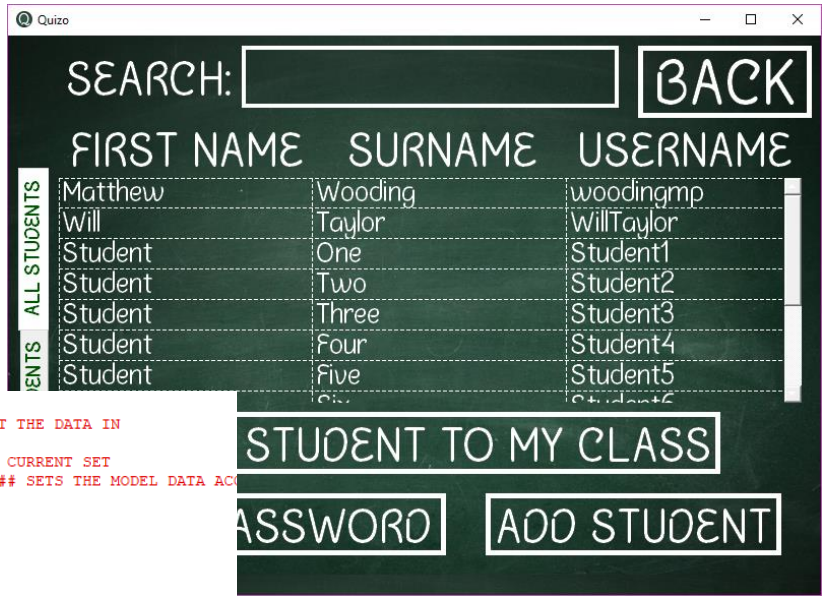
Although it appears as though the program is working correctly, it is not actually displaying all students in the all students table. When I was creating the leaderboard, I added ten additional student accounts, however because they are not taught by any teachers, they are not appearing in either table.

In order to fix this issue, I have added an if statement that will check which table is currently being processed. Depending on which table is currently being displayed, a different SQL statement will be used.

```
    def viewtable(self,table,searchword,teacher):
        searchword = "%"+searchword+"%" ## FORMATS THE SEARCH WORD FOR USE IN THE SQL STATEMENT
        if teacher == "": ## CHECKS IF TABLE IS ALL STUDENTS OR MY STUDENTS
            query = 'SELECT FirstName,Surname,Username FROM Students WHERE Students.Username LIKE ? or Students.FirstName LIKE ? or Students.Surname LIKE ?;' ## QUERY TO FETCH ALL STUDENTS
            cur.execute(query,(searchword,searchword,searchword,)) ## EXECUTES QUERY
        else:
            teacher = "%"+teacher+"%" ## FORMATS THE TEACHER USERNAME FOR USE IN THE SQL STATEMENT
            query = '''SELECT Students.FirstName,Students.Surname,Students.Username FROM Students INNER JOIN STLinks on Students.Username=STLinks.SUser WHERE STLinks.TUser LIKE ?
and (Students.Username LIKE ? or Students.FirstName LIKE ? or Students.Surname LIKE ?);''' ## QUERY TO FETCH ALL STUDENTS FROM DATABASE
            cur.execute(query,(teacher,searchword,searchword,searchword,searchword,)) ## EXECUTES QUERY
        students = cur.fetchall() ## FETCHES DATA FROM DATABASE
        model = QtGui.QStandardItemModel(len(students),3) ## CREATES A MODEL TO PUT THE DATA IN
        for row in range(0,len(students)): ## LOOPS THROUGH EACH SET
            for column in range(0,3): ## LOOPS THROUGH SET NAME AND AUTHOR FOR THE CURRENT SET
                model.setData(model.index(row,column),str(students[row][column])) ## SETS THE MODEL DATA ACCORDING TO THE FETCHED DATA
        table.setModel(model) ## SETS THE MODEL TO THE TABLE VIEW
        tempwidth = [250,250,225] ## COLUMN WIDTHS FOR THE TABLE
        for i in range(3): ## LOOPS THROUGH EACH COLUMN
            table.setColumnWidth(i,tempwidth[i]) ## SETS THE WIDTH OF THE COLUMN
        table.show() ## SHOWS THE TABLE
```

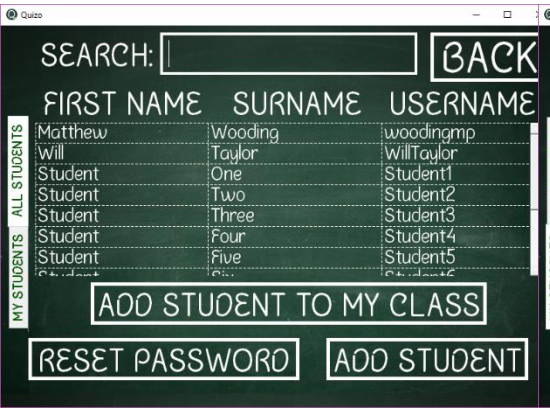
Although this achieves the desired outcome, I have realised that when it comes to selecting students from the tables, I will need a list for each table, holding all the rows, meaning that I have to make two separate variables for the fetched data.

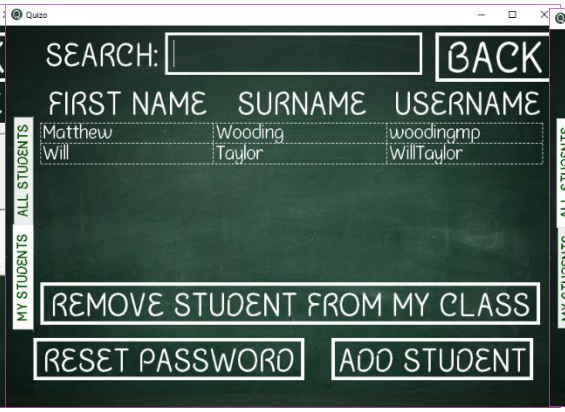
I have tidied up the code a bit and added the new variables below.

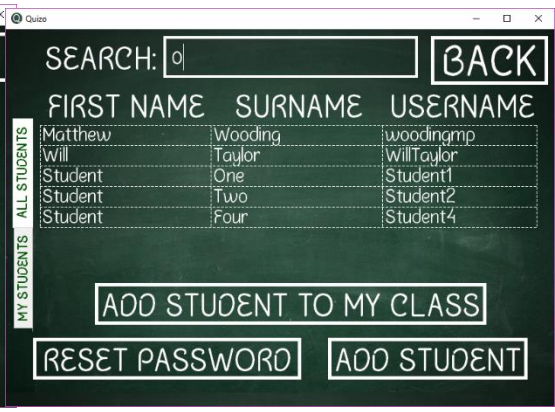



```
    def viewtable(self,table,students):
        model = QtGui.QStandardItemModel(len(students),3) ## CREATES A MODEL TO PUT THE DATA IN
        for row in range(0,len(students)): ## LOOPS THROUGH EACH SET
            for column in range(0,3): ## LOOPS THROUGH SET NAME AND AUTHOR FOR THE CURRENT SET
                model.setData(model.index(row,column),str(students[row][column])) ## SETS THE MODEL DATA AC
        table.setModel(model) ## SETS THE MODEL TO THE TABLE VIEW
        tempwidth = [250,250,225] ## COLUMN WIDTHS FOR THE TABLE
        for i in range(3): ## LOOPS THROUGH EACH COLUMN
            table.setColumnWidth(i,tempwidth[i]) ## SETS THE WIDTH OF THE COLUMN
        table.show() ## SHOWS THE TABLE

    def search(self):
        searchword = "%"+self.searchedit.text()+"%" ## FORMATS THE SEARCH WORD FOR USE IN THE SQL STATEMENT
        query = 'SELECT FirstName,Surname,Username FROM Students WHERE Students.Username LIKE ? or Students
        cur.execute(query,(searchword,searchword,searchword,)) ## EXECUTES QUERY
        self.allstudents = cur.fetchall() ## FETCHES DATA FROM THE DATABASE
        query = '''SELECT Students.FirstName,Students.Surname,Students.Username FROM Students INNER JOIN ST
and (Students.Username LIKE ? or Students.FirstName LIKE ? or Students.Surname LIKE ?);''' ## QUERY TO FETC
        cur.execute(query,(cur_user,searchword,searchword,searchword,)) ## EXECUTES QUERY
        self.mystudents = cur.fetchall() ## FETCHES DATA FROM THE DATABASE
        self.viewtable(self.mystudstable,self.mystudents) ## DISPLAYS THE MY STUDENTS TABLE
        self.viewtable(self.allstudstable,self.allstudents) ## DISPLAYS THE ALL STUDENTS TABLE
```

Below is a short test showing that this new code works as before.







OBJECTIVES 30,31, AND 32 COMPLETED – PROGRAM SUCCESSFULLY FETCHES AND DISPLAYS ALL STUDENTS FROM DATABASE, AND CAN FILTER BY TEACHER

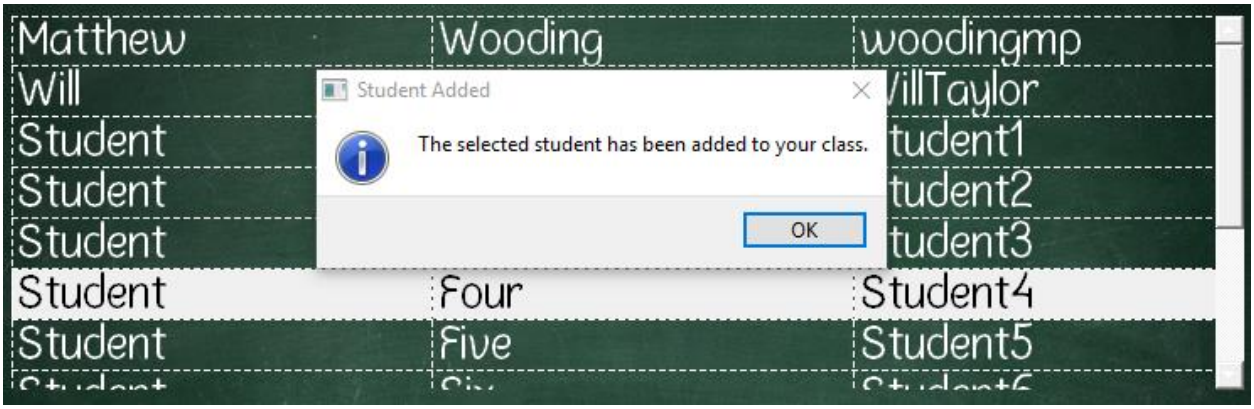
ADDING AND REMOVING STUDENTS TO AND FROM CLASSES

As in most secondary schools, the Royal Grammar School teaches boys from year 7 to year 13, and every year it is common for teachers to be allocated new classes to teach, filled with different pupils. Due to this changing of classes, my program needs to be able to allow teachers to add and remove students to and from their classes at will.

In order to support this, I have written some code for both the adding button, and the removing button.

For the adding students button, I have written some code that will fetch the current student selected if there is one (if not an error message will pop up to the user), and the program will then check if this student is already in this teacher's class. If the student is already in the class, the teacher will be notified of this, and if not, a new record will be added to the STLinks table in the database, adding the chosen student to the teacher's class. The code for this can be seen below, alongside tests proving it works as intended.

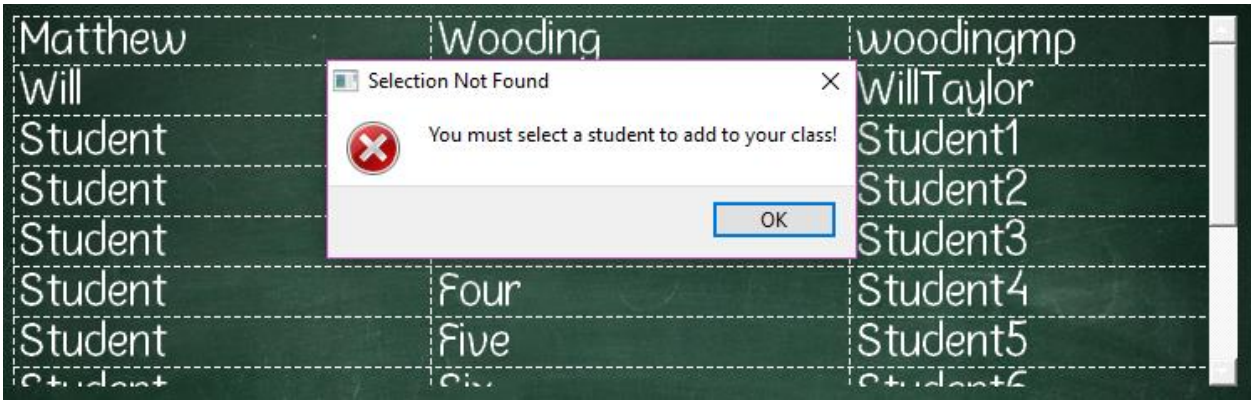
```
def addtoclass(self):
    selection = self.allstudstable.selectionModel().selectedRows() ## FETCHES THE SELECTED ROWS FROM THE TABLE
    if selection == []: ## CHECKS IF ANY ROW HAS BEEN SELECTED
        CreateOutbox("Selection Not Found","You must select a student to add to your class!","",QtGui.QMessageBox.Critical)
    else:
        selectedrow = selection[0].row() ## FINDS INDEX OF SELECTED ROW
        studentdetails = self.allstudents[selectedrow] ## GETS THE DETAILS OF THE SELECTED STUDENT
        query = 'SELECT * FROM STLinks WHERE SUser=? AND TUser=?;' ## TRIES TO FIND EXISTING LINK IN TABLE
        cur.execute(query,(studentdetails[2],cur_user,)) ## EXECUTES QUERY
        link = cur.fetchone() ## FETCHES ANY DATA FROM DATABASE
        if link != None: ## CHECKS IF A LINK WAS FOUND
            CreateOutbox("Conflict Found","This student is already in your class.","",QtGui.QMessageBox.Information)
        else:
            query = 'INSERT INTO STLinks (TUser,SUser) VALUES (?,?);' ## CREATES NEW LINK
            cur.execute(query,(cur_user,studentdetails[2],)) ## EXECUTES QUERY
            con.commit() ## WRITES CHANGES TO DATABASE
            CreateOutbox("Student Added","The selected student has been added to your class.","",QtGui.QMessageBox.Information)
            self.search() ## UPDATES THE TABLES
```



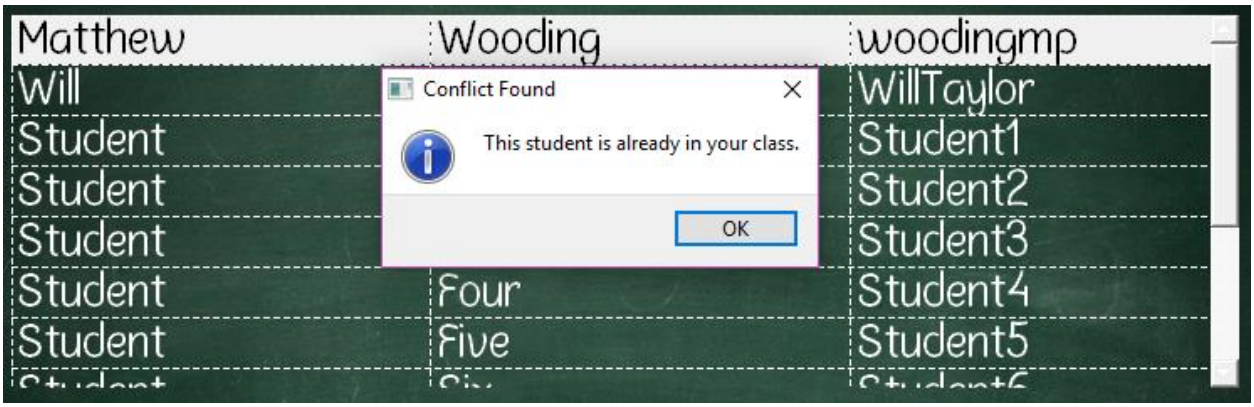
Test 1 – Adding a student to a class – Successful – Student Four was successfully added to the class of "theteacher" class as can be seen in the database browser to the side.

Table: STLinks

	TUser	SUser
	Filter	Filter
1	theteacher	woodingmp
2	theteacher	WillTaylor
3	theteacher	Student4



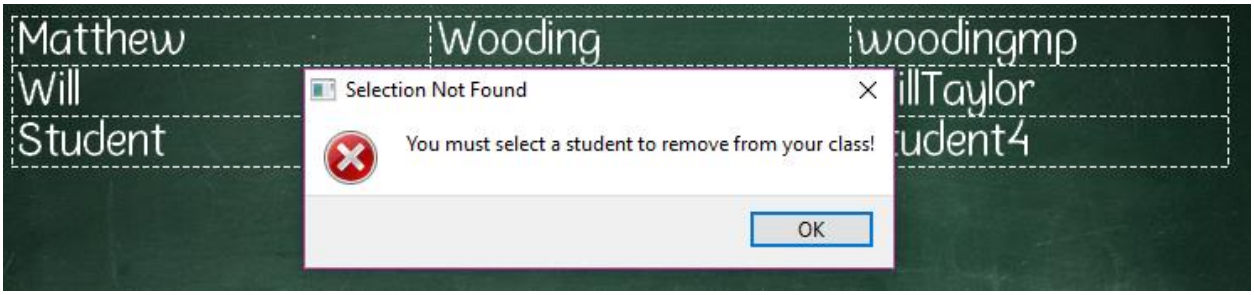
Test 2 – Trying to add a student without selecting a row – Successful – When I loaded the window again, and had no row selected, upon pressing the add button, the program successfully told me that no row was selected.



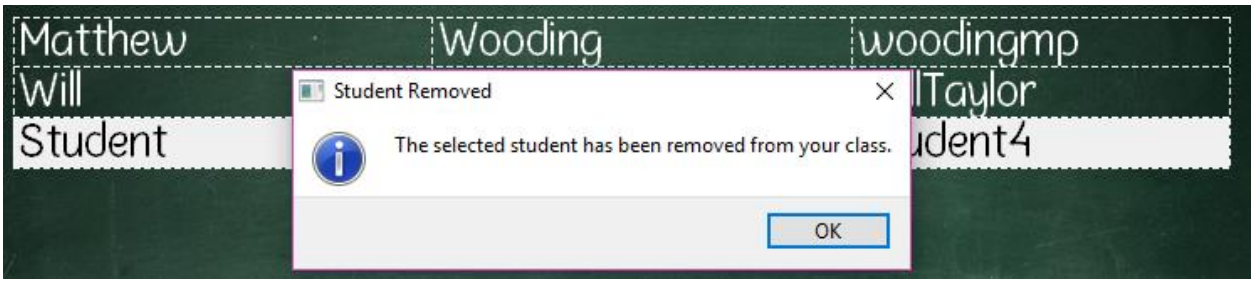
Test 3 – Trying to add a student who is already in the class – Successful – When trying to add "woodingmp" to the class of "theteacher" the program recognised that the account was already in the class and informed the user of that.

Now that the program is able to successfully add students to a teacher's class, I will implement the ability to remove student's from a class. This process is essentially the same as adding, with a few tweaks, so I will be using the same code and editing it slightly. This code, followed by some tests can be seen below.

```
def removefromclass(self):
    selection = self.mystudstable.selectionModel().selectedRows() ## FETCHES THE SELECTED ROWS FROM THE TABLE
    if selection == []: ## CHECKS IF ANY ROW HAS BEEN SELECTED
        CreateOutbox("Selection Not Found","You must select a student to remove from your class!","",QtGui.QMessageBox.Critical)
    else:
        selectedrow = selection[0].row() ## FINDS INDEX OF SELECTED ROW
        studentdetails = self.mystudents[selectedrow] ## GETS THE DETAILS OF THE SELECTED STUDENT
        query = 'DELETE FROM STLinks WHERE SUser=? AND TUser=?;' ## DELETES ANY LINK WITH THE SELECTED STUDENT AND TEACHER
        cur.execute(query,(studentdetails[2],cur_user,)) ## EXECUTES QUERY
        con.commit() ## WRITES CHANGES TO DATABASE
        CreateOutbox("Student Removed","The selected student has been removed from your class.","",QtGui.QMessageBox.Information)
        self.search() ## UPDATES THE TABLES
```



Test 1 – Trying to remove a student without selecting a row – Successful – Upon loading the window and pressing the remove button, the program successfully told me that no row had been selected.



Test 2 – Removing a student from a class – Successful – When "Student4" was selected and the remove button was pressed, the program successfully removed the link between this student and "theteacher". This can be seen to the side, in the database browser.

Table: STLinks

	TUser	SUser
	Filter	Filter
1	theteacher	woodingmp
2	theteacher	WillTaylor

OBJECTIVE 33 COMPLETED – TEACHERS CAN ADD OR REMOVE STUDENTS TO AND FROM THEIR CLASSES WITHIN THE PROGRAM

RESETTING PASSWORDS

When a student forgets their password, they need to be able to reset it so that they can continue to use the program without needing to have a new account created. To prevent students being able to reset anyone's password, this process can only be carried out by teachers.

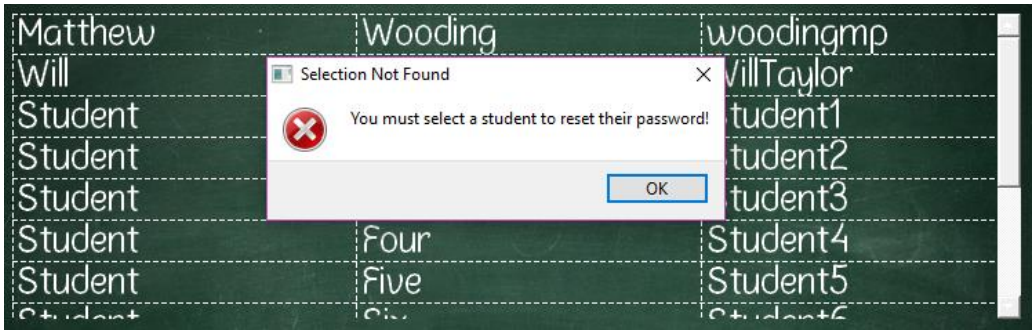
In order to reset passwords, the program will need a standardised password that the user's password will be set to when reset. When the user next tries to login with this password, they will be prompted to change their password to something else. For this, I have decided on the password "Quiz0123" as it is easy to remember and fairly short.

Due to all the fetching of details in a number of functions in this class, I have created a new function that returns the details of any student that is selected, depending on which tab is currently being viewed.

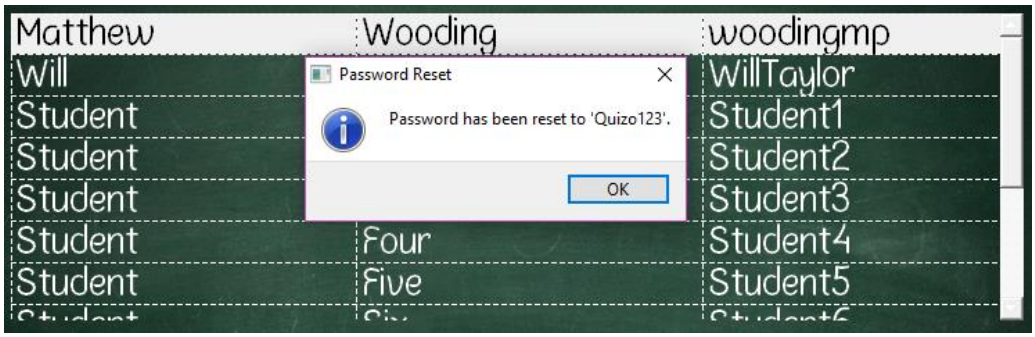
```
def findselection(self):
    studentdetails = [] ## NO STUDENT SELECTED
    if self.studentTabs.currentIndex() == 0: ## CHECKS WHICH TAB IS CURRENTLY BEING VIEWED
        selection = self.allstudstable.selectionModel().selectedRows() ## FINDS SELECTED ROWS IN ALL TABLE
        if selection != []: ## CHECKS IF ANYTHING WAS SELECTED
            selectedrow = selection[0].row() ## FINDS INDEX OF SELECTED ROW
            studentdetails = self.allstudents[selectedrow] ## GETS DETAILS OF SELECTED STUDENT
    else:
        selection = self.mystudstable.selectionModel().selectedRows() ## FINDS SELECTED ROWS IN MY TABLE
        if selection != []: ## CHECKS IF ANYTHING WAS SELECTED
            selectedrow = selection[0].row() ## FINDS INDEX OF SELECTED ROW
            studentdetails = self.mystudents[selectedrow] ## GETS DETAILS OF SELECTED STUDENT
    return studentdetails ## RETURNS THE DETAILS OF THE SELECTED STUDENT, OR AN EMPTY LIST
```

Now that I have added this function to my program, I have written the code for resetting passwords, which can be seen below alongside some tests.

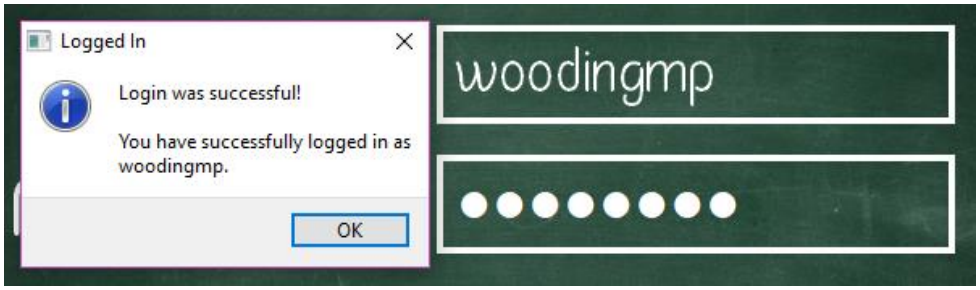
```
def resetpass(self):
    studentdetails = self.findselection() ## FIND THE SELECTED STUDENT'S DETAILS
    if studentdetails == []: ## CHECKS IF A STUDENT WAS SELECTED
        CreateOutbox("Selection Not Found","You must select a student to reset their password!","",QtGui.QMessageBox.Critical)
    else:
        password = "Quizo123" ## SETS TEMPORARY PASSWORD IN PLAIN TEXT
        password = hashlib.sha256(password.encode()).hexdigest() ## HASHES PASSWORD FOR DATABASE
        query = 'REPLACE INTO Students (Username,Password,FirstName,Surname) VALUES (?,?,?,?);' ## QUERY TO CHANGE PASSWORD
        cur.execute(query,(studentdetails[2],password,studentdetails[0],studentdetails[1],)) ## EXECUTES THE QUERY
        con.commit() ## WRITES CHAGNES TO DATABASE
        CreateOutbox("Password Reset","Password has been reset to 'Quizo123'.","",QtGui.QMessageBox.Information)
```



Test 1 – Trying to reset a password with no student selected – Successful – When no student was selected, the program successfully prompted me to select a student to reset their password.



Test 2 – Resetting a student's password – Successful – To check that the passwords were being reset correctly, I reset the password of "woodingmp", and the program returned an output stating that the reset was successful. To ensure it was, I logged out of the teaching account, and tried to log into the "woodingmp" account using the default password "Quizo123". As expected, the login was successful.



OBJECTIVE 34 COMPLETED – TEACHERS CAN RESET STUDENT'S PASSWORDS TO THE DEFAULT "Quizo123"

ALLOWING STUDENTS TO CHANGE THEIR PASSWORDS AFTER RESET

Now that teachers are able to reset any student's password, the students need to be able to change their default password to a stronger one that they will remember. In order to do this, I have created a new window in QtDesigner for changing a user's password when they login using the default one.

I have created a new class for this window, and all it does is check that the user's new password is strong, and that they confirm it by typing in the same password twice. If both these conditions are met, the password will be changed.

In order to reach this window, I have added some code to the login screen, so that users that successfully log on with the default password are directed here before going to their home screen.

The code for this new class, and the changes made to the login class can be seen below.

```
hashedattempt = hashlib.sha256(password.encode()).hexdigest() ## HASHES USER'S ATTEMPT
if hashedattempt == tempuser[1]: ## CHECKS TO SEE IF THE HASHED ATTEMPT IS THE SAME AS THE HASHED PASSWORD
    cur_user = username ## SETS CURRENT USER TO THE GIVEN USERNAME
    if hashedattempt == hashlib.sha256("Quizo123".encode()).hexdigest(): ## CHECKS IF USER IS LOGGING IN WITH DEFAULT PASSWORD
        ResetPassword_Window.show() ## SHOWS THE RESET PASSWORD WINDOW
```

Changes made to the login class – Program checks if the password used to login is the default one.

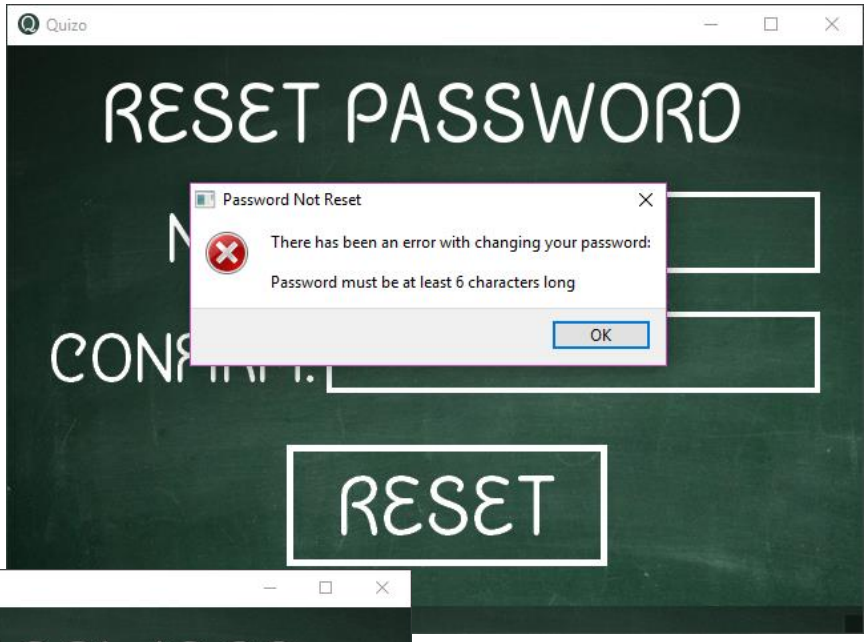
```
class ResetPassword(QtGui.QMainWindow,ResetPassword_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.setupUi(self)
        self.resetbutton.clicked.connect(self.reset)

    def validatepassword(self,password):
        errors = []
        if len(password) > 5: ## CHECKS THAT PASSWORD IS LONGER THAN FIVE CHARACTERS
            uppercase = 0 ## RESETS NUMBER OF UPPERCASE CHARACTERS FOUND IN PASSWORD
            lowercase = 0 ## RESETS NUMBER OF LOWERCASE CHARACTERS FOUND IN PASSWORD
            if password.isalnum() == False: ## CHECKS IF STRING IS MADE UP OF ONLY ALPHANUMERICAL CHARACTERS
                errors.append("Password contains invalid character(s)")
            for char in password: ## STARTS FOR LOOP WHICH GOES THROUGH EACH CHARACTER IN PASSWORD
                if char.isupper() == True: ## CHECKS IF CHARACTER IS UPPERCASE
                    uppercase += 1 ## INCREMENTS NUMBER OF UPPERCASE LETTERS
                elif char.islower() == True: ## CHECKS IF CHARACTER IS LOWERCASE
                    lowercase += 1 ## INCREMENTS OF LOWERCASE LETTERS
            if (uppercase == 0) or (lowercase == 0): ## CHECKS TO SEE IF PASSWORD DOESN'T REACH REQUIREMENTS
                errors.append("Password must contain at least 1 uppercase and 1 lowercase letter")
        else:
            errors.append("Password must be at least 6 characters long")
        return errors ##  RETURNS THE ERROR LIST

    def reset(self):
        new = self.newedit.text() ## GETS NEW PASSWORD
        confirm = self.confirmedit.text() ## GETS CONFIRMED PASSWORD
        errors = self.validatepassword(new) ## GETS ANY ERRORS FROM VALIDATING PASSWORD
        if new != confirm: ## CHECKS IF NEW AND CONFIRM ARE IDENTICAL
            errors.append("Both passwords must be identical to confirm")
        if errors == []: ## CHECKS IF THERE ARE NO ERRORS
            query = 'SELECT FirstName,Surname,Username FROM Students WHERE Username=?;' ## QUERY TO FETCH DETAILS OF STUDENT
            cur.execute(query,(cur_user,)) ## EXECUTES QUERY
            details = cur.fetchall()[0] ## FETCHES DETAILS
            password = hashlib.sha256(new.encode()).hexdigest() ## HASHES NEW PASSWORD
            query = 'REPLACE INTO Students (Username,Password,FirstName,Surname) VALUES (?,?,?,?);' ## QUERY TO REPLACE PASSWORD
            cur.execute(query,(details[2],password,details[0],details[1],)) ## EXECUTES QUERY
            con.commit() ## WRITES CHANGES TO DATABASE
            CreateOutbox("Password Reset","Your password has been successfully changed.","",QtGui.QMessageBox.Information)
            ResetPassword_Window.hide() ## HIDES THE RESET WINDOW
            Login_Window.hide() ## HIDES THE LOGIN WINDOW
            StudentHome_Window.show() ## SHOWS THE STUDENT HOME WINDOW
        else:
            errormessage = "\n".join(errors) ## CONVERTS LIST OF ERRORS INTO SINGLE STRING
            CreateOutbox("Password Not Reset","There has been an error with changing your password:",errormessage,QtGui.QMessageBox.Critical)
```

In order to test that this new class in my program is working correctly, I have attempted to login to the program using "woodingmp", the same account for which I reset the password in previous tests.

Logging in with this account and the default password "Quizo123" led me to the reset password screen as expected.

Once on this screen, I decided to test that all the password validation was working correctly, and that I was sent to the correct home screen after changing my password.

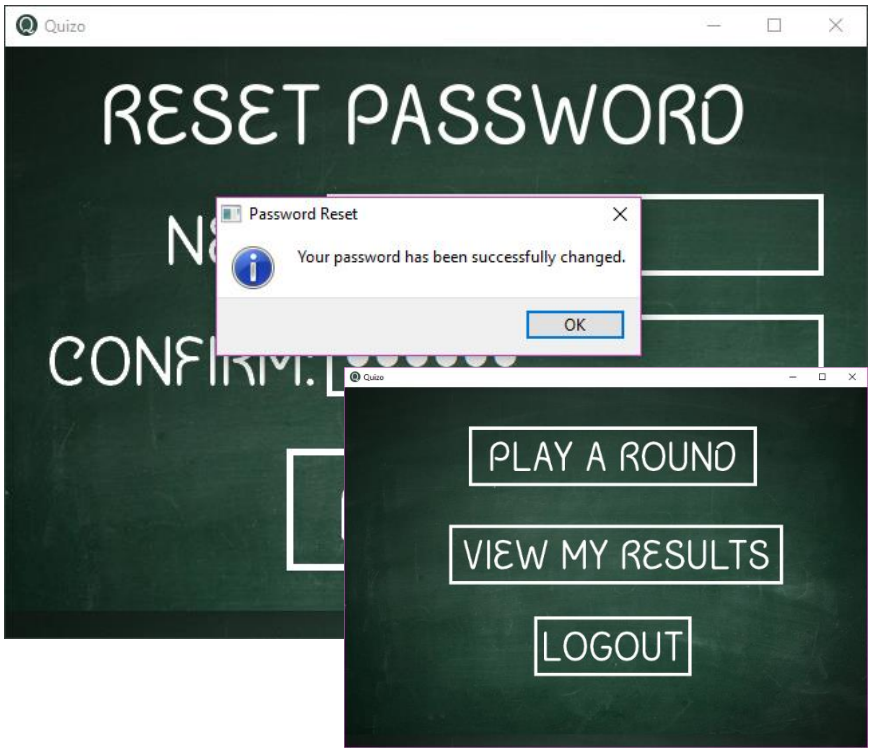
Test 1 – Entering less than 6 characters for a password – Successful – When I left both fields blank and pressed the reset button, the program informed me that passwords had to be at least 6 characters long.

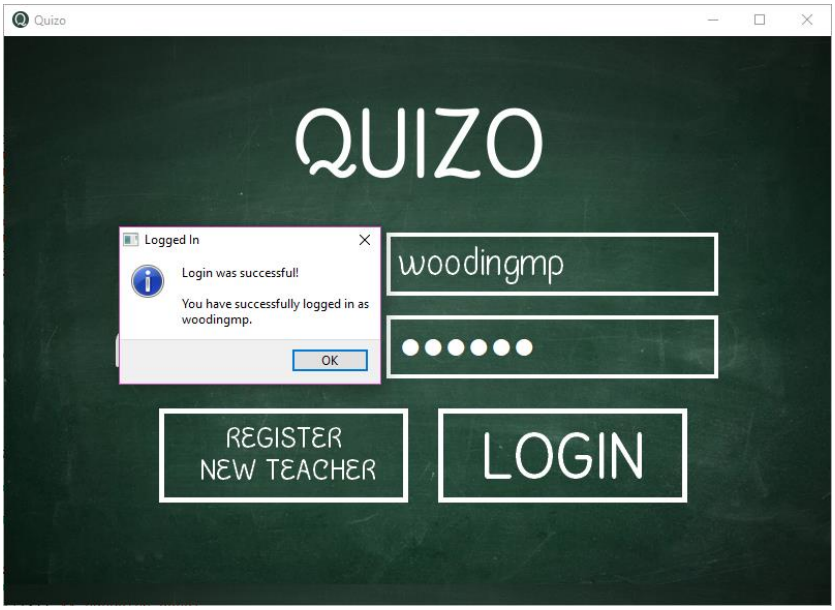


Test 2 – Entering a password without uppercase letters – Successful – To test this validation I used "abcdefg" as the password in both fields, and the program successfully informed me that passwords need at least one uppercase and one lowercase letter.

Test 3 – Entering different passwords – Successful – For this test I entered "Abcdefg" into the first field, and "abcdefg" into the second field. The program then successfully checked and informed me that the fields had to be identical to confirm the password change.

Test 4 – Entering identical valid passwords – Successful – After entering the same password "Matt27" into both fields, the programs correctly changed my password and moved me to the student home screen.





Test 5 – Check password change – Successful – To check that the password was definitely changed, I logged out of the program and tried to log back in as "woodingmp". Instead of using the default password, I tried logging in with the new password "Matt27" and as seen below, the program successfully recognised my login details and gave me access to the program.

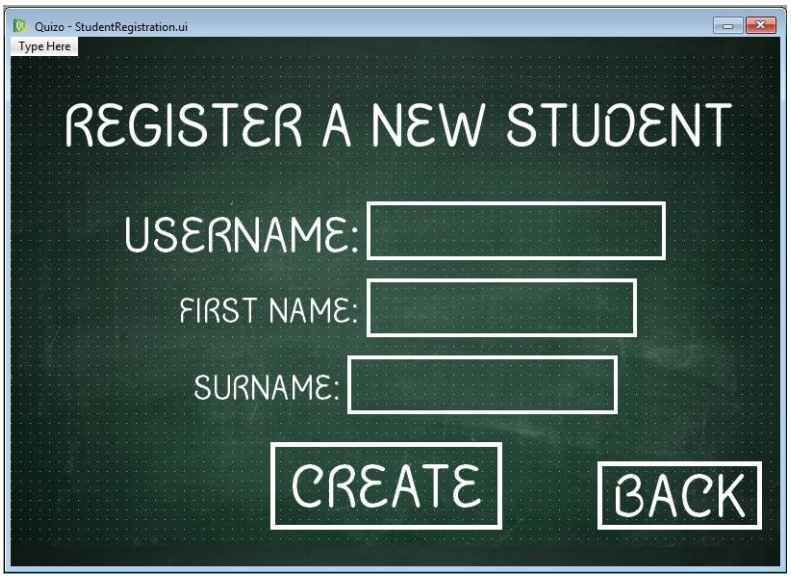
OBJECTIVES 17 AND 18 COMPLETED – WHEN LOGGING IN, PROGRAM CAN DETECT USERS WHO HAVE HAD THEIR PASSWORD RESET AND ALLOWS THEM TO CHANGE IT

ADDING NEW STUDENTS

When new boys join the RGS, new accounts will need to be added to the database for any that wish to use the program. Instead of having to go into the database itself and do this, the program needs to be able to let teachers add students to the system.

Adding a student is similar to adding a teacher, except there are fewer fields to check for validation:

- username must be between 3 and 15 characters, alphanumerical, and can't already be in use by another user
- First name must be made up of letters only
- Surname must be made up of letters, and can contain up to one dash for double barrels



I have created the window for adding students in QtDesigner according to my original design, and I have added a new class to my program for this window to be implemented.

I have added the necessary code for checking all fields are valid, and adding the account to the database if this is the case. This code, along with the navigation code I have added to the view students class can be seen below.

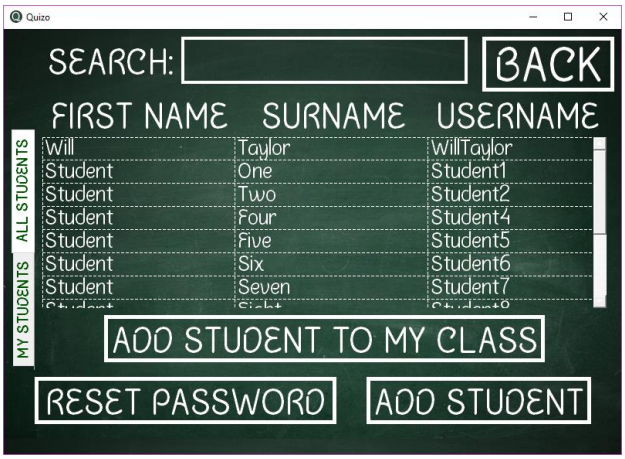
```
def add(self):
    StudentRegistration_Window.show() ## SHOWS THE STUDENT REGISTRATION WINDOW
    ViewStudents_Window.hide() ## HIDES THE VIEW STUDENTS WINDOW
```

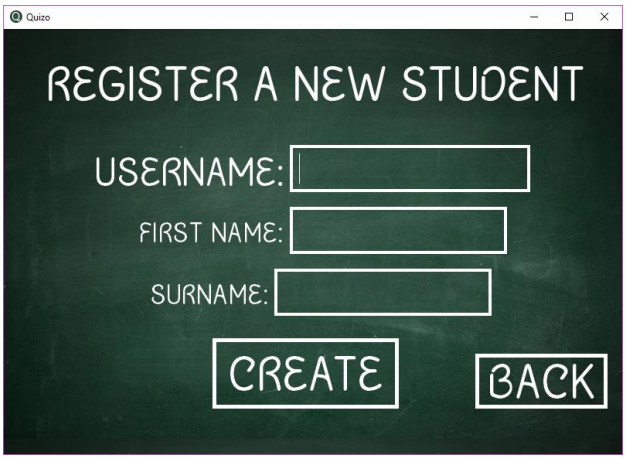
This is the code inside the view students window that allows teachers to get to the student registration screen.

```
def addstudent(self):
    username = self.useredit.text() ## FETCHES CONTENTS OF USERNAME BOX
    firstname = self.firstedit.text() ## FETCHES CONTENTS OF FIRSTNAME BOX
    surname = self.suredit.text() ## FETCHES CONTENTS OF SURNAME BOX
    errors = [] ## RESETS THE ERRORS WHEN ADDING A NEW ACCOUNT

    if len(username) > 2: ## CHECKS THAT THE USERNAME IS LONGER THAN 2 CHARACTERS, AS USERNAME MUST BE AT LEAST 3 CHARACTERS LONG
        if username.isalnum() == False: ## CHECKS IF STRING IS MADE UP OF ONLY ALPHANUMERICAL CHARACTERS
            errors.append("Username contains invalid character(s)")
        else:
            tables = ["Teachers","Students"] ## LIST CONTAINING TABLE NAMES TO BE SEARCHED
            for x in tables: ## STARTS A FOR LOOP FOR SEARCHING THE TABLES
                query = "SELECT Username FROM "+x+" WHERE Username='"+username+"';" ## CREATES THE QUERY FOR SEARCHING THE CURRENT TABLE
                cur.execute(query) ## EXECUTES THE QUERY
                temp = cur.fetchall() ## FETCHES ANY DATA FROM THE DATABASE THAT WAS SELECTED
                if temp!= []: ## CHECKS IF ANY DATA WAS FETCHED
                    errors.append("Username already taken")
                    break ## BREAKS OUT OF THE LOOP IF THE USERNAME HAS BEEN FOUND
    else:
        errors.append("Username must be at least 3 characters long")

    if firstname != "": ## CHECKS IF FIRSTNAME FIELD IS EMPTY
        if firstname.isalpha() == False: ## CHECKS IF FIRSTNAME IS MADE UP OF ONLY LETTERS
            errors.append("First name contains invalid character(s)")
        else:
            firstname = firstname.title()
    else:
        errors.append("First name must be filled in")

    if surname != "":
        dashes = 0 ## RESETS NUMBER OF DASHES IN SURNAME
        for char in surname: ## STARTS FOR LOOP WHICH GOES THROUGH EACH CHARACTER IN SURNAME
            if char == "-": ## CHECKS IF CHARACTER IS A DASH
                dashes += 1 ## INCREMENTS THE NUMBER OF DASHES IN THE SURNAME
            elif char.isalpha() == False: ## CHECKS IF CHARACTER IS NOT A LETTER
                        errors.append("Surname contains invalid character(s)")
                        break ## PREVENTS THE ERROR FROM BEING PRINTED MORE THAN ONCE
        if dashes > 1: ## CHECKS IF USER HAS MORE THAN ONE DASH IN THEIR SURNAME
            errors.append("Surnames can contain a maximum of one dash")
        else:
            surname = surname.title() ## CAPITALISES THE FIRST LETTER OF EACH WORD
            if dashes == 1: ## CHECKS IF USER HAS A DOUBLE BARRELLED SURNAME
                barrels = surname.split("-") ## SPLITS SURNAME INTO SEPARATE BARRELS
                if barrels[0] == "" or barrels[1] == "": ## CHECKS THAT NEITHER BARREL IS NULL
                    errors.append("If using a double barrelled surname, both barrels must be used")
    else:
        errors.append("Surname must be filled in")

    if errors != []: ## CHECKS IF USER HAS HAD ANY ERRORS
        errormessage = "\n".join(errors) ## CONVERTS LIST OF ERRORS INTO SINGLE STRING
        CreateOutbox("Errors Found","There has been an error with creating this account:",errormessage,QtGui.QMessageBox.Critical) ## CREATES ERROR MESSAGE BOX
    else:
        password = hashlib.sha256("Quizo123".encode()).hexdigest() ## HASHES THE PASSWORD FOR STORING IN THE DATABASE
        query = 'INSERT INTO Students (Username,Password,FirstName,Surname) VALUES (?,?,?,?);' ## INSERTS NEW ACCOUNT INTO DATABASE WITH CORRECT DATA
        cur.execute(query,(username,password,firstname,surname,)) ## EXECUTES THE QUERY
        query = 'INSERT INTO STLinks (TUser,SUser) VALUES (?,?);' ## INSERTS NEW LINK BETWEEN TEACHER AND STUDENT
        cur.execute(query,(cur_user,username,)) ## EXECUTES THE QUERY
        con.commit() ## WRITES CHANGES TO THE DATABASE
        CreateOutbox("Account Created","This new account has been created!","",QtGui.QMessageBox.Information) ## CREATES ACCOUNT CREATED MESSAGE BOX
```

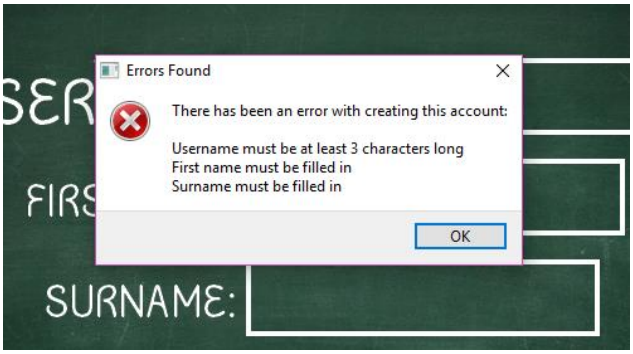
The code above is the procedure that runs when a teacher presses the create button in student registration. I wrote it using the teacher registration code, however I have made a few tweaks. The most notable is the change to validating surnames; instead of replacing the first character in each barrel, I am now using the ".title()" method to capitalise the first letter in a string, and make the rest lowercase. Another difference in this code is that I have added an extra SQL statement to be executed, which creates a new record in the "STLinks" table that adds this new student to the current teacher's class.

In order to showcase this code working as intended, I have taken some screenshots which can be seen below.



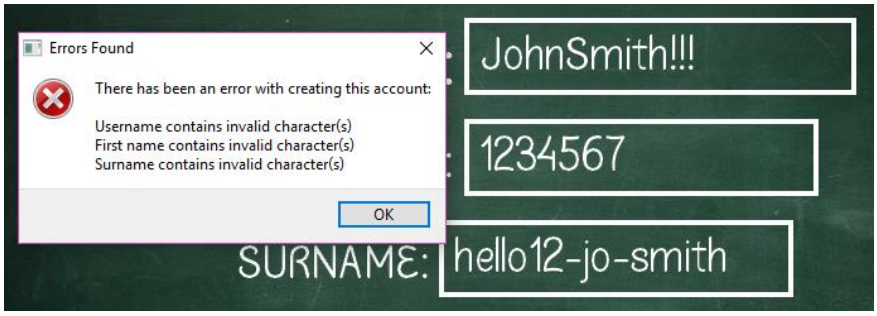


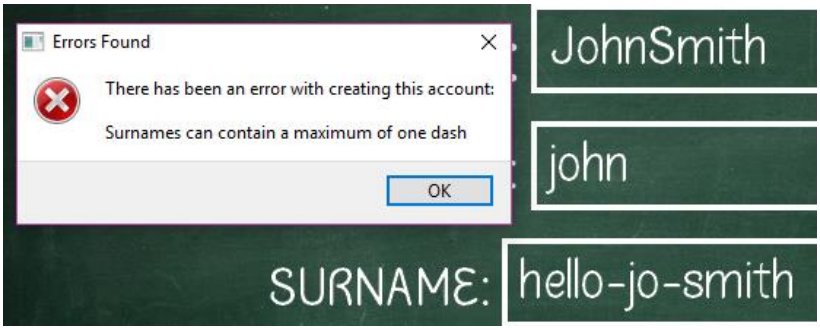
Test 1 – Opening the window – Successful – Upon pressing the add student button in the view students class, the student registration window appeared.



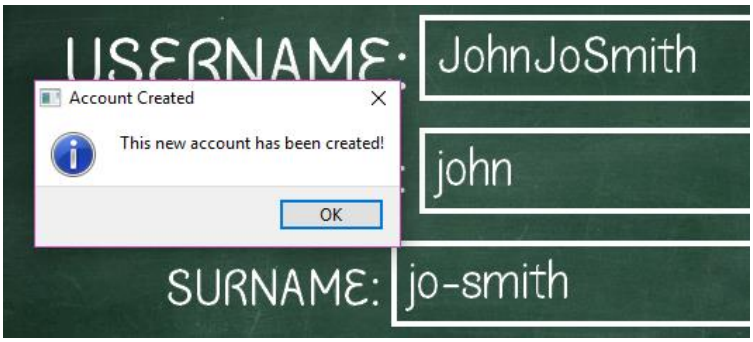
Test 2 – Creating an account without any details – Successful – The program correctly pointed out all three errors when trying to create an account without filling in the fields.

Test 3 – Creating an account with invalid characters – Successful – The program identified that punctuation is disallowed in usernames, and numbers are disallowed in both first and surnames.

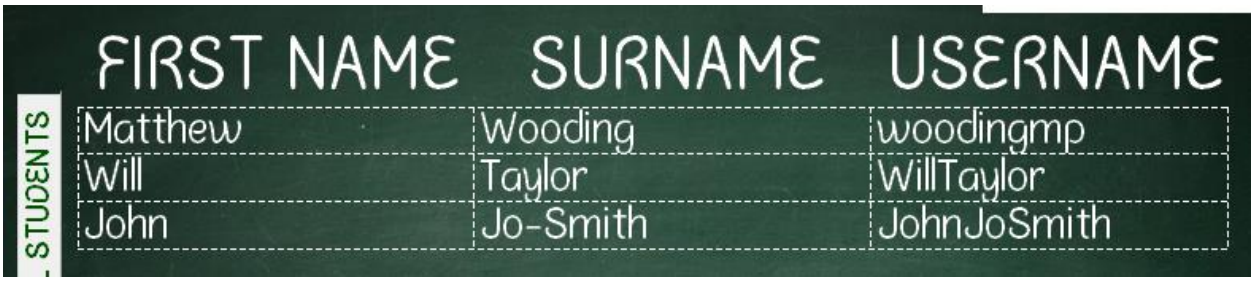




Test 4 – Using too many dashes – Successful – The program successfully prevented a surname with two dashes being used.



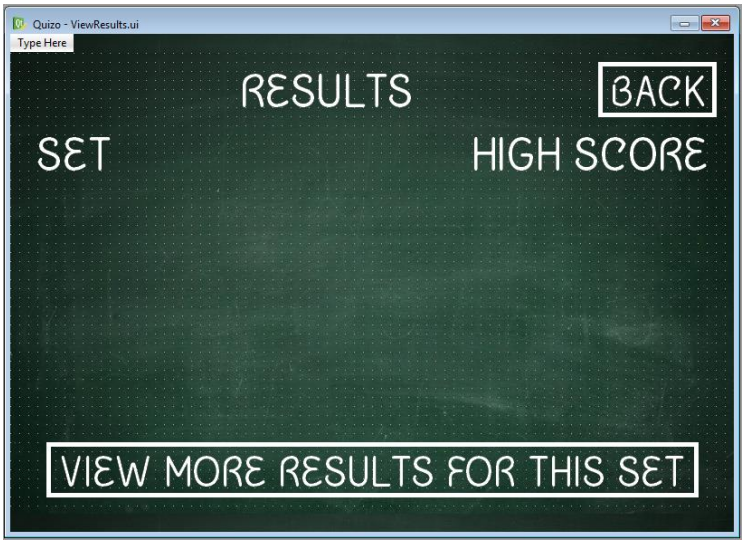
Test 5 – Using valid data – Successful – The program successfully created the account when all valid data was used. This new account was added to the database, and its link with the current teaching account can be shown, as when moving back to the view students screen, it was in the "my students" table.



Test 6 – Checking the password – Successful – To check that the correct password had been used when creating the account, I logged out of the teacher account, and tried to log back in as the new account, and the program successfully prompted me to change my password as it was still the default.

OBJECTIVES 25 AND 35 COMPLETED – TEACHERS CAN ADD NEW STUDENTS TO THE DATABASE SO THAT THEY CAN USE THE PROGRAM, SO LONG AS THEIR DETAILS ARE FIRST VALIDATED – VIEW STUDENTS WINDOW IS NOW FEATURE COMPLETE AND TEACHERS CAN REACH THE WINDOW FROM THEIR HOME SCREEN

CREATING THE VIEW RESULTS WINDOW



The student results window allows both teachers and students to view progress in all their attempted sets. Students will only be able to view their progress, whereas teachers will be able to view the progress of any student who is in their class.

Using my initial design, I have created the window using the QtDesigner, which can be seen to the side.

In order for this window to function for use by teachers and students, I will create the class and ensure that the username of the account that's results are being viewed is passed as a parameter to the necessary functions.

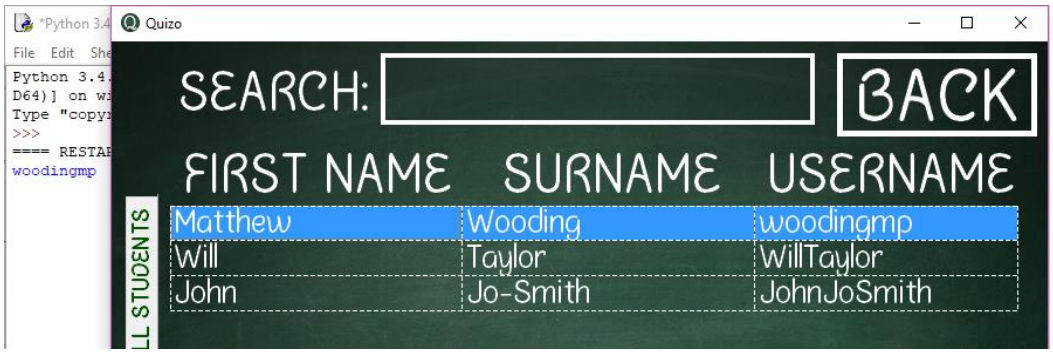
Given that this is how I will use the window, I have written the necessary code to fetch all the high scores for the given user and display them in the table, as well as the navigation code for the view students and student home windows.

NAVIGATING FROM VIEWING STUDENTS - SETUP

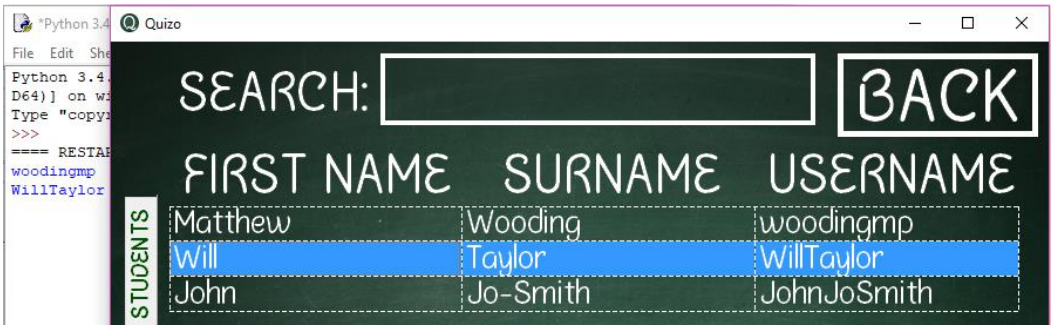
In order to view the progress of a student that a teacher has in their class, the teacher will be able to double click a student and it will send them to this next screen with the student's username as a parameter. This code can be seen below, alongside a test showing that the correct student's username is printed, and will eventually passed to the new window.

```
self.mystudstable.doubleClicked.connect(self.viewresults)
```
- Written in the "init" function

```
def viewresults(self):
    studentusername = self.findselection()[2] ## FETCHES THE USERNAME OF THE CHOSEEN STUDENT
    print(studentusername) ## CHECKING THE CORRECT USERNAME IS FETCHED - WILL PASS TO NEW WINDOW
```



Test 1 – Successful – After double clicking "woodingmp" in the table, the username "woodingmp" was printed to the IDLE



Test 2 – Successful – After double clicking "WillTaylor" in the table, the username "WillTaylor" was printed to the IDLE

Test 3 – Successful – After double clicking "JohnJoSmith" in the table, the username "JohnJoSmith" was printed to the IDLE

CLASS CREATION

```
class ViewResults(QtGui.QMainWindow,ViewResults_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.setupUi(self)
        self.backbutton.clicked.connect(self.Back)
        self.viewbutton.clicked.connect(self.analyse)

    def analyse(self):
        selection = self.resultstable.selectionModel().selectedRows() ## FINDS SELECTED ROWS IN TABLE
        if selection != []: ## CHECKS IF ANYTHING WAS SELECTED
            selectedrow = selection[0].row() ## FINDS INDEX OF SELECTED ROW
            setdetails = self.attemptedsets[selectedrow] ## GETS THE SET DETAILS FOR SELECTED SET (Name,Score,ID)
            query = "SELECT COUNT(Score) FROM Scores WHERE SID=? AND Username=?;" ## QUERY THAT FETCHES THE NUMBER OF SCORES BY THE USER IN A SET
            cur.execute(query,(setdetails[2],self.username,)) ## EXECUTES THE QUERY WITH THE CORRECT PARAMETERS
            scorenum = cur.fetchone()[0] ## FETCHES THE ACTUAL NUMBER OF SCORES
            if scorenum > 1000:
                outbox = QtGui.QMessageBox() ## CREATES THE MESSAGE BOX
                outbox.setWindowTitle("Set Analysis") ## SETS THE TITLE OF THE MESSAGE BOX
                outbox.setText("You are trying to analyse a set where there are more than 1000 scores.") ## SETS THE MAIN TEXT OF THE MESSAGE BOX
                outbox.setInformativeText("This may take some time to process, would you like to continue?\n(If there are more than 20,000 results, it is possible the program will crash)")
                outbox.setIcon(QtGui.QMessageBox.Information) ## SETS THE ICON OF THE MESSAGE BOX
                outbox.setStandardButtons(QtGui.QMessageBox.Ok | QtGui.QMessageBox.Cancel) ## SETS THE BUTTONS FOR THE MESSAGE BOX
                choice = outbox.exec() ## EXECUTES THE MESSAGE BOX AND RETRIEVES THE BUTTON PRESSED
                if choice == QtGui.QMessageBox.Ok: ## CHECKS IF OK WAS PRESSED
                    SetAnalysis.fetchresults(SetAnalysis_Window,setdetails,self.username) ## PASSES THE NECESSARY PARAMETERS TO START FETCHING RESULTS FOR ANALYSIS
                    SetAnalysis_Window.show() ## SHOWS THE ANALYSIS WINDOW
                    Results_Window.hide() ## HIDES THE RESULTS WINDOW
            elif scorenum == 1: ## CHECKS IF USER HAS ONLY ATTEMPTED THE SET ONCE
                CreateOutbox("Not Enough Attempts!","At least two attempts are required to review progress.","",QtGui.QMessageBox.Information)
            else:
                SetAnalysis.fetchresults(SetAnalysis_Window,setdetails,self.username) ## PASSES THE NECESSARY PARAMETERS TO START FETCHING RESULTS FOR ANALYSIS
                SetAnalysis_Window.show() ## SHOWS THE ANALYSIS WINDOW
        else:
            CreateOutbox("No Set Selected","You must select a question set to analyse!","",QtGui.QMessageBox.Critical)

    def displayresults(self,username):
        self.username = username ## ALLOWS ALL FUNCTIONS WITHIN THE CLASS TO USE THE USERNAME
        query = '''SELECT * FROM (SELECT Sets.SetName,Scores.Score,Scores.SID FROM Scores INNER JOIN Sets ON Scores.SID=Sets.SID WHERE Username=?
ORDER BY Score ASC) AS allscores GROUP BY SID ORDER BY Score DESC;''' ## FETCHES A USER'S HIGHEST SCORE IN ALL ATTEMPTED SETS
        cur.execute(query,(username,)) ## EXECUTES QUERY
        self.attemptedsets = cur.fetchall() ## FETCHS DATA FROM DATABASE
        model = QtGui.QStandardItemModel(len(self.attemptedsets),2) ## CREATES A MODEL TO PUT THE DATA IN
        for row in range(0,len(self.attemptedsets)): ## LOOPS THROUGH EACH SET
            for column in range(0,2): ## LOOPS THROUGH SET NAME AND SCORE FOR THE CURRENT SET
                model.setData(model.index(row,column),str(self.attemptedsets[row][column])) ## SETS THE MODEL DATA ACCORDING TO THE FETCHED DATA
        self.resultstable.setModel(model) ## SETS THE MODEL TO THE TABLE VIEW
        tempwidth = [500,265] ## COLUMN WIDTHS FOR THE TABLE
        for i in range(2): ## LOOPS THROUGH EACH COLUMN
            self.resultstable.setColumnWidth(i,tempwidth[i]) ## SETS THE WIDTH OF THE COLUMN
        self.resultstable.show() ## SHOWS THE TABLE

    def Back(self):
        if cur_user == self.username: ## CHECKS IF USER IS A STUDENT OR TEACHER
            StudentHome_Window.show() ## SHOWS THE STUDENT HOME WINDOW
        else:
            ViewStudents_Window.show() ## SHOWS THE VIEW STUDENTS WINDOW
        ViewResults_Window.hide() ## HIDES THE VIEW RESULTS WINDOW
```

As can be seen above, this class code fetches all the highest scores for the selected user's attempted sets and then displays them in a table. This code also allows the user to select a question set and use the analysis window to check progress over time.

NAVIGATING FROM VIEWING STUDENTS – ADDITIONAL CODE

Now that I have written the code for the class, I have gone back to this window to add in the code for starting the new window when a student is double clicked, and this code can be seen below.

```
def results(self):
    ViewResults.displayresults(ViewResults_Window,cur_user) ## BEGINS THE FUNCTION TO DISPLAY HIGH SCORES IN THE VIEW RESULTS WINDOW
    ViewResults_Window.show() ## SHOWS THE VIEW RESULTS WINDOW
    StudentHome_Window.hide() ## HIDES THE STUDENT HOME WINDOW
```

Inside the StudentHome class

```
def viewresults(self):
    studentusername = self.findselection()[2] ## FETCHES THE USERNAME OF THE CHOSEEN STUDENT
    ViewResults.displayresults(ViewResults_Window,studentusername) ## BEGINS THE FUNCTION TO DISPLAY HIGH SCORES IN THE VIEW RESULTS WINDOW
    ViewResults_Window.show() ## SHOWS THE VIEW RESULTS WINDOW
    ViewStudents_Window.hide() ## HIDES THE VIEW STUDENTS WINDOW
```

Inside the ViewStudents class

TESTING THE WINDOW

Having implemented my initial code for this new window and its functionality with other windows within the program, I will now test every possible interaction with the window, in order to check that the program can't be crashed or produce errors.

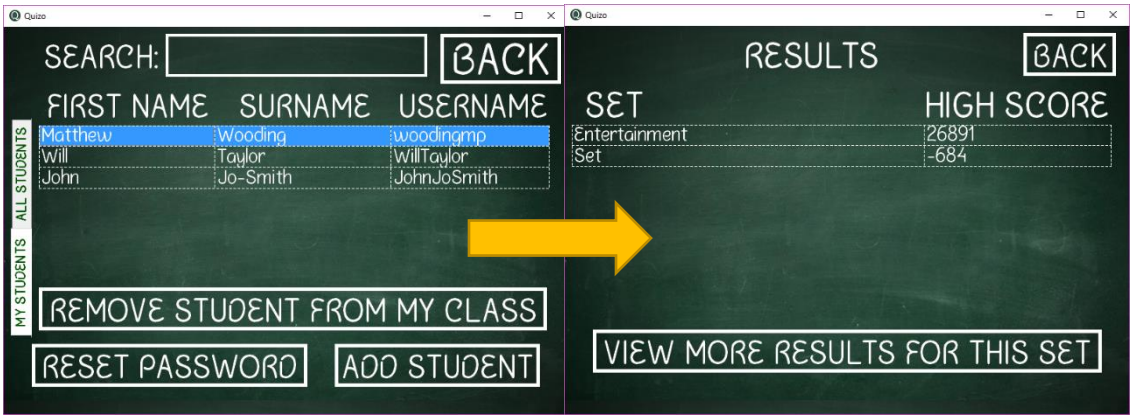
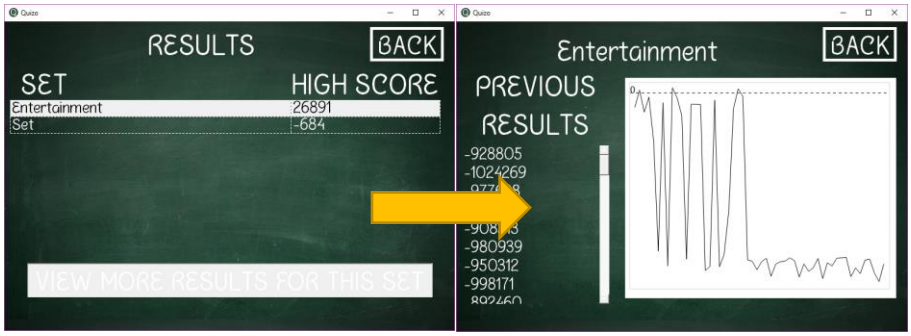


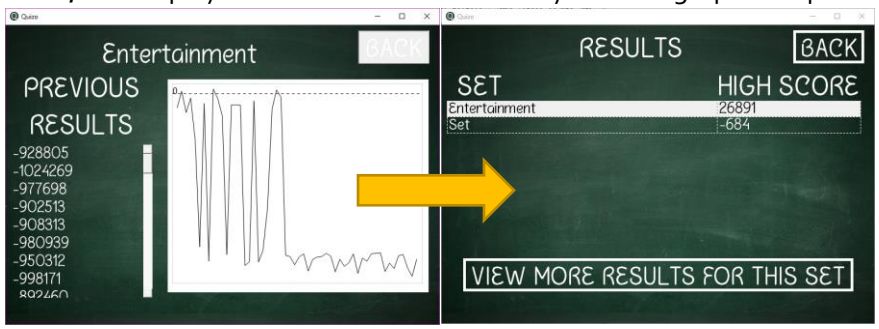
Table: Scores

	Username	SID	Score
	Filter	Filter	Filter
1	Student6	3	27316
2	woodingmp	3	26891
3	Student9	3	25824
4	woodingmp	3	21627
5	woodingmp	3	14786
6	WillTaylor	3	12315
7	woodingmp	2	-684
8	woodingmp	2	-1212

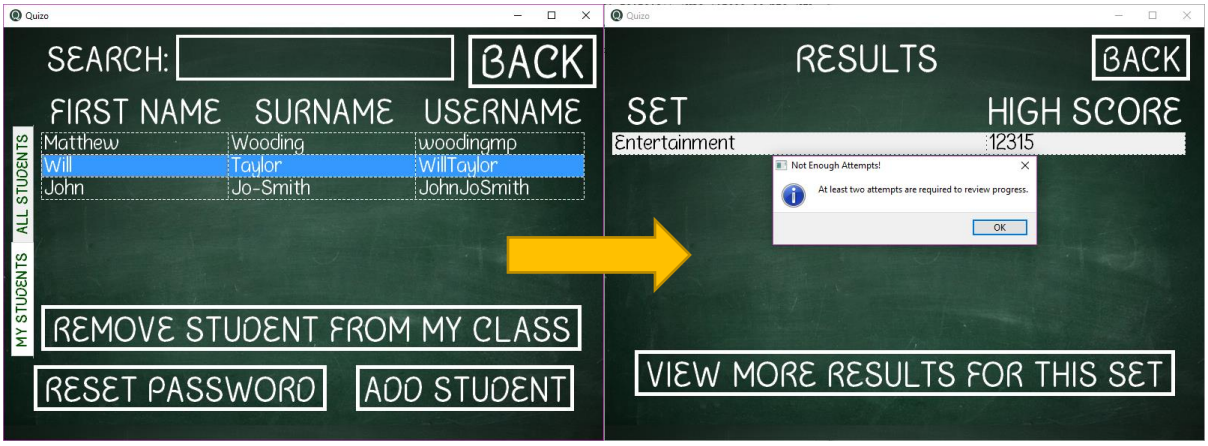
Test 1 – View a student's results while logged in as a teacher – Successful – While logged in as "theteacher" I double clicked the user "woodingmp" from the my students tab in the view students window. Doing this successfully sent me to the view results window, which displayed that "woodingmp" had attempted two question sets, alongside their highest scores in each. As can be seen in the database browser, when ordered by score, it shows that the high scores shown are correct.



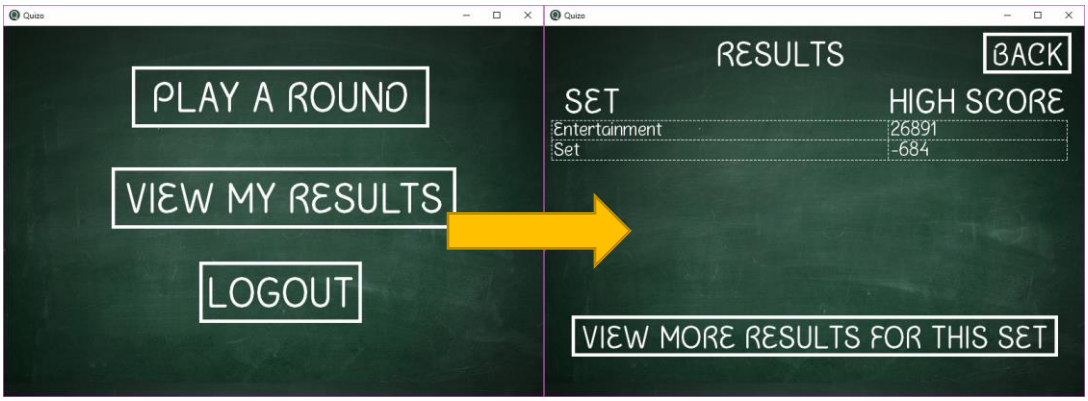
Test 2 – Analyse a user's attempts in a selected set – Successful – When I selected the "Entertainment" set and pressed the view more button, the program successfully directed me to the analysis page for this set, and displayed all the results obtained by "woodingmp" as expected.



Test 3 – Pressing the "Back" button – Successful – When pressing the back button, the program successfully returned me to the View Results Window.



Test 4 – Trying to analyse a set with only one result in the database – Successful – In order to check the program was checking how many times a set had been attempted, I viewed the results for the user "WillTaylor" and tried to view more results for the "Entertainment" set, for which this account had only recorded one result. As expected, the program prevented me from analysing the results.



Test 5 – Viewing results as a student – Successful – I logged out of the program as "theteacher" and logged back in as "woodingmp" to check that the window would be the same view as a teacher would see.

OBJECTIVES 28,36, AND 37 COMPLETED – STUDENTS AND TEACHERS CAN ACCESS THE RESULTS OF INDIVIDUAL STUDENTS, SEEING THEIR HIGH SCORES IN EACH OF THE SETS THEY HAVE ATTEMPTED, AND CAN DO IN-DEPTH ANALYSIS IF DESIRED

CREATING THE VIEW SETS WINDOW

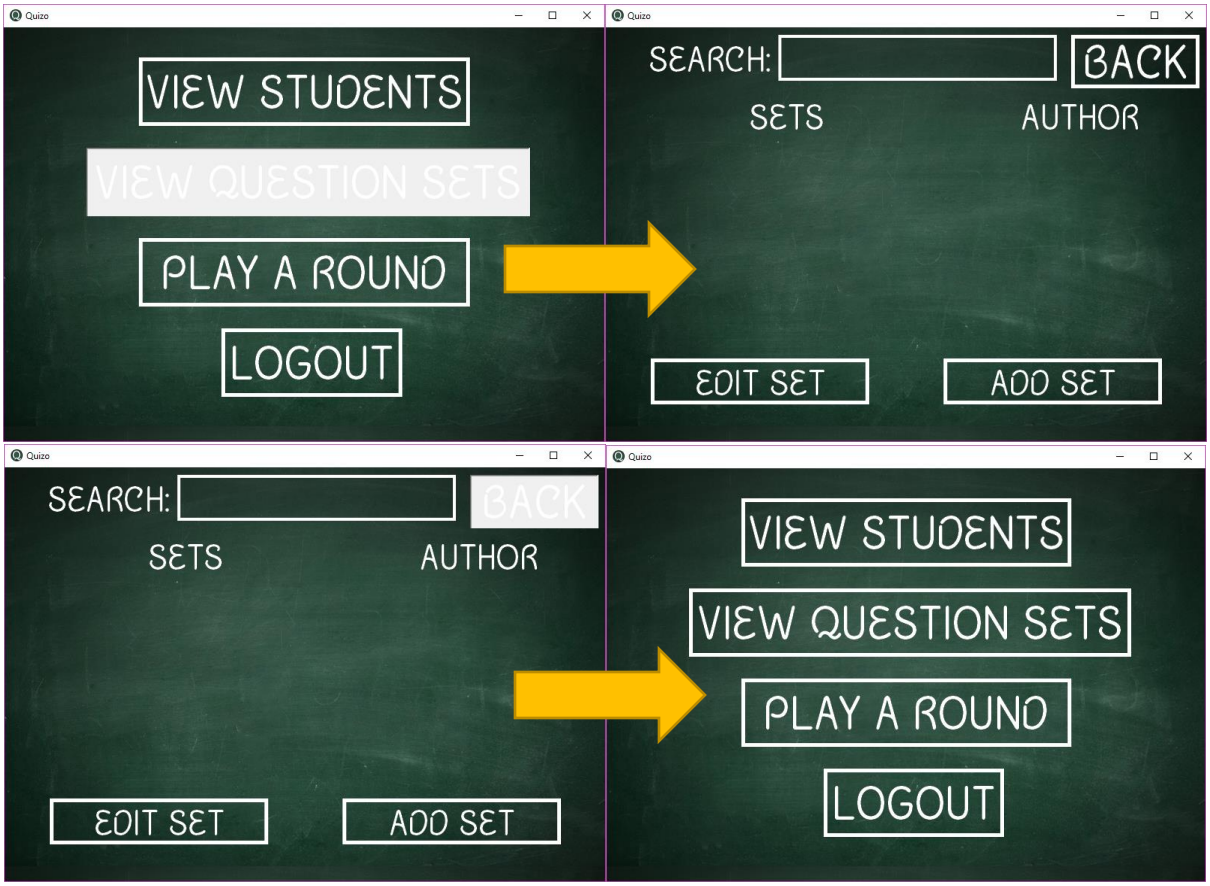
The next window to create is the view sets window, which can be viewed only by teachers. This window allows them to view all sets, edit ones they've previously created, and add new ones to the program. I have created a design for this in QtDesigner, which can be seen to the right, and I have written the code below to implement the window into the program, and allow the user to navigate to the window.


```
    def sets(self):
        ViewSets_Window.show() ## SHOWS THE VIEW SETS WINDOW
        TeacherHome_Window.hide() ## HIDES THE TEACHER HOME WINDOW
```

Added into the TeacherHome class

```
class ViewSets(QtGui.QMainWindow,ViewSets_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.setupUi(self)
        self.backbutton.clicked.connect(self.Back)
        self.addsetbutton.clicked.connect(self.addset)
        self.editsetbutton.clicked.connect(self.editset)

    def editset(self):
        print("EDIT SET")

    def addset(self):
        print("ADD SET")

    def Back(self):
        TeacherHome_Window.show() ## SHOWS THE TEACHER HOME WINDOW
        ViewSets_Window.hide() ## HIDES THE VIEW SETS WINDOW
```

New View Sets class for the View Sets window



Navigation Test – Successful – To ensure teachers could reach the new window and return to their home screen, I logged into the program as "theteacher" and then successfully moved to the "ViewSets" window, and then back to the home window.

DISPLAYING SETS, SEARCHING AND LOCATING SELECTIONS

This window will be displaying the exact same data as the "Play Game" window which was for selecting quizzes to attempt, and for this reason, I have simply copied over the code for displaying all the question sets, along with their authors, the search function, and the function for fetching the selected question set, all of which can be seen below inside this class.

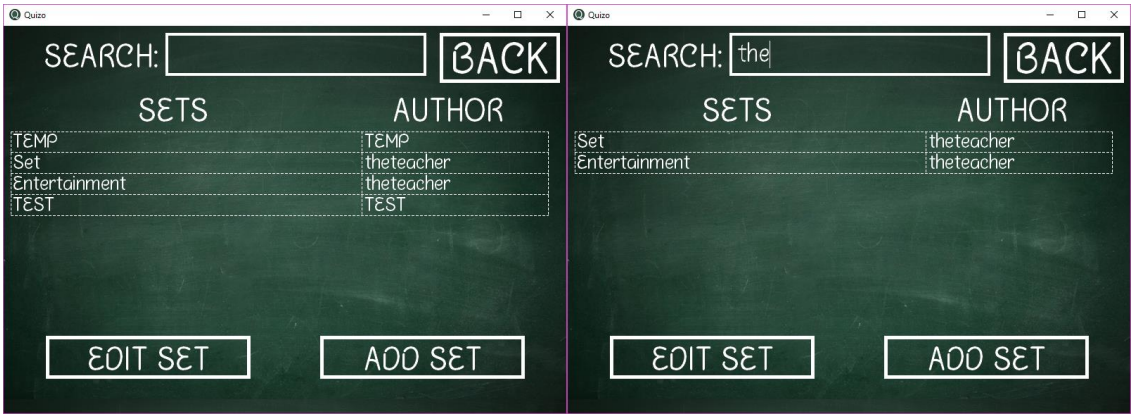
```
def sets(self):
    ViewSets.search(ViewSets_Window) ## DISPLAYS THE TABLE BEFORE THE WINDOW OPENS
```

Added extra line in Teacher Home class to display the table before the window is opened

```
class ViewSets(QtGui.QMainWindow,ViewSets_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.setupUi(self)
        self.backbutton.clicked.connect(self.Back) ## RUNS BACK FUNCTION WHEN BACK BUTTON PRESSED
        self.addsetbutton.clicked.connect(self.addset) ## RUNS ADD SET FUNCTION WHEN ADD BUTTON PRESSED
        self.editsetbutton.clicked.connect(self.editset) ## RUNS EDIT SET FUNCTION WHEN EDIT BUTTON PRESSED
        self.searchedit.textChanged.connect(self.search) ## RUNS SEARCH FUNCTION WHENEVER SEARCH BAR CONTENTS CHANGE

    def editset(self):
        selection = self.setstable.selectionModel().selectedRows() ## FETCHES THE SELECTED ROWS FROM THE TABLE
        if selection == []: ## CHECKS IF ANY ROW HAS BEEN SELECTED
            CreateOutbox("Selection Not Found","You must select a question set to edit!","",QtGui.QMessageBox.Critical)
        else:
            selectedrow = selection[0].row() ## FINDS INDEX OF SELECTED ROW
            setdetails = self.allsets[selectedrow] ## SETS THE CURRENT PLAYSET TO THE SELECTED SET
            if setdetails[1] == cur_user: ## CHECKS THAT THE CURRENT USER IS THE AUTHOR OF THE SET
                print(setdetails)
            else:
                CreateOutbox("Selection Error","You may only edit sets that you have created!","",QtGui.QMessageBox.Information)

    def addset(self):
        print("ADD SET")

    def search(self):
        word = "%" + self.searchedit.text() + "%" ## GETS STRING FROM SEARCH BAR
        for char in ["'",";",'"']: ## LOOPS THROUGH THE DISALLOWED CHARACTERS
            word = word.replace(char,"") ## REMOVES CHARACTERS FROM USER'S ENTRY
        query = 'SELECT SetName,Author,SID FROM Sets WHERE SetName LIKE ? OR Author LIKE ?;' ## QUERY TO FETCH SET NAMES WITH SIMILAR TITLES OR AUTHORS TO USER'S SEARCH
        cur.execute(query,(word,word,)) ## EXECUTES QUERY
        self.allsets = cur.fetchall() ## FETCHES DATA FROM DATABASE
        model = QtGui.QStandardItemModel(len(self.allsets),2) ## CREATES A MODEL TO PUT THE DATA IN
        for row in range(0,len(self.allsets)): ## LOOPS THROUGH EACH SET
            for column in range(0,2): ## LOOPS THROUGH SET NAME AND AUTHOR FOR THE CURRENT SET
                model.setData(model.index(row,column),str(self.allsets[row][column])) ## SETS THE MODEL DATA ACCORDING TO THE FETCHED DATA
        self.setstable.setModel(model) ## SETS THE MODEL TO THE TABLE VIEW
        self.setstable.setColumnWidth(0,500) ## SETS COLUMN WIDTH FOR SETS COLUMN
        self.setstable.setColumnWidth(1,265) ## SETS COLUMN WIDTH FOR AUTHOR COLUMN
        self.setstable.show() ## SHOWS THE TABLE


    def Back(self):
        self.searchedit.setText("") ## CLEARS THE SEARCH BAR
        TeacherHome_Window.show() ## SHOWS THE TEACHER HOME WINDOW
        ViewSets_Window.hide() ## HIDES THE VIEW SETS WINDOW
```

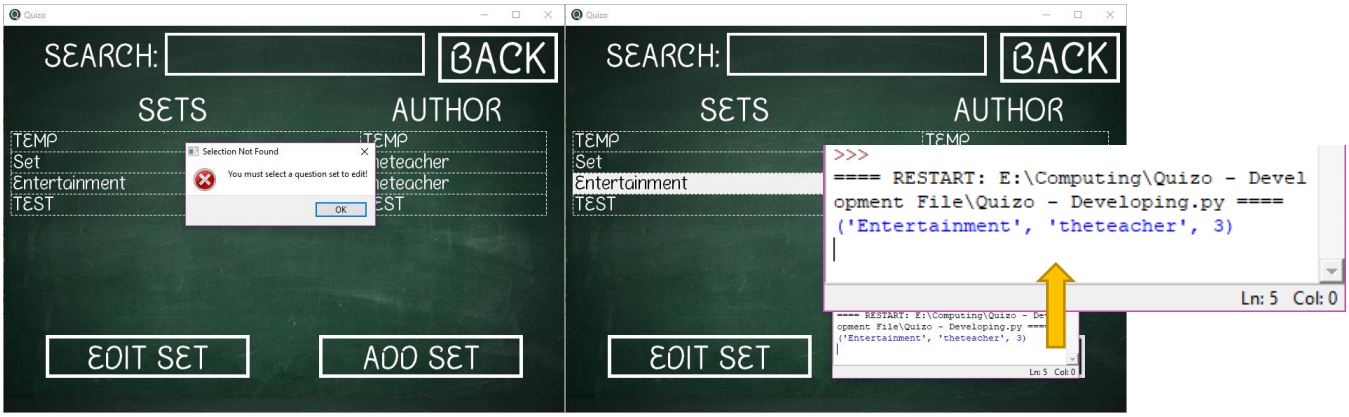
Since the only function that requires the user to select a row is the edit function, I decided to just move the code from the original "findselection" function to the "editset" function.

Having now copied over this code, even though I have previously tested it, I will check again to make sure it is all working in this new class.



Test 1 – Opening the window – Successful – Upon first loading the window, the program successfully displayed all the question sets in the program

Test 2 – Searching – Successful – When using the search bar and entering "the", the program successfully only displayed the two sets "Set" and "Entertainment" that had been made by "**the**teacher"



Test 3 – Editing a set without selecting a set – Successful – When trying to edit a set without actually selecting anything, the program successfully gave an output stating I had to select an item

Test 4 – Editing a set after selecting something – Successful – When trying to edit a set after actually selecting one, the program successfully printed the correct set details to the IDLE.

OBJECTIVES 26, AND 43 COMPLETED – TEACHERS CAN ACCESS A WINDOW FOR VIEWING ALL QUESTION SETS, DISPLAYED IN A TABLE

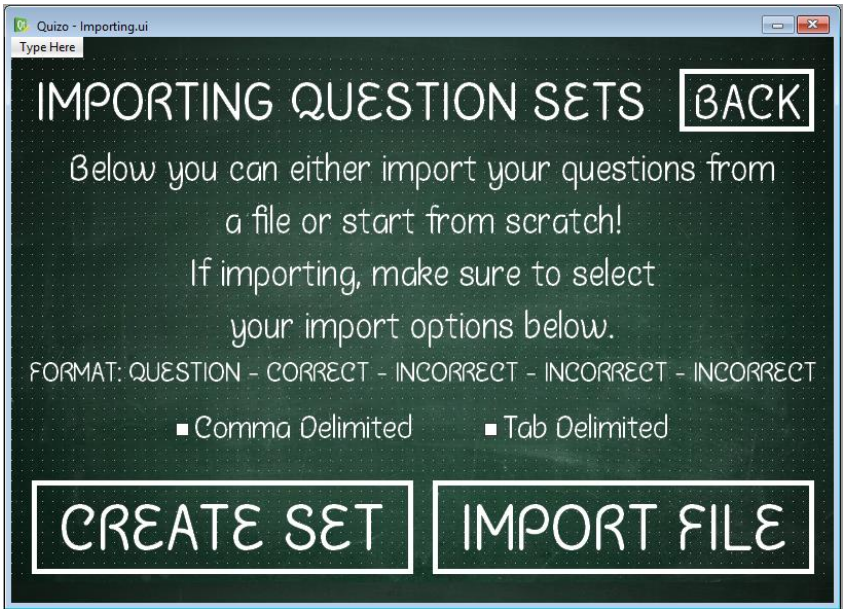
Now that this window is feature complete except for navigating to future windows, I will move onto creating the next window.

CREATING THE IMPORT WINDOW

In this program, teachers need to be able to add their own question sets, so that more questions will be available for students to attempt. In order for teachers to do this, the process needs to be fast and easy. While I will add a way for teachers to build their quizzes entirely inside the program, I think that the ability to import questions from excel spreadsheets or text documents would be helpful, and could make the process of adding new questions easier for teachers who already have large banks of questions that they've accumulated over their career.

To the right is the window that I have created in QtDesigner that allows teachers to either import their own questions, or add their own into the program. This screen will be accessible by pressing the "Add Set" button in the "View Sets" window.

I have added this new window to the program with its own class, and I have added the necessary navigation code between this window and the previous, which can be seen below.

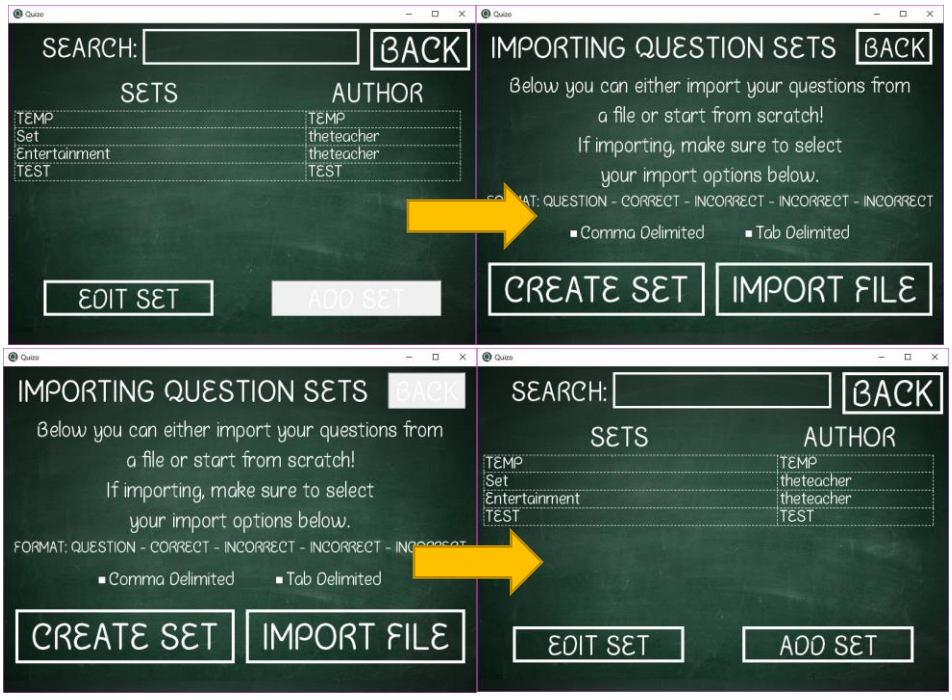


```
def addset(self):
    Importing_Window.show() ## SHOWS THE IMPORTING WINDOW
    ViewSets_Window.hide() ## HIDES THE VIEW SETS WINDOW
```

Navigation code within the View Sets window to go to the Importing window when the add set button is pressed

```
class Importing(QtGui.QMainWindow,Importing_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.setupUi(self)
        self.importbutton.clicked.connect(self.imp) ## RUNS THE IMP FUNCTION WHEN IMPORT IS CLICKED
        self.createbutton.clicked.connect(self.create) ## RUNS THE CREATE FUNTION WHEN CREATE IS CLICKED
        self.backbutton.clicked.connect(self.back) ## RUNS THE BACK FUNCTION WHEN BACK IS CLICKED

    def back(self):
        ViewSets_Window.show() ## SHOWS THE VIEW SETS WINDOW
        Importing_Window.hide() ## HIDES THE IMPORTING WINDOW

    def imp(self):
        print("IMPORT")

    def create(self):
        print("CREATE")
```

Now that I have implemented this code, I will test that the program can successfully navigate to and from this window.

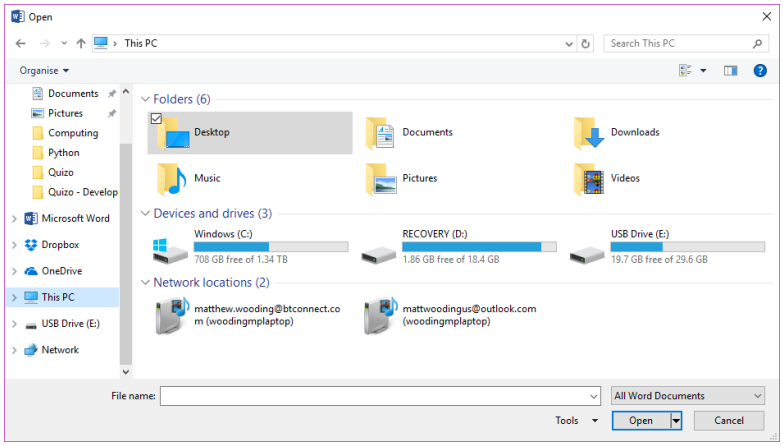


As can be seen in the above tests, the program can successfully move between the two windows.

In order to import questions into my database, I have decided that I will first try to import files in a separate python program.

IMPORTING FILES INTO PYTHON

In order to import files into my program, I will require some form of file browser, similar to the one shown to the side which is used for opening files in Microsoft word.

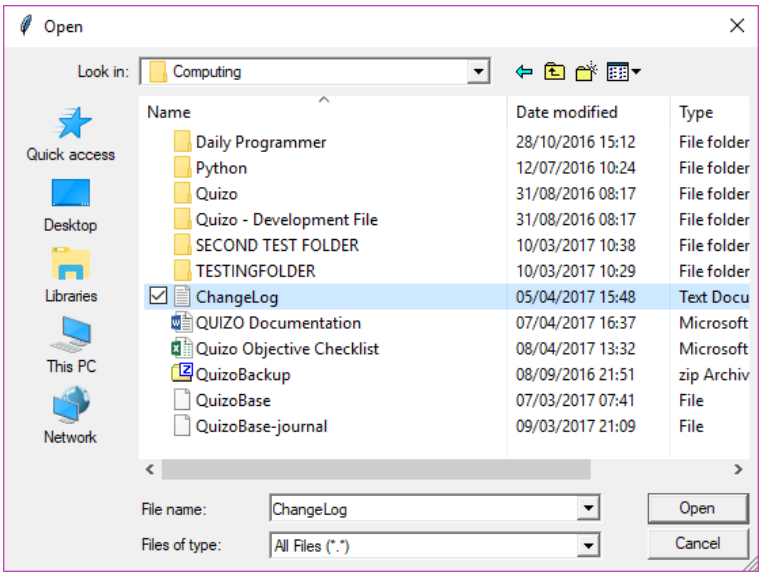


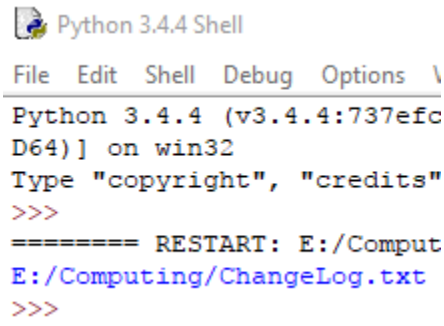
Having done some research into using file browsers in python, I have decided that I will be using the "tkinter" python module, and more specifically the "filedialog" portion, to allow a user to select a file from their computer.

```
from tkinter import *
import sys,os

browser = Tk()
browser.withdraw()
browser.filename = filedialog.askopenfilename(initialdir="/")
print(browser.filename)
```

The code above is code that I have written in a separate file to my main application. I have imported the entirety of the "tkinter" module, and have created a window labelled "browser". This code just opens up a file browser, with the initial directory just being the root of the storage device where the file is being executed from, and allows the user to select any file. To check that the correct file is being selected, I have added a print statement to print the location of the file selected.

I have done some tests below to show the code working as expected.





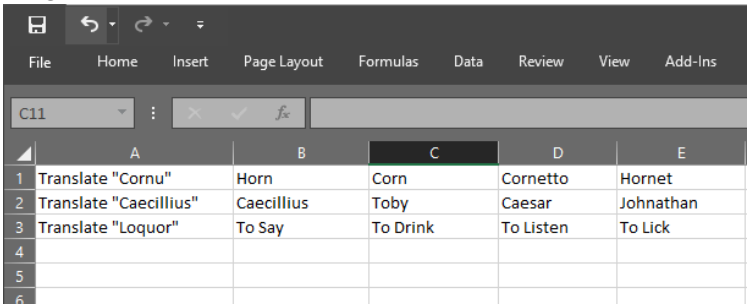
As shown above, when "ChangeLog.txt" was selected, its correct location was printed.

Using this code as a foundation, I have added some extra lines of code to do the following:

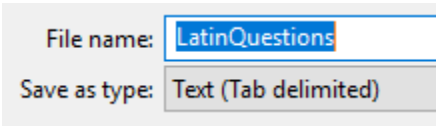
- Allow only specific file types to be selected (.txt and .csv files)
- Open the file
- Print the contents of the file to the python IDLE

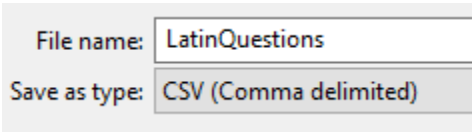
```
browser.filename = filedialog.askopenfilename(initialdir="/",title="Select File",filetypes = (("Text File","*.txt"),("CSV File","*.csv")))

file = open(browser.filename,"rt")
contents = file.read()
file.close()
print(contents)
```

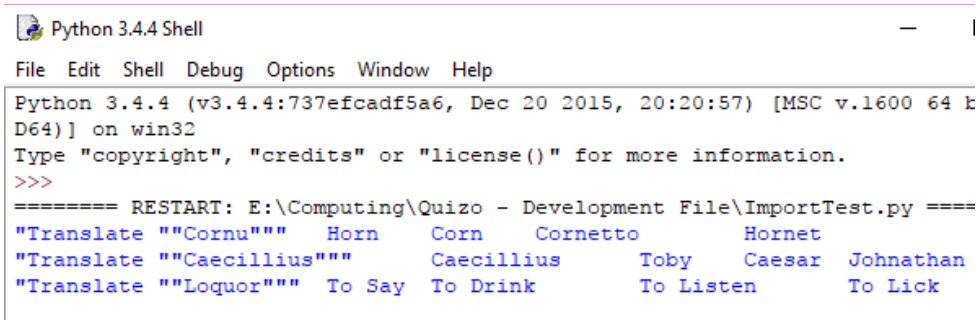
In order to test these new features, I have created a new file with a few questions inside in the correct format, and I have exported this file into both a .txt and .csv file. I will now try to open these files in the program, and check that the contents are correct.



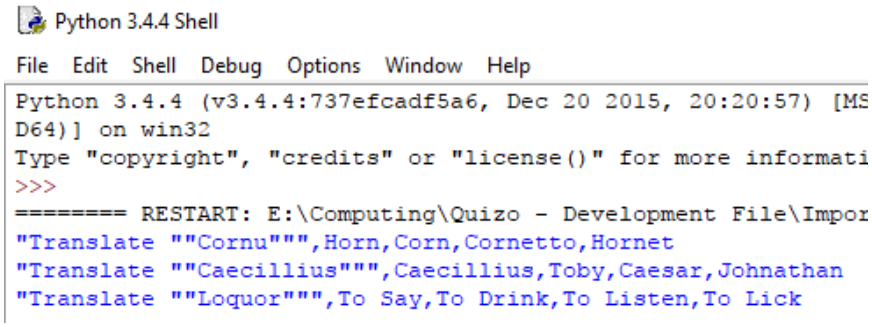
	A	B	C	D	E
1	Translate "Cornu"	Horn	Corn	Cornetto	Hornet
2	Translate "Caecillius"	Caecillius	Toby	Caesar	Johnathan
3	Translate "Loquor"	To Say	To Drink	To Listen	To Lick
4					
5					
6					







Test 1 – Loading .txt file – Successful – Contents were successfully read and printed



Test 2 – Loading .csv file – Successful – Contents were successfully read and printed

Now that the program can read the contents of a chosen file, it needs to be able to format the document into a list that can be read by the program (each question has its own list, and then all these lists are inserted into a parent list).

For tab delimited files, I have written the following code to create this list:

```
tempquestions = contents.split("\n")
questions = []
for each in tempquestions:
    if each != "":
        each = each.split("\t") ## WORKS FOR TAB DELIMITED FILES
        questions.append(each)
print(questions)
```

For comma delimited files, the code is identical, except instead of "\t" in the fifth line, it would split each question with ",".

To show that this code works as expected, I have run this code on the tab delimited file I created beforem and the program output this list:

```
[['"Translate ""Cornu"""', 'Horn', 'Corn', 'Cornetto', 'Hornet']
, ['"Translate ""Caecillius"""', 'Caecillius', 'Toby', 'Caesar',
'Johnathan'], ['"Translate ""Loquor"""', 'To Say', 'To Drink', '
To Listen', 'To Lick']]
```

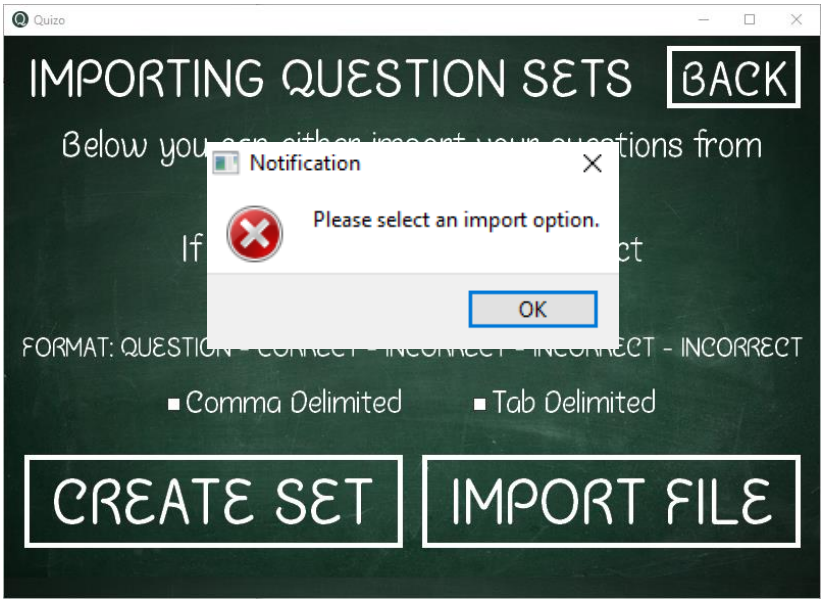
As can be seen, the program successfully split the file into the correct lists. The only issue with the code I have written is that when quote marks are used, the program ends up with a large number of unneccessary quote marks.

Due to time constraints and the amount of code required to try and fix this, I have decided that this is a minor problem for the first prototype. Shoud users end up importing questions with too many quote marks, in the program's initial state they will be able to simply edit each question to remove the quote marks. In future iterations of the program, I will attempt to fix this issue at the source.

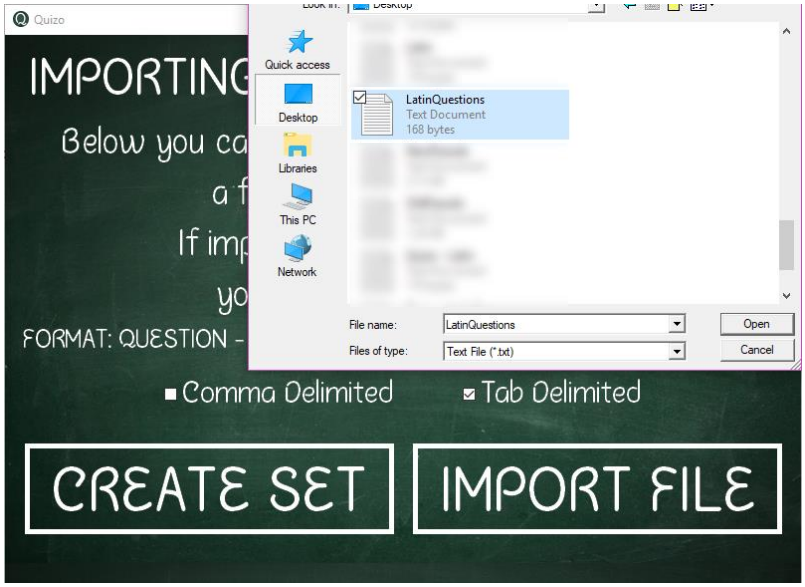
IMPLEMENTING FILE IMPORTING INTO QUIZO

Having now written code that can import files and format them into the lists that my program can utilise, I have moved the code over to my program, and also added the ability for the user to check which type of delimiter is used when trying to import their questions. This code can be seen below, alongside a short test showing that the check box is working as intended.

```
def imp(self):
    if (self.tab.isChecked() != True) and (self.comma.isChecked() != True): ## CHECKS WHETHER A CHECKBOX HAS BEEN TICKED
        CreateOutbox("Notification","Please select an import option.","",QtGui.QMessageBox.Critical)
    else:
        browser = Tk() ## CREATES A TKINTER WINDOW
        browser.withdraw() ## REMOVES THE WINDOW SO IT EXISTS BUT CAN'T BE SEEN
        browser.filename = filedialog.askopenfilename(initialdir = "/",title = "Select File",filetypes = (("Text File","*.txt"),("CSV File","*.csv")))
        if browser.filename != "": ## CHECKS IF A FILE HAS BEEN SELECTED
            file = open(browser.filename,"rt") ## OPENS THE SELECTED FILE
            contents = file.read() ## READS THE CONTENTS OF THE FILE TO A STRING
            temp = contents.split("\n") ## SPLITS THE CONTENTS INTO EACH QUESTION BY LINE
            importedquestions = [] ## LIST OF ALL IMPORTED QUESTIONS
            for each in temp: ## LOOPS THROUGH EACH LINE OF THE FILE
                if each !='': ## CHECKS IF LINE IS EMPTY
                    if self.tab.isChecked() == True: ## CHECKS WHICH TYPE OF DELIMITER HAS BEEN CHOSEN
                        each = each.split("\t") ## SPLITS THE ROW BY TAB
                    else:
                        each = each.split(",") ## SPLITS THE ROW BY COMMA
                    importedquestions.append(each) ## ADDS THE LIST TO THE QUESTION LIST
            print(importedquestions)
```

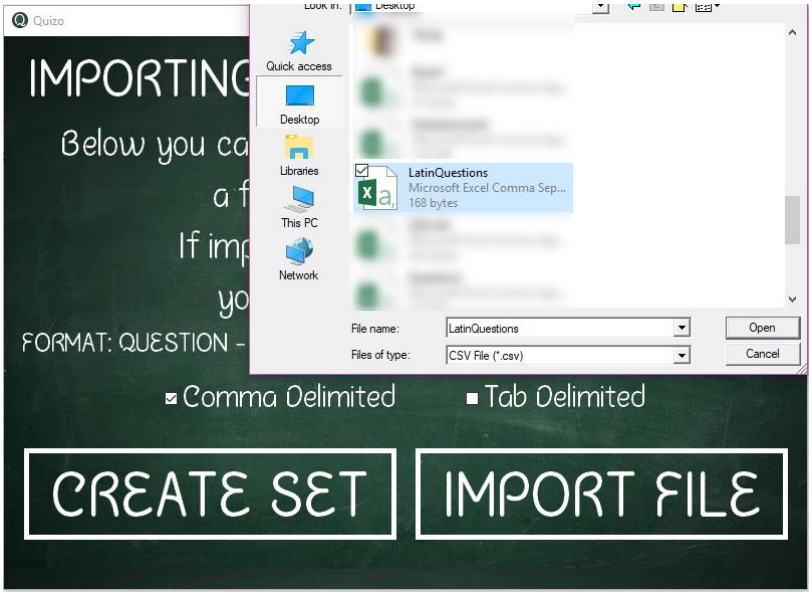


Test 1 – Importing without selecting an option – Successful – The program successfully prevented me from selecting a file to import when I hadn't chosen either comma or tab as a delimiter.



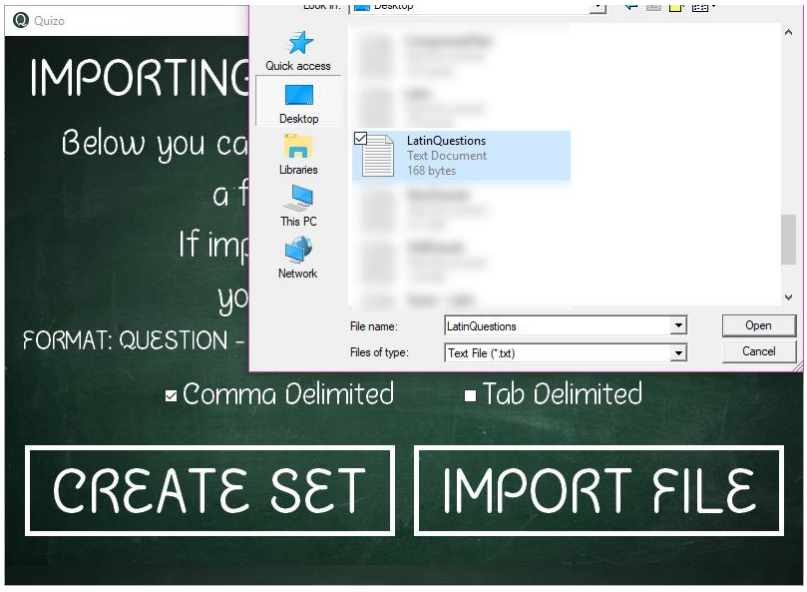

```
==== RESTART: E:\Computing\Quizo - Development File\Quizo - Developing.py ====
[['"Translate ""Cornu"""', 'Horn', 'Corn', 'Cornetto', 'Hornet'], ['"Translate "
"Caecillius"""', 'Caecillius', 'Toby', 'Caesar', 'Johnathan'], ['"Translate ""Lo
quor"""', 'To Say', 'To Drink', 'To Listen', 'To Lick']]
```

Test 2 – Importing a tab delimited file – Successful – Selecting a tab delimeted file to import results in a list with nested lists within it containing all of the questions, in the correct format.




```
[['"Translate ""Cornu"""', 'Horn', 'Corn', 'Cornetto', 'Hornet'], ['"Translate "
"Caecillius"""', 'Caecillius', 'Toby', 'Caesar', 'Johnathan'], ['"Translate ""Lo
quor"""', 'To Say', 'To Drink', 'To Listen', 'To Lick']]
```

Test 3 – Importing a comma delimited file – Successful – Selecting a comma delimited file to import results in a list with nested lists within it containing all of the questions, in the correct format.

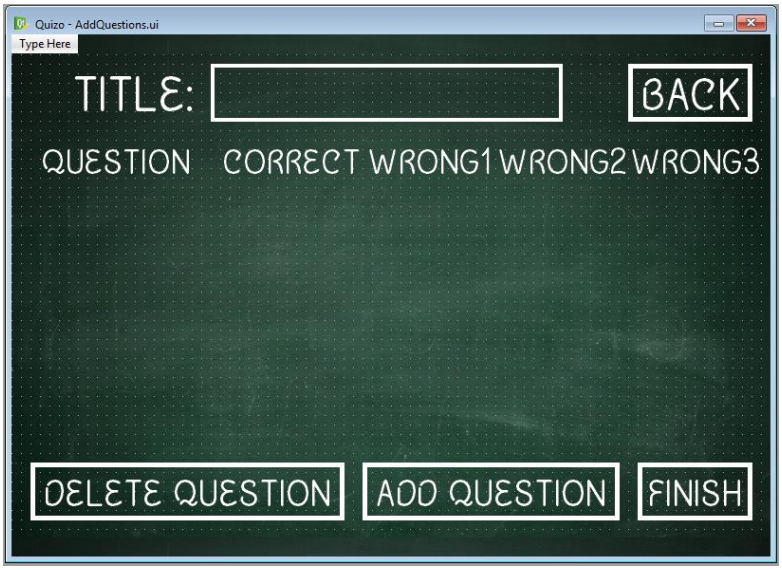



```
[['"Translate ""Cornu"""\tHorn\tCorn\tCornetto\tHornet'], ['"Translate ""Caecill
ius"""\tCaecillius\tToby\tCaesar\tJohnathan'], ['"Translate ""Loquor"""\tTo Say\
tTo Drink\tTo Listen\tTo Lick']]
```

Test 4 – Importing a tab delimited file, with the comma delimited option – Neither success nor failure – When doing this test, I didn't expect the program to magically be able to output the correct list, and instead of trying to do create a list with one delimiter and then the other, I will simply provide an output specifying that the program could not successfully import the file.

OBJECTIVE 45 COMPLETED – PROGRAM HAS A WORKING FILE BROWSER

CREATING THE SET CREATION WINDOW

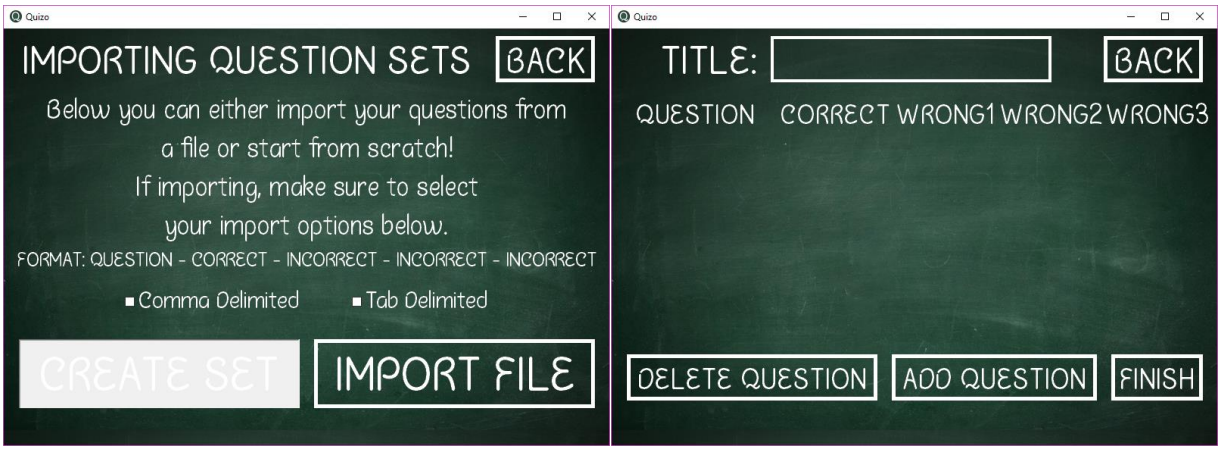


The last window in this portion of the program is the set creation window, which will be used by teachers to double check their question sets before they upload them to the database. They will be able to see all their questions and answers, add new questions, remove questions, and edit any existing questions.

The design for this window can be seen to the side as I have created it using the QtDesigner.

I have written the code for implementing the window into the program below, and I have also added some navigation code so that the user can reach the window. A test for this can be seen below also.

```
class AddQuestions(QtGui.QMainWindow,AddQuestions_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.setupUi(self)
        self.finishbutton.clicked.connect(self.finish)
        self.addbutton.clicked.connect(self.add)
        self.deletebutton.clicked.connect(self.delete)
        self.backbutton.clicked.connect(self.Back)

    def Back(self):
        ViewSets_Window.show() ## SHOWS THE VIEW SETS WINDOW
        AddQuestions_Window.hide() ## HIDES THE ADD QUESTIONS WINDOW

    def add(self):
        print("ADD")

    def delete(self):
        print("DELETE")

    def finish(self):
        print("FINISH")
```



Test – Navigating to window – Successful – Program opened window successfully when create pressed

As I will be using the same window for adding new quizzes, editing existing ones, and importing, I will need to pass parameters to the window for when it is loaded. The parameters that I will be passing to the window, are the name of the set (will be empty if creating a new set), and the list of all questions (empty if creating a new set).

When I was making the importing window, I didn't have any validation for trying to import files in an incorrect format. The reason I didn't validate this in the previous window is because I plan to do this now. I plan to validate whether or not a file is imported correctly by trying to insert the data into a table model; if the data cannot be inserted correctly, there will have been a problem with importing, and a message box can be created.

The code for creating a table of the given questions can be seen below.

```
def displayquestions(self,name,questions):
    self.titleedit.setText(name) ## SETS TITLE OF SET
    model = QtGui.QStandardItemModel(len(questions),5) ## CREATES A TABLE MODEL
    try: ## ATTEMPTS TO EXECUTE THE FOLLOWING CODE, OTHERWISE WILL JUMP TO EXCEPT
        for x in range(0,len(questions)): ## LOOPS THROUGH EACH QUESTION IN QUESTIONS LIST
            for i in range(0,5): ## LOOPS THROUGH EACH COLUMN
                model.setData(model.index(x,i),str(questions[x][i])) ## SETS DATA IN TABLE
    except:
        model = QtGui.QStandardItemModel(0,5) ##CREATES EMPTY MODEL
        CreateOutbox("Notification","There was an issue importing your questions, ensure they are in the format you specified and please try again.","",QtGui.QMessageBox.Critical)
    self.questiontable.setModel(model) ## SETS MODEL TO TABLE
    tempwidth = [210,153,139,139,139] ## LIST OF COLUMN WIDTHS
    for i in range(5): ## LOOPS THROUGH EACH COLUMN
        self.questiontable.setColumnWidth(i,tempwidth[i]) ## SETS WIDTH OF COLUMN
    self.questiontable.show() ## DISPLAYS TABLE
```

I have added the necessary navigation code to the previous classes as follows:

```
def create(self):
    AddQuestions.displayquestions(AddQuestions_Window,"",[]) ## CREATES EMPTY TABLE
    AddQuestions_Window.show() ## SHOWS THE ADD QUESTIONS WINDOW
    Importing_Window.hide() ## HIDES THE IMPORTING WINDOW
```

Added inside Importing window for when create button is pressed

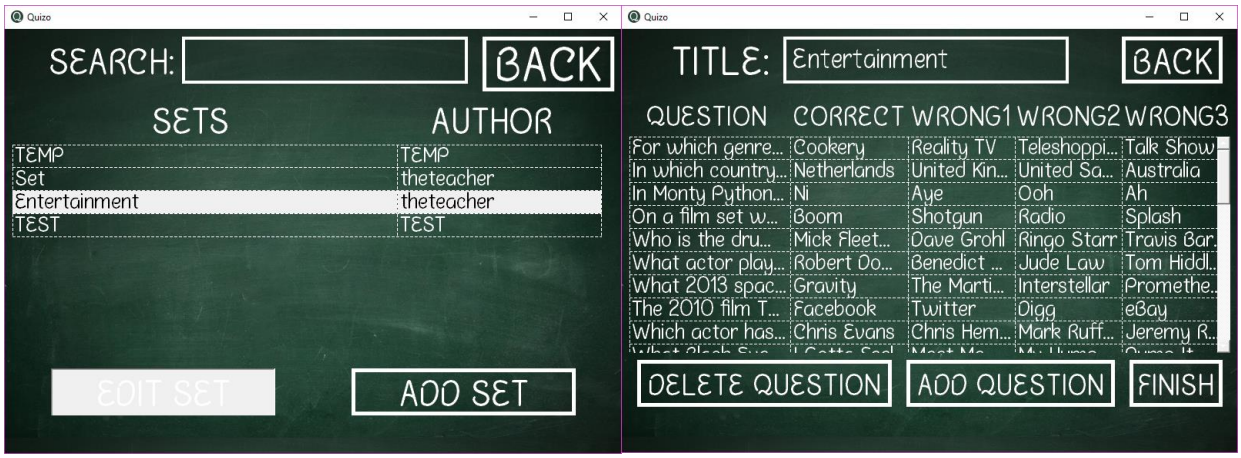
```
AddQuestions.displayquestions(AddQuestions_Window,"",importedquestions) ## CREATES TABLE WITH IMPORTED QUESTIONS
AddQuestions_Window.show() ## SHOWS THE ADD QUESTIONS WINDOW
Importing_Window.hide() ## HIDES THE IMPORTING WINDOW
```

Added inside the imp function for when a user tries to import questions

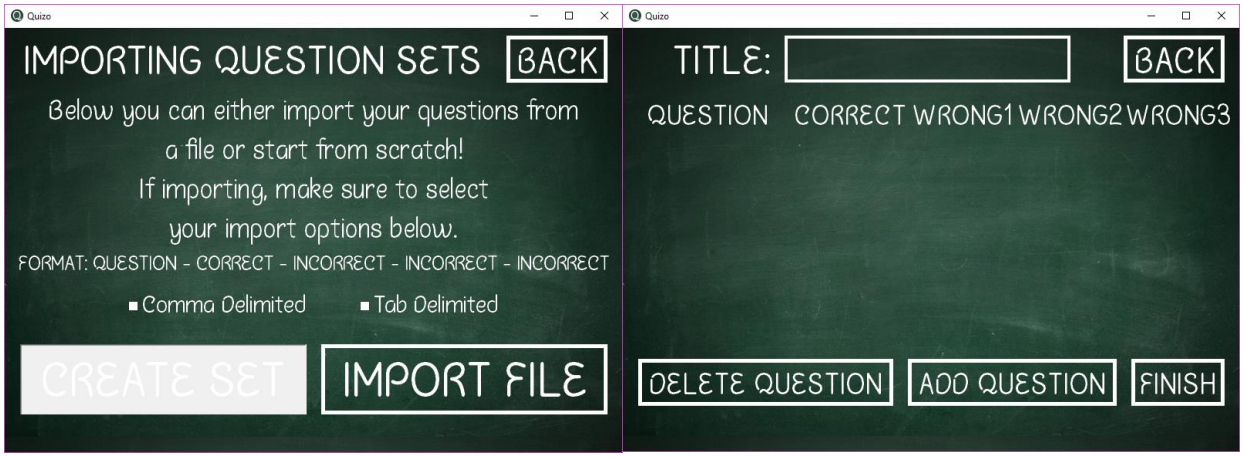
```
            query = '''SELECT Questions.Question,Questions.Correct,Questions.Wrong1,Questions.Wrong2,Questions.Wrong3
FROM Questions INNER JOIN QSLinks ON Questions.QID=QSLinks.QID WHERE QSLinks.SID=?;''' ## FETCHES QUESTION FOR GIVEN SET
            cur.execute(query,(setdetails[2],)) ## EXECUTES QUERY
            setquestions = cur.fetchall() ## FETCHES ALL QUESTIONS
            AddQuestions.displayquestions(AddQuestions_Window,setdetails[0],setquestions) ## CREATES EMPTY TABLE
            AddQuestions_Window.show() ## SHOWS THE ADD QUESTIONS WINDOW
            Importing_Window.hide() ## HIDES THE IMPORTING WINDOW
```

Added inside edit function in View Sets class to fetch all questions for a selected set, and then pass these details to the add questions class

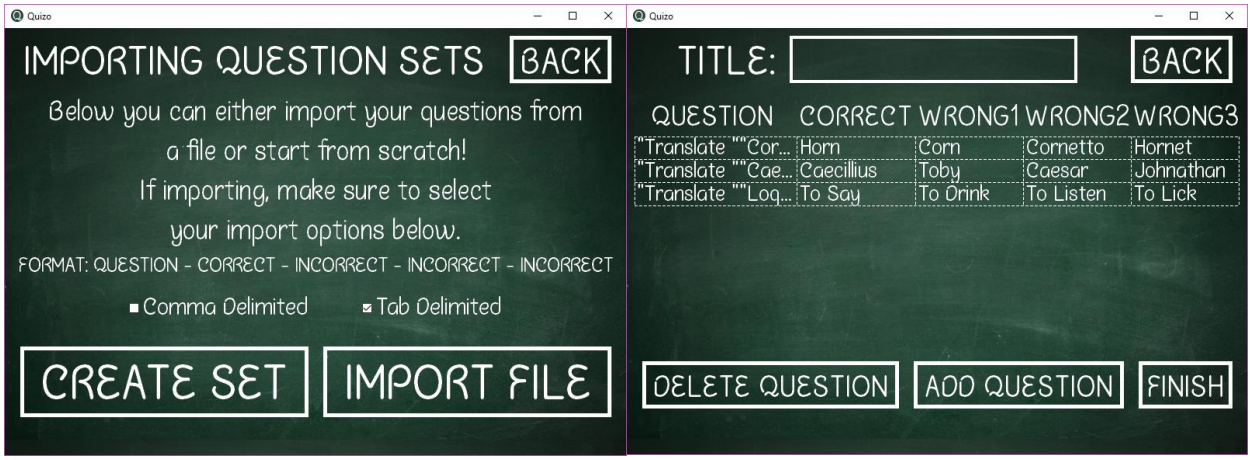
In order to find out whether this new code works as intended, I have done a few tests.



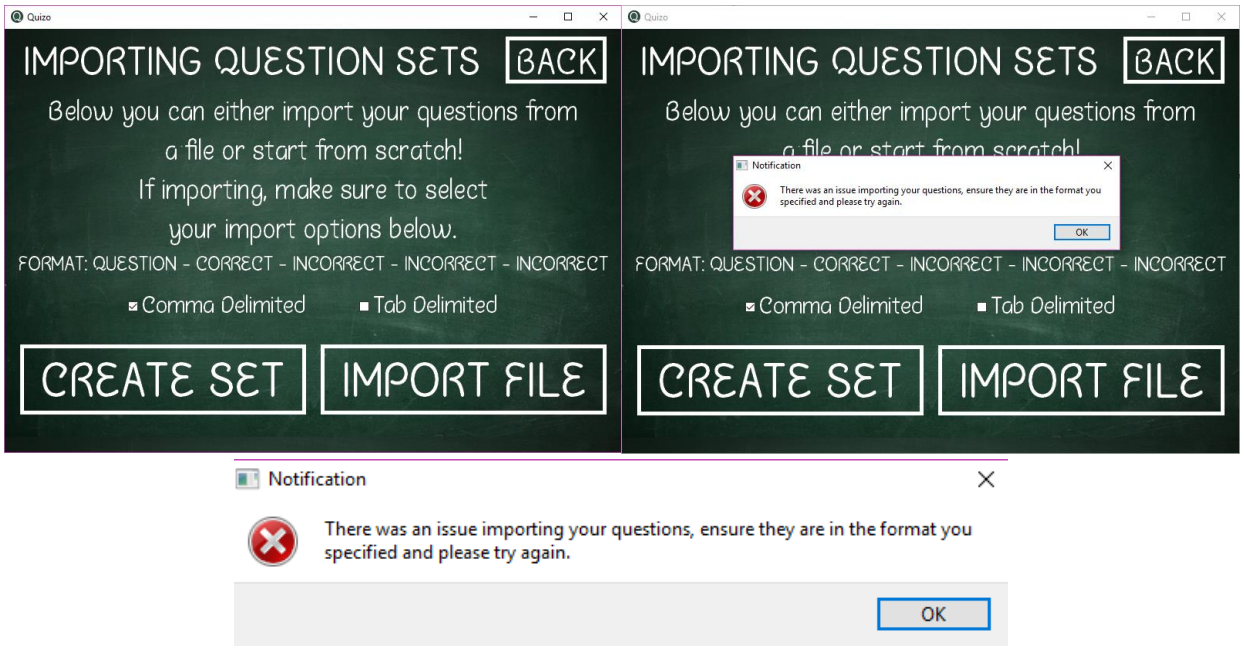
Test 1 – Edit "Entertainment" set – Successful – When trying to edit this set, the program successfully set the title of the set, and displayed all the existing questions in the table.



Test 2 – Create new set – Successful – When trying to create a new set, the program successfully opened the window with a blank title and table.



Test 3 – Import set with correct option – Successful – When trying to import a tab delimited file with the tab delimited option, the questions were correctly displayed in the table, and the title was left blank.



Test 4 – Import set with incorrect option – Successful – When trying to import a tab delimited file with the comma delimited option, the program provided an output explaining that there was an error importing the questions.

ADDING AND DELETING QUESTIONS

Now that the table creation for this window is working, the user needs to be able to add or remove questions from the table. Instead of previous tables in my program, this table allows editing for every cell in the table, so that the user can edit any part of the question set they want, and then at the end when they try to save the quiz, the program can fetch all the data from the table and save that to the database.

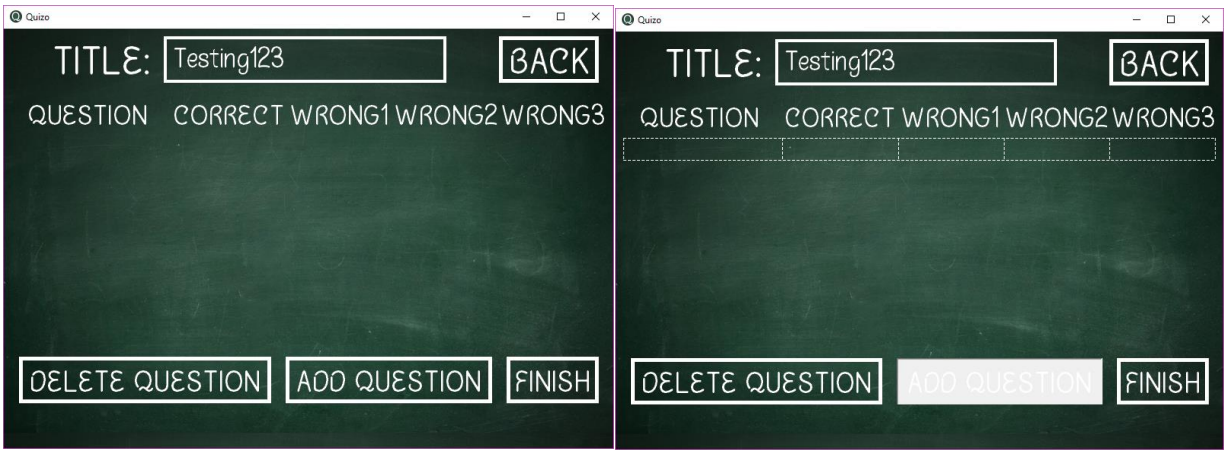
In order to add a question, I have added the "self" prefix to the table model, so that the model can be edited by any of the functions within the class. Now that any function can edit the table model, I have added the code for adding a new row to the table.

```
def add(self):
    self.model.appendRow(QtGui.QStandardItem()) ## ADDS NEW ROW TO TABLE
```

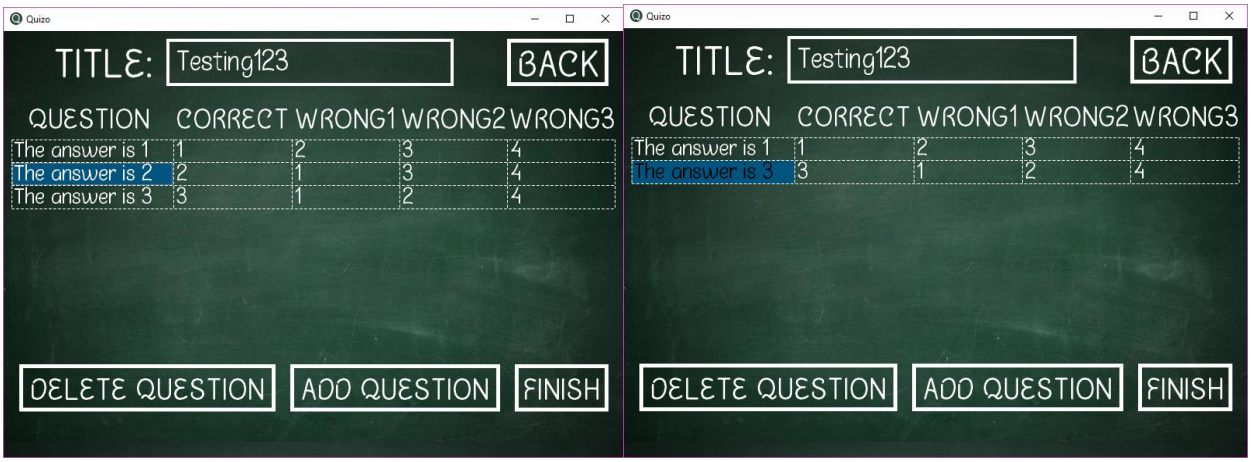
In order to remove a question from the table, the program needs to find the selected question, and then delete that row from the table model. The code for this can be seen below.

```
def delete(self):
    selection = self.questiontable.selectionModel().selectedIndexes() ## FETCHES THE SELECTED ITEMS FROM THE TABLE
    if selection == []: ## CHECKS IF ANY ROW HAS BEEN SELECTED
        CreateOutbox("Notification","You must select a question to remove!","",QtGui.QMessageBox.Critical)
    else:
        selectedrow = selection[0].row() ## GETS ROW NUMBER OF SELECTED ITEM
        self.model.removeRow(selectedrow) ## REMOVES ROW FROM TABLE MODEL
```

Below are tests displaying both of these new function working.



Test 1 – Adding a question – Successful – After pressing the add question button, a new row was added to the table.



Test 2 – Deleting a question – Successful – After adding a few questions to the table and filling them out, I selected the second question and then pressed the delete question button. As can be seen above, the question was then removed from the table, leaving only two questions.

OBJECTIVES 44, AND 47 COMPLETED – TEACHERS CAN ADD OR EDIT QUESTION SETS

WRITING TO THE DATABASE

Now that users can edit, add or import their own questions into this window, they need to be able to write their changes to the database, so that their question sets can be attempted by their students. Due to the way that the database works, editing and adding question sets will work essentially the same way.

When trying to write changes for a question set, the program will first check what the title that the user has entered is. If the user has already created a question set with this title, they will be asked whether they would like to overwrite this question set, or if they would like to rename the set.

If the user decides to overwrite the previous question set, all the old questions and links between the set and question will be deleted, before then adding the new questions and links to the database. If the user hasn't created a set with the same name before, then the same will occur, except a new record will be added to the Sets table with the new set details before then adding the other details.

IMPLEMENTING REGULAR EXPRESSIONS

Before writing all the code for inserting the questions into the database, I have decided that I will implement regular expressions to validate the title entered by the user, rather than using string manipulation like I have done in previous windows.

To test my regular expressions before inserting them into my program, I am using an online tool, *regex101.com*, and these tests can be seen below, showing how I reached the expression I will be using for validating the title of the question sets.

Title requirements:

- ONLY Alphanumerical characters, spaces, or dashes
- Must begin with an alphanumerical character
- No consecutive spaces or dashes (independent of each other)
- Length must be between three and forty characters inclusive

I have written an initial expression to try and accommodate these requirements, however my first attempt has shown problems which can be seen below.

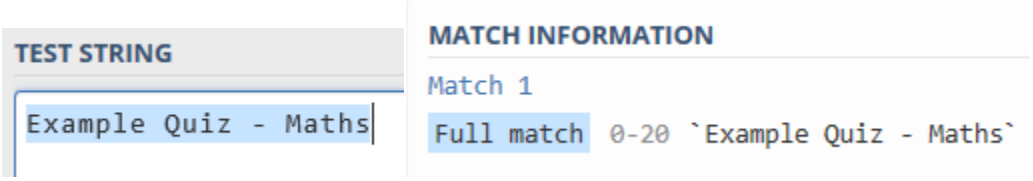
EXPRESSION: /^[\w](?!.*-{2})(?!.* {2})[\w -]{2,39}$/

To explain how I came to this expression, I have explained what each section of it is used for:

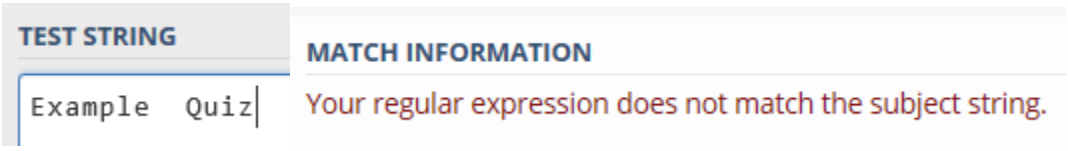
- ^ starts trying to match from the start of the string
- \w is a token for all alphanumerical characters
- ?! is a negative look ahead which acts as a not command, meaning anything within the brackets that is found causes the string not to be matched
- .* is used to match any following character
- {2} is a quantifier that checks if this many or more matches are found in a row
- {2,39} is a quantifier that I am using to check the length of the string (Must be between 3 and 39 matches for [\w -] which are the allowed characters)
- $ denotes the end of the string

Test Number	Test String	Purpose	Expected Result
1	Example Quiz – Maths	Check that when all requirements are met, the string is matched	Match
2	Example Quiz	Check that double spaces are not allowed	No Match
3	Example Quiz -- Maths	Check that double dashes are not allowed	No Match
4	-Example-Quiz	Check that first character must be alphanumerical	No Match
5	Example_Quiz!! – "Topic's?"	Check that punctuation besides dashes are disallowed	No Match
6	Hi	Check that minimum length is validated	No Match
7	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa	Check that maximum length is validated	No Match

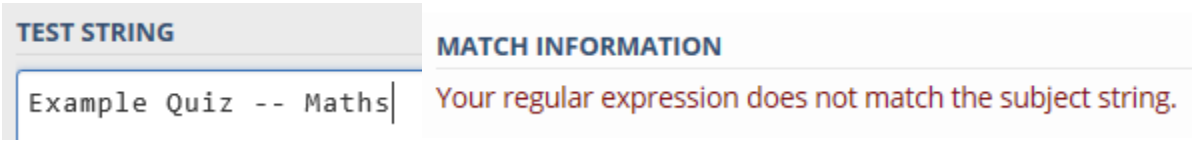
Test 1 – Successful – The expression successfully matched the string



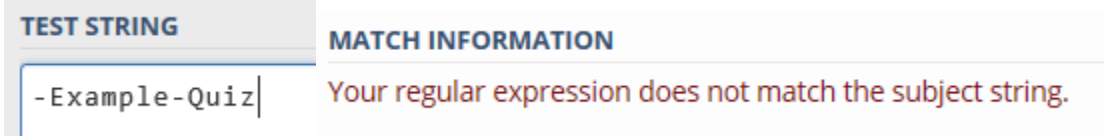
Test 2 – Successful – The expression successfully denied the string due to the double space



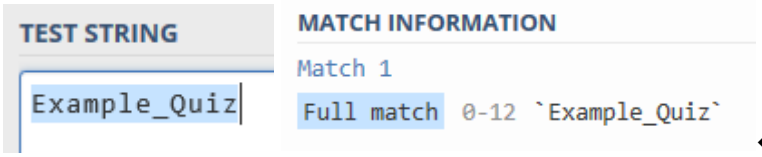
Test 3 –Successful – The expression successfully denied the string due to the double dash



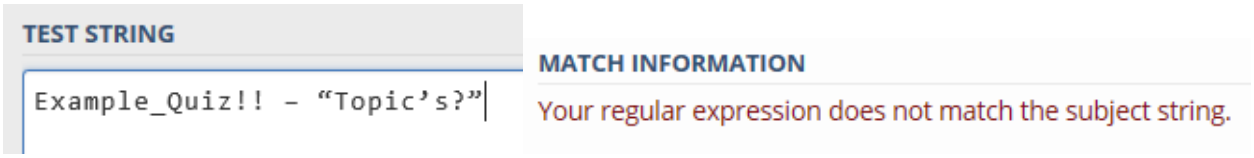
Test 4 – Successful – The expression successfully denied the string due to the first character being a dash rather than being alphanumerical



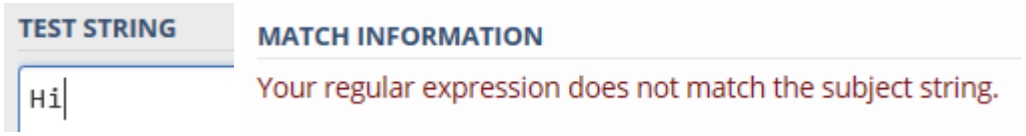
Test 5 – Unsuccessful – While entering the string, I expected the string to stop being matched after the underscore was entered. However, the string continued to match until the first exclamation mark was entered. Having looked at the "\w" token, I now see that this also includes underscores. For this reason, I will replace the \w token in my expression with [A-Z,a-z,0-9] in order to only match alphanumerical characters.



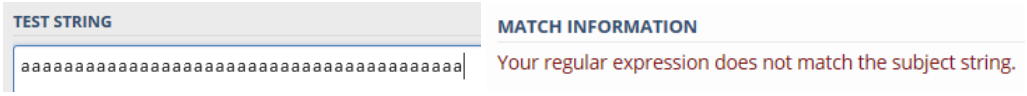
← Underscore included in \w



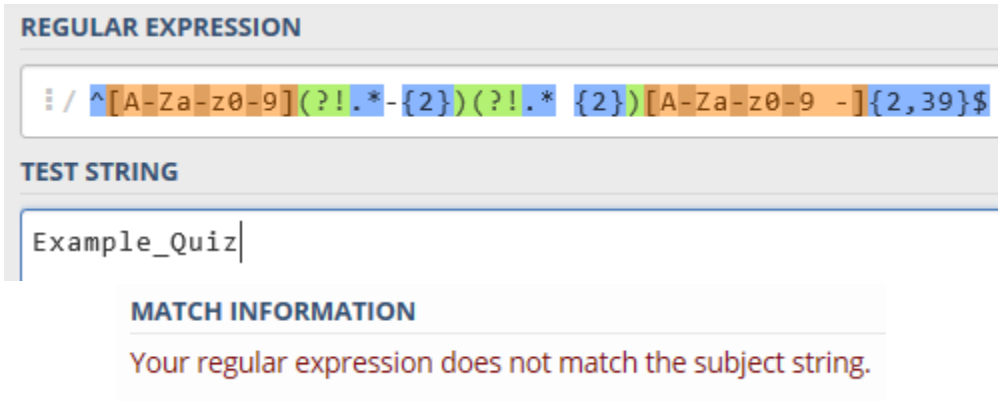
Test 6 – Successful – The expression successfully denied the string due to being less than three characters long



Test 7 – Successful – The expression successfully denied the string due to being more than forty characters long



Due to test 5 being unsuccessful, I have replaced the \w token in my expression with [A-Za-z0-9] to ensure that underscores are not allowed in the title, and I have used the same string in test 5 to check whether this new expression is working correctly. As can be seen below, this change has fixed the issue.



Now that I have checked that my expression is working correctly, I can incorporate it into my program. To do this, I have imported the "re" module into my program which allows me to check matchings when using regular expressions.

The code for checking whether the entered string matches the regex pattern is written below, and the value of the variable "validtitle" is different depending on whether there is a match or not; if the string matches the pattern, the variable will be set to a match object, whereas if there is no match the variable will just be set to None. Due to this, I will just have to check whether the value is None, as can be seen below.

```
import re

def regex():
    title = input("Enter title:")
    validtitle = re.match("[A-Za-z0-9](?!.*-{2})(?!.* {2})[A-Za-z0-9 -]{2,39}$",title)
    print(validtitle)
    if validtitle:
        print("MATCHES PATTERN")
    else:
        print("INVALID")
    regex()
```

I have done a quick demonstration of this regular expression, doing three of my previous tests, showing that the module is working as intended in a separate file which can be seen below.

```
Enter title:Example Quiz - Maths
<_sre.SRE_Match object; span=(0, 20), match='Example Quiz - Maths'>
MATCHES PATTERN
Enter title:Example_Quiz
None
INVALID
Enter title:Example  Quiz
None
INVALID
```

WRITING TO THE DATABASE CONTINUED

Now that I can use regular expressions in my program, I can fetch and write all the necessary data to the database to add or edit question sets. I have written some preliminary code below which will check whether the user has already created a question set with the same name, and if so will ask whether they want to overwrite the previous quiz. If they do wish to overwrite the data, the program will delete all the old questions, links, scores, and the set from the database, before adding the new data.

```
    def finish(self):
        title = self.titleedit.text() ## GET TEXT FROM TITLE ENTRY BOX
        validtitle = re.match("^[A-Za-z0-9](?!.*-{2})(?!.* {2})[A-Za-z0-9 -]{2,39}$",title) ## USES REGULAR EXPRESSION TO CHECK IF VALID
        if not validtitle:
            CreateOutbox("Invalid Title",
                         '''The title you have entered is invalid. All titles must:","Contain only alphanumerical characters, spaces, or dashes
\nHave no double dashes
\nHave no double spaces
\nBegin with an alphanumerical character
\nBe between 3 and 40 characters in length''',QtGui.QMessageBox.Critical)

        else:
            query = 'SELECT SID,SetName,Author FROM Sets WHERE SetName=? AND Author=?;' ## CHECKS IF USER HAS PREVIOUSLY MADE A SET WITH THIS NAME
            cur.execute(query,(title,cur_user,)) ## EXECUTES QUERY
            setdetails = cur.fetchall() ## FETCHES ANY FOUND DETAILS
            if setdetails != []: ## CHECKS IF A SET WAS FOUND
                msg = QtGui.QMessageBox() ## CREATES A MESSAGE BOX
                msg.setStandardButtons(QtGui.QMessageBox.Ok | QtGui.QMessageBox.Cancel) ## SETS BUTTONS FOR BOX
                msg.setWindowTitle("Overwrite Quiz") ## SETS TITLE
                msg.setIcon(QtGui.QMessageBox.Critical) ## SETS ICON
                msg.setText("You already have a set with this title, would you like to replace the existing questions?") ## SETS TEXT
                if msg.exec() == QtGui.QMessageBox.Ok: ## CHECKS IF OK WAS PRESSED
                    query = 'DELETE FROM Questions WHERE Questions.QID IN (SELECT QSLinks.QID FROM QSLinks WHERE QSLinks.SID=?);' ## DELETES OLD QUESTIONS
                    cur.execute(query,(setdetails[0][0],))
                    query = 'DELETE FROM Scores WHERE SID=?;' ## DELETES ANY OLD SCORES
                    cur.execute(query,(setdetails[0][0],))
                    query = 'DELETE FROM QSLinks WHERE QSLinks.SID=?;' ## DELETES OLD LINKS
                    cur.execute(query,(setdetails[0][0],))
                    query = 'DELETE FROM Sets WHERE SID=?;' ## DELETES OLD SET
                    cur.execute(query,(setdetails[0][0],))
                    con.commit() ## COMMITS CHANGES TO DATABASE
                    self.writechanges(title)
            else:
                self.writechanges(title)
```

As can be seen in the previous code, whether the user is overwriting an old set or creating a new one, they will be redirected to the same section of code that will allow the program to write all their questions to the database. When trying to add questions to the database, the program will need to do multiple things.

First the program will need to gather all the questions from the table and generate a list of questions from this. Due to the possibility of having issues when trying to add the data to the database, I have decided that the program will remove all apostrophes, quote marks, and semi-colons from all data being input as standard.

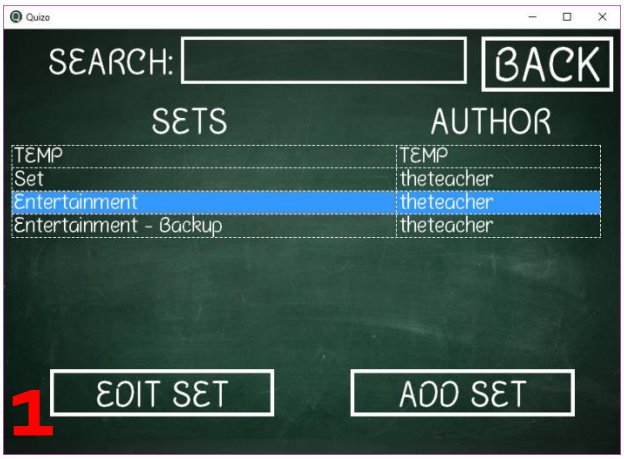
It will then need to find a unique set id for the new question set, as well as unique question ids for each question being added. Instead of finding the number of sets and questions and just adding onto that to find new and unique values, I will be iterating through ids to find ones that haven't been taken.

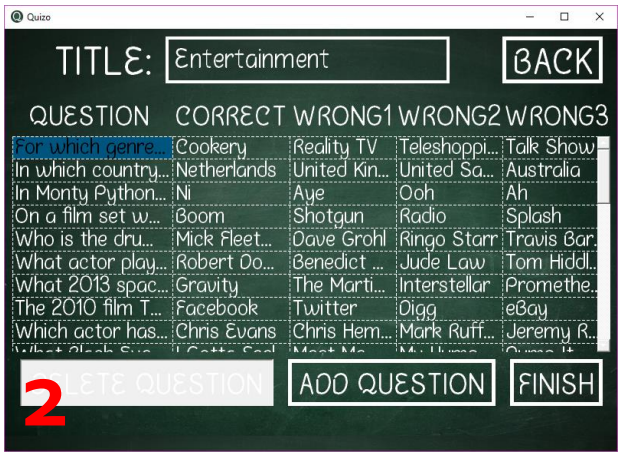
The code for my solution to these can be seen below, alongside some tests to check that all the functionality is working as intended.

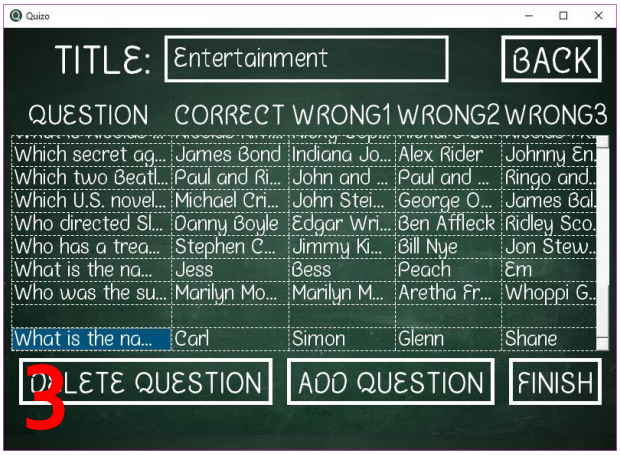
```
def writechanges(self,title):
    questions = [] ## LIST OF QUESTIONS TO WRITE TO DATABASE
    for row in range(self.model.rowCount()): ## LOOPS THROUGH EACH ROW IN TABLE
        tempquestion = [] ## MAKES A TEMPORARY LIST FOR CURRENT ROW
        for column in range(5): ## LOOPS THROUGH EACH COLUMN IN TABLE
            index = self.model.index(row,column) ## FINDS INDEX OF CURRENT CELL
            item = str(self.model.data(index)) ## GETS TEXT FROM CURRENT CELL
            item = item.replace('"','').replace("'","").replace(";","") ## REMOVES ANY APOSTROPES,QUOTE MARKS, OR SEMI-COLONS FOR VALIDATION
            tempquestion.append(item) ## ADDS CELL DATA TO TEMPORARY LIST
        if ("None" not in tempquestion) and ("" not in tempquestion): ## CHECKS IF ROW HAS ANY INCOMPLETE DATA
            questions.append(tempquestion) ## ADDS QUESTION TO LIST OF QUESTIONS

    query = "SELECT SID FROM Sets;" ## FETCHES ALL SET IDS FROM DATABASE
    cur.execute(query) ## EXECUTES QUERY
    allsids = [sub[0] for sub in cur.fetchall()] ## LIST OF ALL SET IDS IN A USABLE FORMAT
    checkid = 1 ## SETS STARTING ID FOR FINDING AN ID FOR THE NEW SET
    allocated = False
    while allocated == False: ## CHECKS IF AN ID HAS BEEN FOUND FOR THE SET
        if checkid not in allsids: ## CHECKS IF THE CURRENT ID IS ALREADY TAKEN
            setid = str(checkid) ## SETS THE ID OF THE SET
            query = "INSERT INTO Sets VALUES (?,?,?);" ## ADDS NEW SET TO DATABASE
            cur.execute(query,(setid,title,cur_user,)) ## EXECUTES QUERY
            con.commit() ## WRITES CHANGES TO DATABASE
            allocated = True ## PREVENTS FURTHER SEARCHING FOR AN ID
        else:
            checkid += 1 ## INCREMENTS THE ID TO CHECK

    query = "SELECT QID FROM Questions;" ## FETCHES ALL QUESTION IDS FROM DATABASE
    cur.execute(query) ## EXECUTES QUERY
    allqids = [sub[0] for sub in cur.fetchall()] ## LIST OF ALL QUESTION IDS IN A USABLE FORMAT
    checkid = 1 ## SETS STARTING ID FOR FINDING AN ID FOR THE QUESTIONS
    for question in questions: ## LOOPS THROUGH EACH QUESTION THAT NEEDS TO BE ADDED TO THE DATABASE
        allocated = False
        while allocated == False: ## CHECKS IF AN ID HAS BEEN FOUND FOR THIS QUESTION YET
            if checkid not in allqids:
                query = "INSERT INTO Questions VALUES (?,?,?,?,?,?);" ## ADDS QUESTION TO DATABASE
                cur.execute(query,(str(checkid),question[0],question[1],question[2],question[3],question[4],)) ## EXECUTES QUERY
                query = "INSERT INTO QSLinks VALUES (?,?);" ## ADDS LINK TO DATABASE
                cur.execute(query,(setid,str(checkid),)) ## EXECUTED QUERY
                con.commit() ## WRITES CHANGES TO DATABASE
                allocated = True ## PREVENTS FURTHER SEARCHING FOR AN ID FOR THIS QUESTION
            checkid += 1 ## INCREMENTS THE ID TO CHECK
    CreateOutbox("Question Set Added","Your questions have been added to the database!","",QtGui.QMessageBox.Information)
    ViewSets.search(ViewSets_Window) ## UPDATES THE TABLE BEFORE THE WINDOW OPENS
    ViewSets_Window.show() ## SHOWS THE VIEW SETS WINDOW
    AddQuestions_Window.hide() ## HIDES THE ADD QUESTIONS WINDOW
```

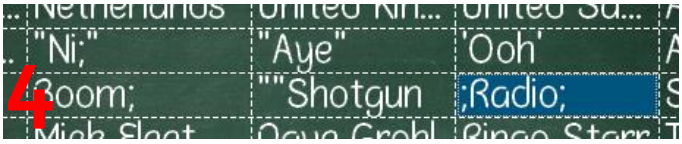
Test 1 – Making a copy of the "Entertainment" Set – Successful – Before making any major changes to the database, I selected the "Entertainment" Set and pressed the edit button. When I got to the next window, I changed the title of the set to "Entertainment – Backup" and pressed finish. When I was returned to the view sets window, the new set was shown, and when pressing edit for this set, the exact same questions were shown, making the copy successful.

Test 2 – Editing the "Entertainment" Set – Successful – To test editing a set, I have selected to edit the "Entertainment" set[1], and I have made several changes to see if the program successfully makes the amendments. I have removed the first question from the set[2], added an additional two questions[3], one of which I filled out and one I left blank. I then also added quote marks, apostrophes, and semi-colons to some of the cells[4], and then pressed the finish button[5]. The program successfully displayed a message box saying the update was complete[6], and when I went to edit the set again[7], all the changes had been made successfully[8].

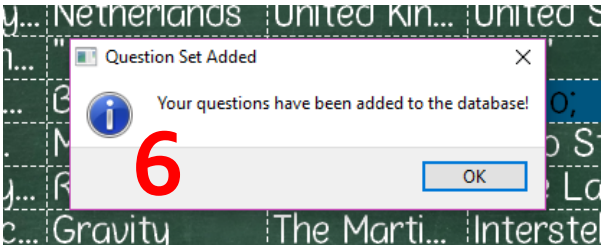

Quizo
SEARCH:
BACK
SETS
AUTHOR
TEMP
TEMP
Set
theteacher
Entertainment
theteacher
Entertainment - Backup
theteacher
EDIT SET
ADD SET
1

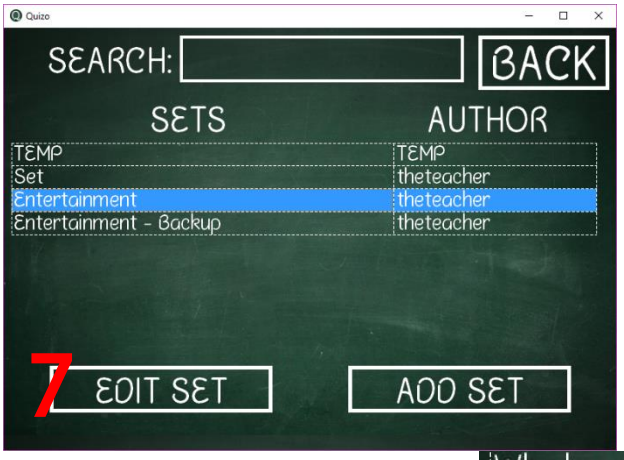

Quizo
TITLE:
Entertainment
BACK
QUESTION
CORRECT
WRONG1
WRONG2
WRONG3
For which genre...
Cookery
Reality TV
Teleshoppi...
Talk Show
In which country...
Netherlands
United Kin...
United Sa...
Australia
In Monty Python...
Ni
Aye
Ooh
Ah
On a film set w...
Boom
Shotgun
Radio
Splash
Who is the dru...
Mick Fleet...
Dave Grohl
Ringo Starr
Travis Bar...
What actor play...
Robert Do...
Benedict ...
Jude Law
Tom Hiddl...
What 2013 spac...
Gravity
The Marti...
Interstellar
Promethe...
The 2010 film T...
Facebook
Twitter
Digg
eBay
Which actor has...
Chris Evans
Chris Hem...
Mark Ruff...
Jeremy R...
ADD QUESTION
FINISH
2

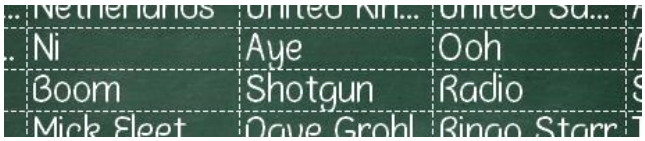

Quizo
TITLE:
Entertainment
BACK
QUESTION
CORRECT
WRONG1
WRONG2
WRONG3
Which secret ag...
James Bond
Indiana Jo...
Alex Rider
Johnny En...
Which two Beatl...
Paul and Ri...
John and ...
Paul and ...
Ringo and...
Which U.S. novel...
Michael Cri...
John Stei...
George O...
James Bal...
Who directed Sl...
Danny Boyle
Edgar Wri...
Ben Affleck
Ridley Sco...
Who has a trea...
Stephen C...
Jimmy Ki...
Bill Nye
Jon Stew...
What is the na...
Jess
Bess
Peach
Em
Who was the su...
Marilyn Mo...
Marilyn M...
Aretha Fr...
Whoppi G...
What is the na...
Carl
Simon
Glenn
Shane
DELETE QUESTION
ADD QUESTION
FINISH
3


"Ni;"
"Aye"
'Ooh'
Boom;
""Shotgun
;Radio;
4


Overwrite Quiz
You already have a set with this title, would you like to replace the existing questions?
OK
Cancel
5


Question Set Added
Your questions have been added to the database!
OK
6


Quizo
SEARCH:
BACK
SETS
AUTHOR
TEMP
TEMP
Set
theteacher
Entertainment
theteacher
Entertainment - Backup
theteacher
EDIT SET
ADD SET
7

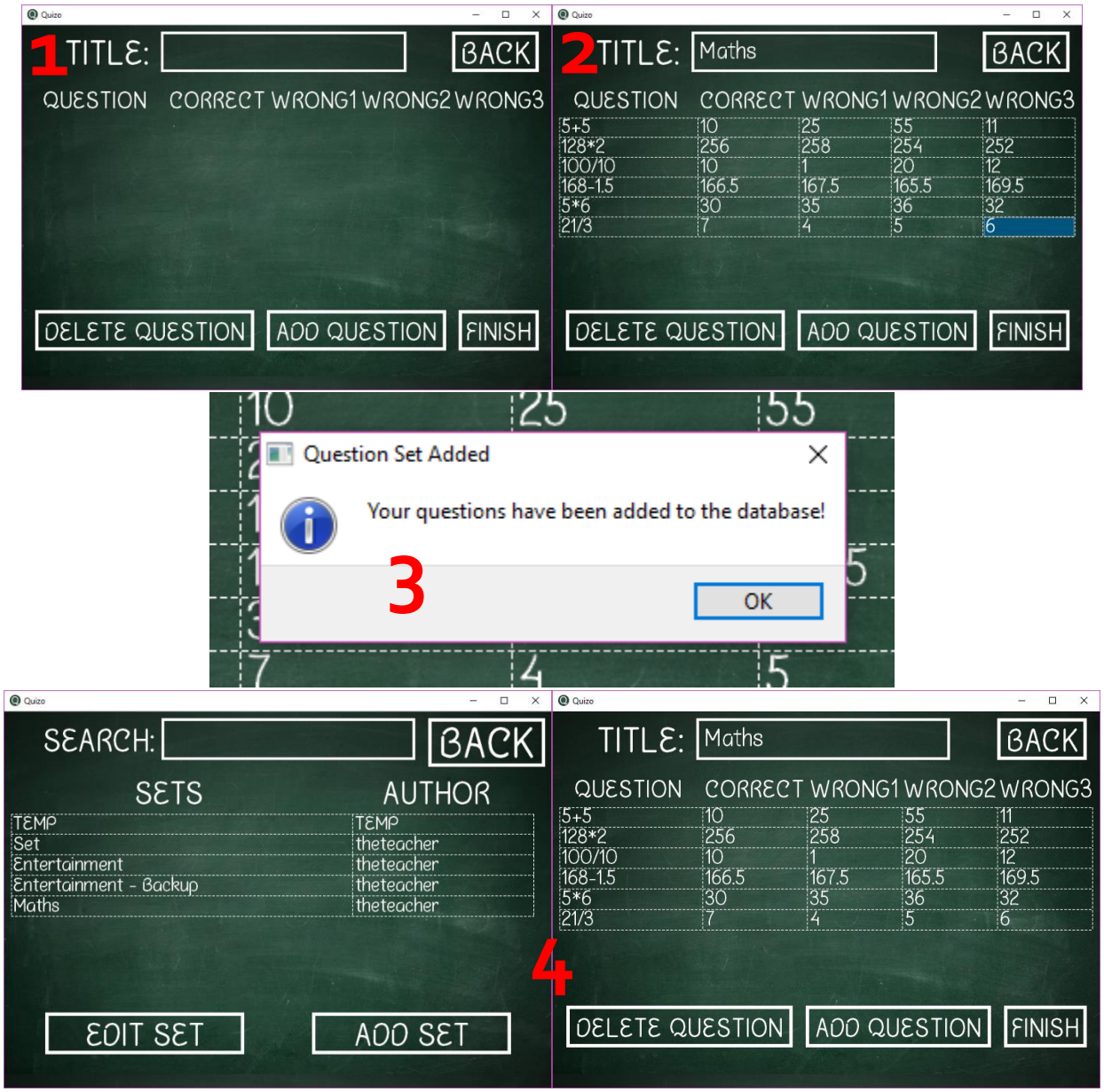

Ni
Aye
Ooh
Boom
Shotgun
Radio

8


Who has a trea...
Stephen C...
Jimmy Ki...
Bill Nye
Jon Stew...
What is the na...
Jess
Bess
Peach
Em
Who was the su...
Marilyn Mo...
Marilyn M...
Aretha Fr...
Whoppi G...
What is the na...
Carl
Simon
Glenn
Shane

Test 3 – Editing the "Entertainment" Set but not overwriting – Successful – I made a few changes to the set as can be seen below[1], and when the program asked if I wanted to overwrite the previous set, I pressed cancel[2]. As expected, no changes were made to the database[3].

Test 4 – Creating a new set – Successful – I decided to create a new set[1], labelling it the "Maths" set, and when I added a few questions[2] and pressed finish[3], the program successfully added my new set to the database[4].



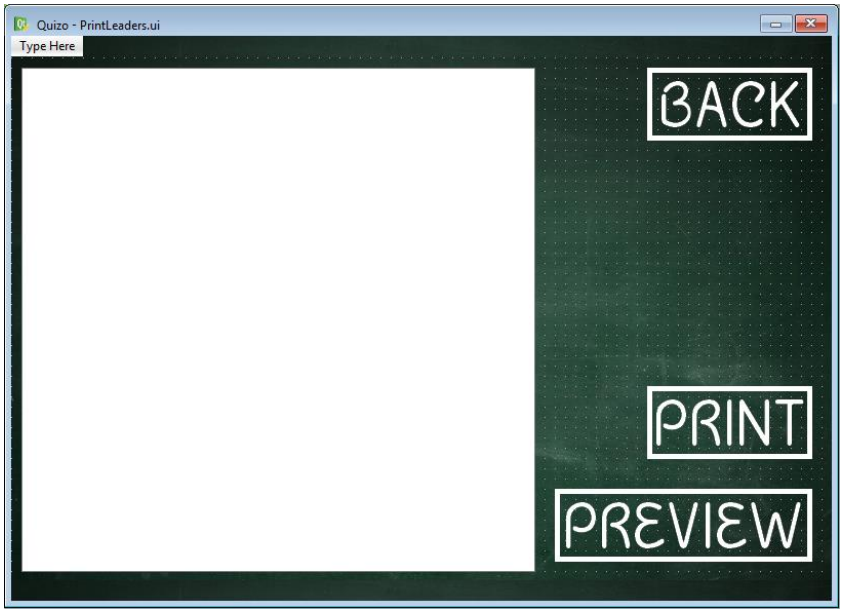
Given that my above tests were all successful, I can safely say that this window is now feature complete and I can move onto other sections of the program.

OBJECTIVES 46, AND 48 COMPLETED – USER CAN UPLOAD NEW QUESTIONS TO THE DATABASE BY IMPORTING FROM A FILE, EDITING EXISTING SETS, OR CREATING NEW ONES FROM SCRATCH

LEADERBOARD PRINTING

From my initial interviews with members of the RGS, they liked the competitive aspect of the program, and being able to see how they are progressing against their peers. Although the program has a top 10 leader board, the aim is to have more than ten students competing in quizzes, and so a leader board showing all users partaking in a quiz is necessary. Once this leader board is created, a good way of displaying it is by being able to print it out, and so I will add this functionality.

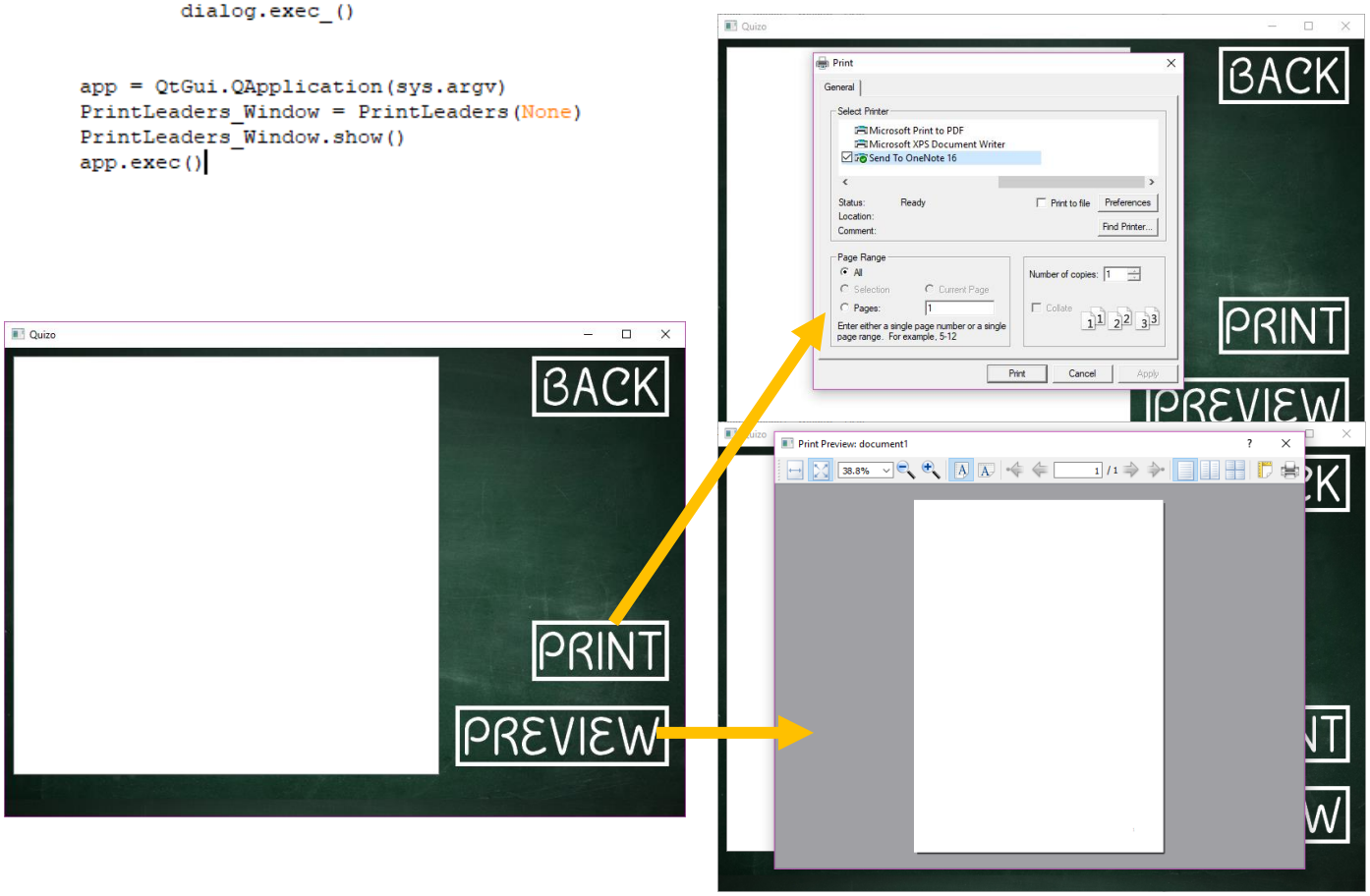
Since the leader board will be able to be printed, I have decided that I will not be using any kind of background, and that it will just be a black and white, easy to read table. Having done a bit of research into printing, I have decided that I will take all the results, and create a table in a text editor, because PyQt is able to print the contents of a text edit widget directly.



To the side, I have created a window for this leader board, and as shown, the window will only consist of the text editor and three buttons. The editor is read only as the user has no need to edit the leader board in any way, and I have set the font to size 11 as I believe that is large enough to be able to read, while still being able to display many results in the table.

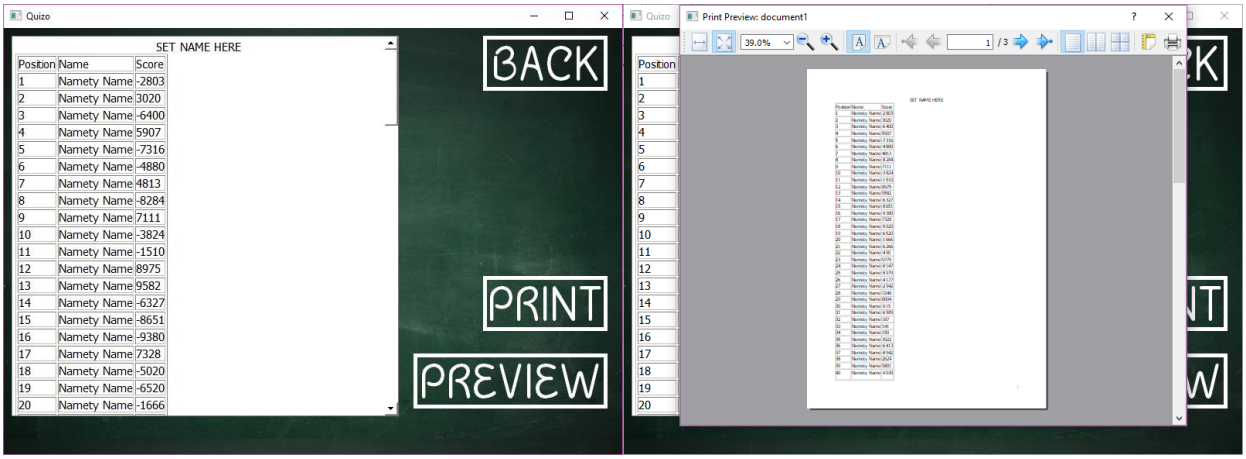
Before trying to implement this window into my program, I have decided that I will make a separate file to test the formatting and printing techniques, before adding it into my program. In the code below, I have created functions for both printing and previewing, using their respective dialog boxes from PyQt. I have also done a quick test below to check that both these dialog boxes appear when their respective buttons are pressed.

```
import sys,os
from PyQt4 import QtCore,QtGui,uic

PrintLeaders_class = uic.loadUiType("PrintLeaders.ui")[0] ## LOADS UI FILE

class PrintLeaders(QtGui.QMainWindow,PrintLeaders_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.setupUi(self)
        self.printButton.clicked.connect(self.printing)
        self.previewButton.clicked.connect(self.preview)
        self.backButton.clicked.connect(self.Back)

    def Back(self):
        print("BACK")

    def printing(self):
        dialog = QtGui.QPrintDialog() ## CREATES THE PRINT DIALOG BOX
        if dialog.exec_() == QtGui.QDialog.Accepted:
            self.textEdit.document().print_(dialog.printer()) ## GETS TEXT FROM TEXT EDIT FOR PRINTING

    def preview(self):
        dialog = QtGui.QPrintPreviewDialog() ## CREATES THE PREVIEW DIALOG BOX
        dialog.paintRequested.connect(self.textEdit.print_) ## GETS PRINT FROM TEXT EDIT TO MAKE A PREVIEW
        dialog.exec_()


app = QtGui.QApplication(sys.argv)
PrintLeaders_Window = PrintLeaders(None)
PrintLeaders_Window.show()
app.exec()
```



Now that I know the window can be displayed correctly, and the dialog boxes are working, I can start inserting data into a table in the editor. To do this, I have to create a table using the "insertTable" function, and then go through each cell, filling it in with data. Since this is a separate window, I will be using the random function to generate random scores, and everyone's name will be identical, but it should be enough to show whether my initial code works or not.

```
def displaytable(self):
    self.cursor = QtGui.QTextCursor()
    self.cursor = self.textEdit.textCursor()
    self.cursor.insertText("SET NAME HERE")
    self.cursor.insertTable(100+1,3)
    self.cursor.insertText("Position")
    self.cursor.movePosition(self.cursor.NextCell)
    self.cursor.insertText("Name")
    self.cursor.movePosition(self.cursor.NextCell)
    self.cursor.insertText("Score")
    self.cursor.movePosition(self.cursor.NextCell)

    position = 1
    for i in range(100):
        self.cursor.insertText(str(position))
        self.cursor.movePosition(self.cursor.NextCell)
        self.cursor.insertText("Namety Name")
        self.cursor.movePosition(self.cursor.NextCell)
        self.cursor.insertText(str(random.randint(-10000,10000)))
        self.cursor.movePosition(self.cursor.NextCell)
        position += 1
```



As can be seen above, the code to create the table worked as intended, however when looking at the print preview, it looked like the way I had formatted the page was wasting a lot of space. Due to this, I have decided that the printout would look better with two columns, wasting less space, and being able to display more users on a single page. To do this, I will create a table with seven columns instead of three, with the fourth column being a divider. The code below is my first attempt at making this, however as can be seen there is a slight issue with it in its current state.

```
titleformat = QtGui.QTextCharFormat()
titleformat.setFontUnderline(True)
titleformat.setFontPointSize(30)
tableformat = QtGui.QTextTableFormat()
tableformat.setAlignment(QtCore.Qt.AlignHCenter)

self.cursor.insertText("SET NAME HERE",titleformat)
self.cursor.insertTable(101,7,tableformat)
self.cursor.insertText("Position")
self.cursor.movePosition(self.cursor.NextCell)
self.cursor.insertText("Name")
self.cursor.movePosition(self.cursor.NextCell)
self.cursor.insertText("Score")
self.cursor.movePosition(self.cursor.NextCell)
self.cursor.insertText("\t")
self.cursor.movePosition(self.cursor.NextCell)
self.cursor.insertText("Position")
self.cursor.movePosition(self.cursor.NextCell)
self.cursor.insertText("Name")
self.cursor.movePosition(self.cursor.NextCell)
self.cursor.insertText("Score")
self.cursor.movePosition(self.cursor.NextCell)

position = 1
for i in range(100):
    self.cursor.insertText(str(position))
    self.cursor.movePosition(self.cursor.NextCell)
    self.cursor.insertText("Namety Name")
    self.cursor.movePosition(self.cursor.NextCell)
    self.cursor.insertText(str(random.randint(-10000,10000)))
    self.cursor.movePosition(self.cursor.NextCell)
    self.cursor.movePosition(self.cursor.NextCell)
    self.cursor.insertText(str(position+100))
    self.cursor.movePosition(self.cursor.NextCell)
    self.cursor.insertText("Namety Name")
    self.cursor.movePosition(self.cursor.NextCell)
    self.cursor.insertText(str(random.randint(-10000,10000)))
    self.cursor.movePosition(self.cursor.NextCell)
    position += 1
```

SET NAME HERE

Position	Name	Score		Position	Name	Score
1	Namety Name	-273		101	Namety Name	-180
2	Namety Name	-5369		102	Namety Name	7643
3	Namety Name	-7450		103	Namety Name	-5755
4	Namety Name	-3258		104	Namety Name	-8509
5	Namety Name	-6263		105	Namety Name	8347
6	Namety Name	5465		106	Namety Name	-6904
7	Namety Name	3616		107	Namety Name	-8749
8	Namety Name	3943		108	Namety Name	1710
9	Namety Name	1226		109	Namety Name	408
10	Namety Name	8857		110	Namety Name	-4170
11	Namety Name	-9169		111	Namety Name	5941
12	Namety Name	7311		112	Namety Name	4370
13	Namety Name	5721		113	Namety Name	4660
14	Namety Name	-7444		114	Namety Name	2173
15	Namety Name	-59		115	Namety Name	4211
16	Namety Name	-3665		116	Namety Name	3439
17	Namety Name	-5807		117	Namety Name	-8499
18	Namety Name	-7916		118	Namety Name	8655
19	Namety Name	-9747		119	Namety Name	-170
20	Namety Name	-4331		120	Namety Name	8040
21	Namety Name	8335		121	Namety Name	-400
22	Namety Name	6585		122	Namety Name	-4858
23	Namety Name	7444		123	Namety Name	-2078
24	Namety Name	9561		124	Namety Name	-1826
25	Namety Name	5277		125	Namety Name	-2582
26	Namety Name	-4006		126	Namety Name	2284
27	Namety Name	-4530		127	Namety Name	-5784
28	Namety Name	5549		128	Namety Name	-802
29	Namety Name	1480		129	Namety Name	5876
30	Namety Name	-6610		130	Namety Name	7149
31	Namety Name	5560		131	Namety Name	-2959
32	Namety Name	3936		132	Namety Name	-8697
33	Namety Name	1857		133	Namety Name	9072
34	Namety Name	-5035		134	Namety Name	1849
35	Namety Name	-7385		135	Namety Name	-9758
36	Namety Name	-5510		136	Namety Name	4846
37	Namety Name	-4681		137	Namety Name	8740
38	Namety Name	-1989		138	Namety Name	-2061
39	Namety Name	817		139	Namety Name	-376

As shown above, the issue with the current table is that instead of getting to the end of a page and then returning to the second "column" on the page, the table goes all the way down the left side for multiple pages, and then it moves to the right side. From this first test, when looking at the first page, the number of rows of users displayed is 39, and then for every page after that there are 42 rows. Using this, I can now fix the table as shown below.

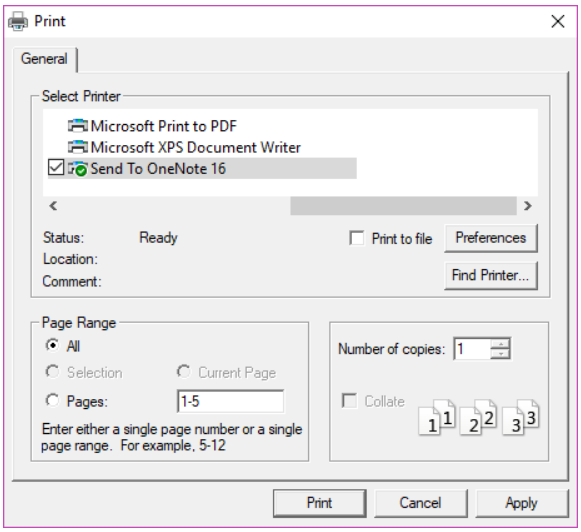
```
def displaytable(self):
    self.cursor = QtGui.QTextCursor()
    self.cursor = self.textEdit.textCursor()

    titleformat = QtGui.QTextCharFormat()
    titleformat.setFontUnderline(True)
    titleformat.setFontPointSize(30)
    tableformat = QtGui.QTextTableFormat()
    tableformat.setAlignment(QtCore.Qt.AlignHCenter)

    self.cursor.insertText("SET NAME HERE",titleformat)

    studentnum = 400
    scores = []
    for i in range(studentnum+1):
        scores.append([str(i),"Namety Name",str(random.randint(-100,100))])

    tempnum = studentnum - 78
    pages = 0
    while tempnum > 0:
        tempnum -= 84
        pages += 1
    totalrows = 41 + (pages*42)

    self.cursor.insertTable(totalrows,7,tableformat)
    self.cursor.insertText("Position")
    self.cursor.movePosition(self.cursor.NextCell)
    self.cursor.insertText("Name")
    self.cursor.movePosition(self.cursor.NextCell)
    self.cursor.insertText("Score")
    self.cursor.movePosition(self.cursor.NextCell)
    self.cursor.insertText("\t")
    self.cursor.movePosition(self.cursor.NextCell)
    self.cursor.insertText("Position")
    self.cursor.movePosition(self.cursor.NextCell)
    self.cursor.insertText("Name")
    self.cursor.movePosition(self.cursor.NextCell)
    self.cursor.insertText("Score")
    self.cursor.movePosition(self.cursor.NextCell)
    page = 1
    columnvalue = 1
    rowsperpage = 39
    while studentnum > 0:
        if page == 1:
            lowerbound = columnvalue
            upperbound = lowerbound + rowsperpage
        else:
            lowerbound = columnvalue + 78 + ((page-2)*84)
            upperbound = lowerbound + rowsperpage
        for i in range(lowerbound,upperbound):
            if studentnum > 0:
                self.cursor.insertText(scores[i][0])
                self.cursor.movePosition(self.cursor.NextCell)
                self.cursor.insertText(scores[i][1])
                self.cursor.movePosition(self.cursor.NextCell)
                self.cursor.insertText(scores[i][2])
                self.cursor.movePosition(self.cursor.NextRow)
                if columnvalue == rowsperpage+1:
                    self.cursor.movePosition(self.cursor.NextCell)
                    self.cursor.movePosition(self.cursor.NextCell)
                    self.cursor.movePosition(self.cursor.NextCell)
                    self.cursor.movePosition(self.cursor.NextCell)
                studentnum -= 1
            else:
                break
        if columnvalue == 1:
            columnvalue += rowsperpage
            for i in range(rowsperpage):
                self.cursor.movePosition(self.cursor.PreviousRow)
            self.cursor.movePosition(self.cursor.PreviousCell)
            self.cursor.movePosition(self.cursor.PreviousCell)
        else:
            page += 1
            columnvalue = 1
            rowsperpage = 42
            self.cursor.movePosition(self.cursor.PreviousCell)
            self.cursor.movePosition(self.cursor.PreviousCell)
            self.cursor.movePosition(self.cursor.PreviousCell)
            self.cursor.movePosition(self.cursor.PreviousCell)
```

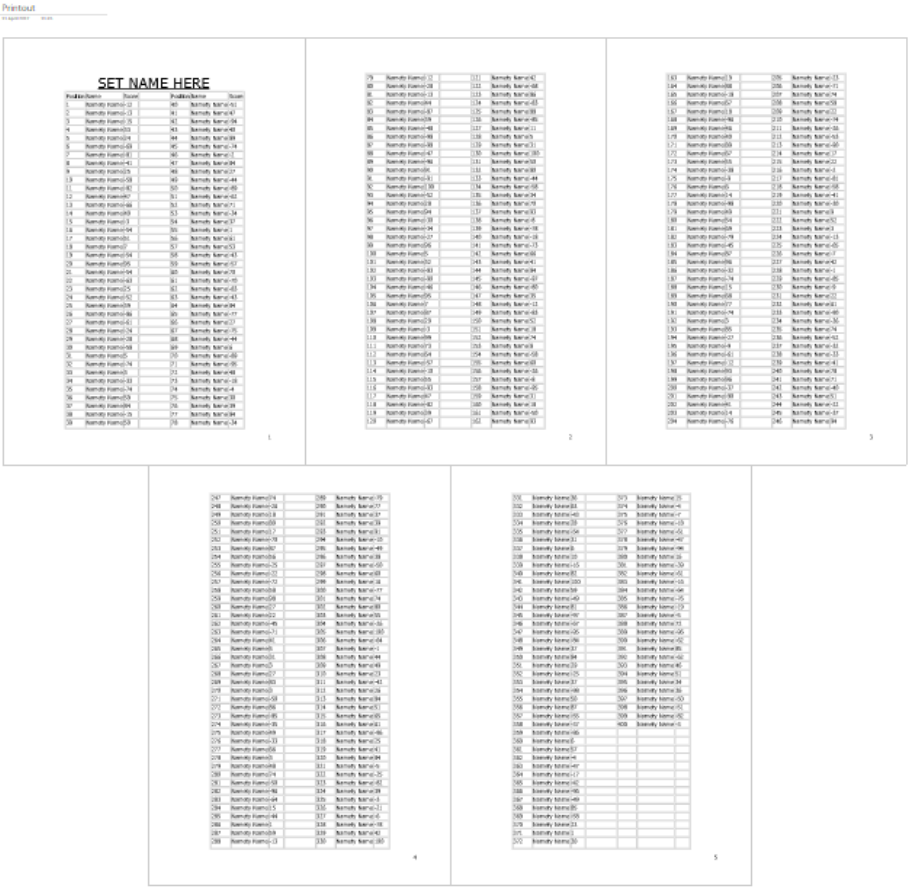
Position	Name	Score		Position	Name	Score
1	Namety Name	-12		40	Namety Name	-51
2	Namety Name	-13		41	Namety Name	47
3	Namety Name	-15		42	Namety Name	-94
4	Namety Name	33		43	Namety Name	40
5	Namety Name	24		44	Namety Name	89
6	Namety Name	-69		45	Namety Name	-74
7	Namety Name	-81		46	Namety Name	-2
8	Namety Name	-41		47	Namety Name	84
9	Namety Name	25		48	Namety Name	27
10	Namety Name	-58		49	Namety Name	-44
11	Namety Name	-82		50	Namety Name	-89
12	Namety Name	47		51	Namety Name	-62
13	Namety Name	-66		52	Namety Name	71
14	Namety Name	40		53	Namety Name	-34
15	Namety Name	-3		54	Namety Name	37
16	Namety Name	-54		55	Namety Name	1
17	Namety Name	51		56	Namety Name	61
18	Namety Name	7		57	Namety Name	53
19	Namety Name	-54		58	Namety Name	-43
20	Namety Name	95		59	Namety Name	-57
21	Namety Name	-54		60	Namety Name	70
22	Namety Name	-63		61	Namety Name	-70
23	Namety Name	25		62	Namety Name	-83
24	Namety Name	-52		63	Namety Name	-43
25	Namety Name	39		64	Namety Name	84
26	Namety Name	-86		65	Namety Name	-77
27	Namety Name	-61		66	Namety Name	27
28	Namety Name	-24		67	Namety Name	-75
29	Namety Name	-28		68	Namety Name	-44
30	Namety Name	-58		69	Namety Name	6
31	Namety Name	5		70	Namety Name	-89
32	Namety Name	-74		71	Namety Name	-95
33	Namety Name	3		72	Namety Name	48
34	Namety Name	-33		73	Namety Name	-18
35	Namety Name	-74		74	Namety Name	-4
36	Namety Name	59		75	Namety Name	30
37	Namety Name	84		76	Namety Name	39
38	Namety Name	-15		77	Namety Name	84
39	Namety Name	50		78	Namety Name	-34

331	Namety Name	36		373	Namety Name	15
332	Namety Name	63		374	Namety Name	-4
333	Namety Name	-43		375	Namety Name	-7
334	Namety Name	28		376	Namety Name	-18
335	Namety Name	-54		377	Namety Name	-61
336	Namety Name	31		378	Namety Name	-47
337	Namety Name	6		379	Namety Name	-94
338	Namety Name	10		380	Namety Name	16
339	Namety Name	-16		381	Namety Name	-39
340	Namety Name	82		382	Namety Name	-61
341	Namety Name	100		383	Namety Name	-16
342	Namety Name	59		384	Namety Name	-64
343	Namety Name	-49		385	Namety Name	-76
344	Namety Name	81		386	Namety Name	-19
345	Namety Name	-97		387	Namety Name	-5
346	Namety Name	-67		388	Namety Name	72
347	Namety Name	-95		389	Namety Name	-96
348	Namety Name	-84		390	Namety Name	-62
349	Namety Name	37		391	Namety Name	85
350	Namety Name	84		392	Namety Name	-62
351	Namety Name	29		393	Namety Name	46
352	Namety Name	-25		394	Namety Name	51
353	Namety Name	37		395	Namety Name	34
354	Namety Name	-98		396	Namety Name	36
355	Namety Name	50		397	Namety Name	-50
356	Namety Name	87		398	Namety Name	-51
357	Namety Name	-55		399	Namety Name	-82
358	Namety Name	-37		400	Namety Name	-3
359	Namety Name	-86				
360	Namety Name	6				
361	Namety Name	57				
362	Namety Name	-4				
363	Namety Name	-47				
364	Namety Name	-17				
365	Namety Name	-42				
366	Namety Name	-96				
367	Namety Name	-49				
368	Namety Name	85				
369	Namety Name	-58				
370	Namety Name	23				
371	Namety Name	1				
372	Namety Name	30				

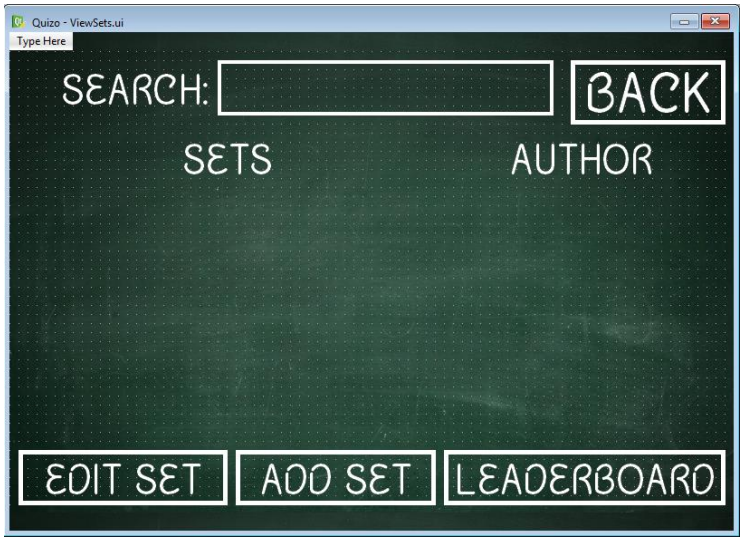
As shown above, when testing this new code with 400 scores, the program creates a large enough table, and displays all scores in the correct format.

Now that the formatting of the table is complete, I will check that when trying to print, the formatting is kept. To test the printing, I will be printing to OneNote, which essentially converts all the pages to jpegs and sends them to a OneNote page.





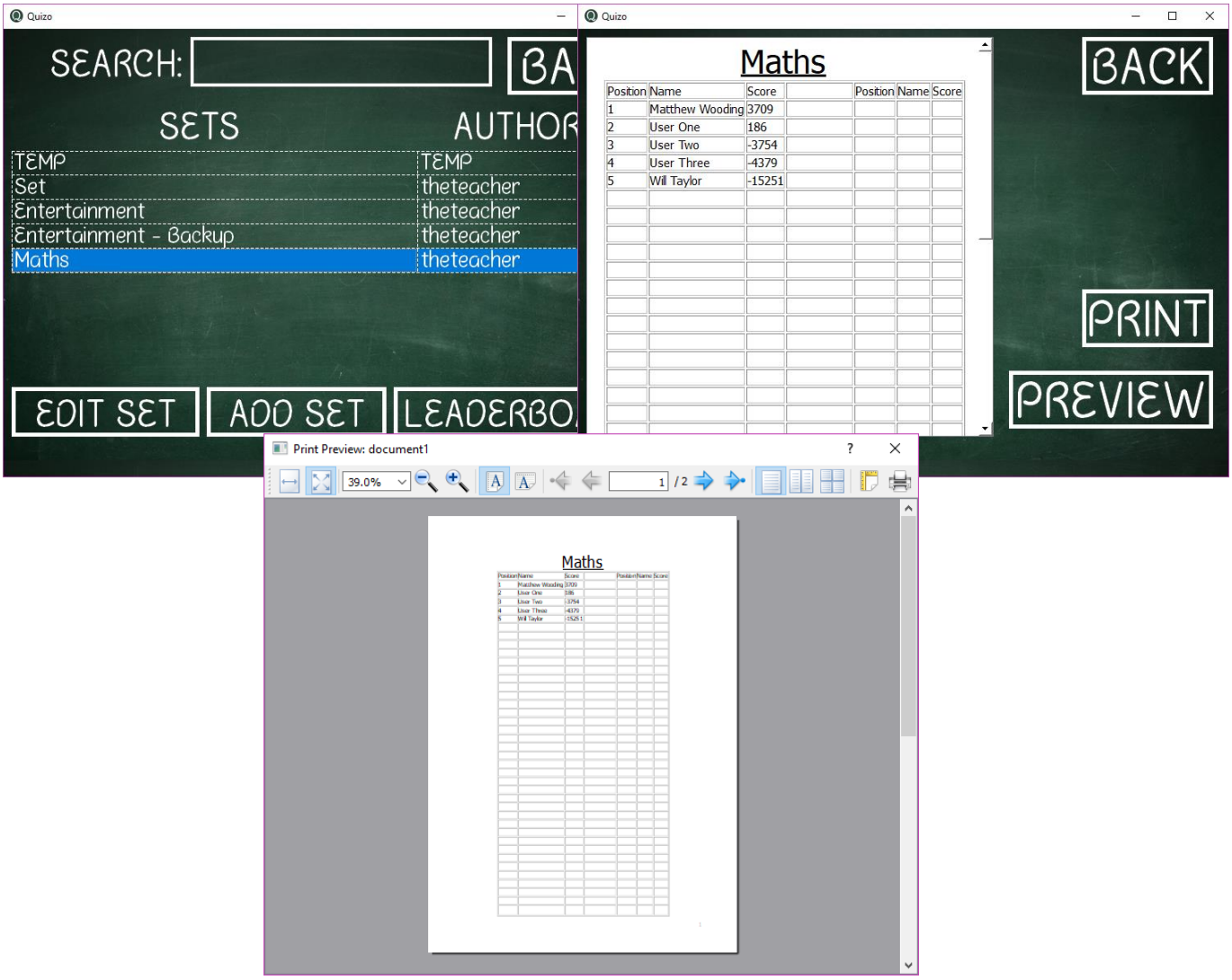
As shown above, the tables were printed successfully, and so this window is now complete. Since I do not currently have a place to put this window in my program, I have gone back into the "View Sets" window and added a new "Leaderboard" button which can be seen below.



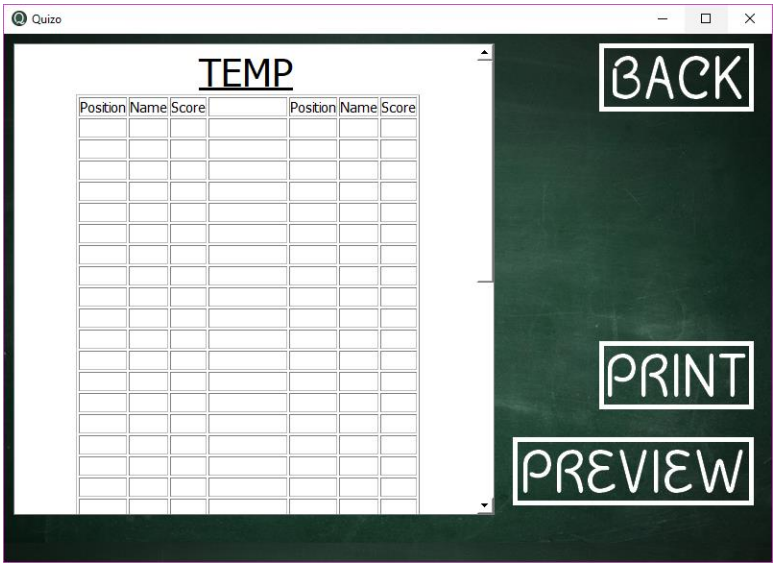
I have now moved the "PrintLeaders" class and window into my main program, and added the necessary navigation code which can be seen below. Additionally, I have replaced the random score generator with the necessary SQLite query to fetch all scores for the chosen set.

```
def leaders(self):
    selection = self.setstable.selectionModel().selectedRows() ## FETCHES THE SELECTED ROWS FROM THE TABLE
    if selection == []: ## CHECKS IF ANY ROW HAS BEEN SELECTED
        CreateOutbox("Selection Not Found","You must select a question set to view the leaderboard!","",QtGui.QMessageBox.Critical)
    else:
        selectedrow = selection[0].row() ## FINDS INDEX OF SELECTED ROW
        setdetails = self.allsets[selectedrow] ## SETS THE CURRENT PLAYSET TO THE SELECTED SET
        PrintLeaders.displaytable(PrintLeaders_Window,setdetails) ## DISPLAYS THE LEADERBOARD
        PrintLeaders_Window.show() ## SHOWS PRINT LEADERS WINDOW
        ViewSets_Window.hide() # HIDES VIEW SETS WINDOW



    def displaytable(self,setdetails):
        self.cursor = self.textEdit.textCursor() ## CREATES CURSOR FOR TEXT EDIT
        titleformat = QtGui.QTextCharFormat() ## CREATES FORMAT FOR THE TITLE
        titleformat.setFontUnderline(True) ## UNDERLINES TITLE
        titleformat.setFontPointSize(30) ## SETS FONT SIZE TO 30
        tableformat = QtGui.QTextTableFormat() ## CREATES FORMAT FOR TABLE
        tableformat.setAlignment(QtCore.Qt.AlignHCenter) ## ALIGNS TABLE IN THE MIDDLE
        self.cursor.insertText(setdetails[0],titleformat) ## INSERTS TITLE

        query = '''SELECT * FROM (SELECT Students.Username,FirstName,Surname,Score FROM Scores
INNER JOIN Students ON Scores.Username=Students.Username WHERE SID=? AND Students.Username NOT IN
(SELECT Username FROM Teachers) ORDER BY Score ASC) AS allscores GROUP BY Username ORDER BY Score DESC;'''
        cur.execute(query,(setdetails[2],)) ## EXECUTES QUERY
        scores = cur.fetchall() ## FETCHES ALL SCORES
        studentnum = len(scores) ## GETS NUMBER OF STUDENTS
        scores.insert(0,('0','0','0','0')) ## NEEDED AS INDEXING STARTS FROM 0

        tempnum = studentnum - 78 ## TEMPORARY NUMBER OF STUDENTS FOR FINDING HOW MANY PAGES ARE NEEDED
        pages = 0 ## RESETS HOW MANY PAGES ARE NEEDED
        while tempnum > 0:
            tempnum -= 84 ## REMOVES ONE PAGE'S WORTH OF STUDENTS FROM TEMPNUM
            pages += 1 ## ADDS A PAGE
        totalrows = 41 + (pages*42) ## FINDS TOTAL NUMBER OF ROWS NEEDED
        page = 1 ## SETS PAGE NUMBER
        columnvalue = 1 ## STARTS ON LEFT COLUMN
        rowsperpage = 39 ## ONLY 39 ROWS OF STUDENTS ON FIRST PAGE
        while studentnum > 0:
            if page == 1: ## CHECKS IF ON FIRST PAGE
                lowerbound = columnvalue ## STARTS FROM ROW 1
                upperbound = lowerbound + rowsperpage ## ENDS AT ROW 40
            else:
                lowerbound = columnvalue + 78 + ((page-2)*84) ## FORMULA FOR FINDING STARTING ROW FOR PAGE
                upperbound = lowerbound + rowsperpage ##  42 ROWS OF STUDENTS ON EVERY REMAINING PAGE
            for i in range(lowerbound,upperbound): ## LOOPS THROUGH EACH ROW NUMBER ON PAGE
                if studentnum > 0:
                    self.cursor.insertText(str(i)) ## I IS POSITION OF USER IN LEADERBOARD
                    self.cursor.movePosition(self.cursor.NextCell)
                    name = scores[i][1]+" "+scores[i][2] ## FINDS FULL NAME OF USER
                    self.cursor.insertText(name)
                    self.cursor.movePosition(self.cursor.NextCell)
                    self.cursor.insertText(str(scores[i][3])) ## INSERTS THE ACTUAL SCORE FOR THE USER
                    self.cursor.movePosition(self.cursor.NextRow)
                    if columnvalue == rowsperpage+1: ## CHECKS IF AT END OF PAGE
                        self.cursor.movePosition(self.cursor.NextCell)
                        self.cursor.movePosition(self.cursor.NextCell)
                        self.cursor.movePosition(self.cursor.NextCell)
                        self.cursor.movePosition(self.cursor.NextCell)
                    studentnum -= 1 ## DECREASES NUMBER OF STUDENTS LEFT TO INSERT
                else:
                    break
            if columnvalue == 1: ## CHECKS IF ON FIRST COLUMN
                columnvalue += rowsperpage ## GOES TO SECOND COLUMN
                for i in range(rowsperpage): ## LOOPS THROUGH ALL ROWS TO GO TO TOP OF PAGE
                    self.cursor.movePosition(self.cursor.PreviousRow)
                self.cursor.movePosition(self.cursor.PreviousCell)
                self.cursor.movePosition(self.cursor.PreviousCell) ## MOVES TO SECOND COLUMN
            else:
                page += 1 ## GOES TO NEXT PAGE
                columnvalue = 1 ## RESETS TO FIRST COLUMN
                rowsperpage = 42 ## ENSURES NUMBER OF ROWS ON PAGE IS 42
                self.cursor.movePosition(self.cursor.PreviousCell) ## MOVES TO FIRST COLUMN
                self.cursor.movePosition(self.cursor.PreviousCell)
                self.cursor.movePosition(self.cursor.PreviousCell)
                self.cursor.movePosition(self.cursor.PreviousCell)
```

To show that the window still works when inside the main program, I have done a quick test which can be seen below.



I also did a quick test for a question set that had no results recorded to check that an empty table was produced, and as shown below, this is the case.



OBJECTIVES 49, AND 50 COMPLETED – TEACHERS CAN NOW VIEW LEADERBOARDS OF EVERYONE WHO HAS ATTEMPTED A QUIZ, AND CAN PRINT THE LEADERBOARD

MAINTENANCE

Now that the program has all the features that were specified in my objectives list, I am going to go back through the program and ensure that it is in a maintainable and easy to understand state. In order to do this, I have made a checklist of all the things that I will be looking at and fixing if necessary:

- Use regular expressions for text validation
- Using question marks in SQL queries instead of string manipulation
- Using Hungarian notation for variables
- Checking user interaction with the program
- Ensure capitalised names for constants

Additionally, now that all the windows to the program are complete, I can check off three of the objectives from my original list.

OBJECTIVES 6, AND 7 COMPLETED – PROGRAM SUCCESSFULLY IMPORTS ALL NECESSARY MODULES AND UTILISISES THEM, USES OOP TO CREATE CLASSES FOR EACH WINDOW

REGULAR EXPRESSIONS

For every bit of validation in my program, I have decided to go back and create a regular expression to reduce all the unnecessary code. Due to this, where before users were given a list of errors they had made when inputting data, the program will just let the user know all the criteria that need to be met for a valid entry.

PASSWORD VALIDATION

For password validation, the user must enter a password with six or more characters, made up of only alphanumerical characters, and must contain at least 1 uppercase and 1 lowercase letter. The new regex statement I will be using to validate this is shown below:

^(?=.*[a-z])(?=.*[A-Z])[a-zA-Z\d]{6,}$

Below I have taken screenshots of all the locations, the teacher registration and reset password screens, where I have added this new regex statement, along with a test showing that the above statement works as intended.

```
    def reset(self):
        new = self.newedit.text() ## GETS NEW PASSWORD
        confirm = self.confirmedit.text() ## GETS CONFIRMED PASSWORD
        valid = re.match("^(?=.*[a-z])(?=.*[A-Z])[a-zA-Z\d]{6,}$",new) ## USES REGULAR EXPRESSION TO CHECK IF VALID
        if valid and new == confirm: ## CHECKS IF PASSWORDS ARE VALID
            query = 'SELECT FirstName,Surname,Username FROM Students WHERE Username=?;' ## QUERY TO FETCH DETAILS OF STUDENT
            cur.execute(query,(cur_user,)) ## EXECUTES QUERY
            details = cur.fetchall()[0] ## FETCHES DETAILS
            password = hashlib.sha256(new.encode()).hexdigest() ## HASHES NEW PASSWORD
            query = 'REPLACE INTO Students (Username,Password,FirstName,Surname) VALUES (?,?,?,?);' ## QUERY TO REPLACE PASSWORD
            cur.execute(query,(details[2],password,details[0],details[1],)) ## EXECUTES QUERY
            con.commit() ## WRITES CHANGES TO DATABASE
            CreateOutbox("Password Reset","Your password has been successfully changed.","",QtGui.QMessageBox.Information)
            ResetPassword_Window.hide() ## HIDES THE RESET WINDOW
            Login_Window.hide() ## HIDES THE LOGIN WINDOW
            StudentHome_Window.show() ## SHOWS THE STUDENT HOME WINDOW
        else:
            CreateOutbox("Password Not Reset",'''There has been an error with changing your password.
These are the conditions that must be met:''','''Both passwords must be identical
Password must only have alphanumerical characters
Password must contain at least 1 uppercase and 1 lowercase letter
Password must be at least 6 characters long''',QtGui.QMessageBox.Critical)
```

```
        valid = re.match("^(?=.*[a-z])(?=.*[A-Z])[a-zA-Z\d]{6,}$",password) ## USES REGULAR EXPRESSION TO CHECK IF VALID
        if not valid:
            errors.append('''Password must only contain alphanumerical characters
Password must contain at least 1 uppercase and 1 lowercase letter
Password must be at least 6 characters long''')
```

USERNAME VALIDATION

For username validation, the only thing that needs to be checked is that it is between 3 and 15 characters long, and only contains alphanumerical characters. The new regex statement I am using for this is:

^[a-zA-Z\d]{3,15}$

Below is where I have implemented this new statement into both the teacher and student registration windows.

```
valid = re.match("^[a-zA-Z\d]{3,15}$",username)
if not valid:
    errors.append("Username must be alphanumerical only\nUsername must be between 3 and 15 characters long")
else:
    tables = ["Teachers","Students"] ## LIST CONTAINING TABLE NAMES TO BE SEARCHED
    for x in tables: ## STARTS A FOR LOOP FOR SEARCHING THE TABLES
        query = "SELECT Username FROM "+x+" WHERE Username='"+username+"';" ## CREATES THE QUERY FOR SEARCHING THE CURRENT TABLE
        cur.execute(query) ## EXECUTES THE QUERY
        temp = cur.fetchall() ## FETCHES ANY DATA FROM THE DATABASE THAT WAS SELECTED
        if temp!= []: ## CHECKS IF ANY DATA WAS FETCHED
            errors.append("Username already taken")
            break ## BREAKS OUT OF THE LOOP IF THE USERNAME HAS BEEN FOUND
```

SURNAME VALIDATION

For surname validation, the input must be alphanumerical only, except for a single dash that is allowed for double barrelled surnames, and there must be text either side of the dash. The regex I have chosen to use can be seen below:

(?=[a-zA-Z-]{1,}$)^[a-zA-Z]+(-[a-zA-Z]+)*$

Below is where I have implemented this validation in both the teacher and student registration windows.

```
        valid = re.match("(?=[a-zA-Z-]{1,}$)^[a-zA-Z]+(-[a-zA-Z]+)*$",surname) ## USES REGEX TO CHECK IF VALID
        if not valid:
            errors.append('''Surname must only contain alphanumerical characters
Surname can contain a maximum of one dash between two barrels
Surname must be at least one character long''')
        else:
            surname = surname.title() ## TITLISES THE SURNAME
```

FIRST NAME VALIDATION

Validating first names is similar to validating surnames, except there are no dashes that need to be checked for, and for this reason, the regex I am using can be seen below:

^[a-zA-Z]{1,}$

Below is the implementation of this validation in the student registration window.

```
        valid = re.match("^[a-zA-Z]{1,}$",firstname) ## CHECKS IF FIRST NAME IS VALID WITH REGEX
        if not valid:
            errors.append('''First name must contain only alphanumerical characters
First name must be filled in''')
        else:
            firstname = firstname.title() ## TITLISES FIRST NAME
```

OBJECTIVE 12 COMPLETED – ALL ENTRY BOXES HAVE WORKING VALIDATION

CONSISTENT SQL QUERIES

When writing my program, I started writing SQL queries by using string manipulation to add parameters to the queries. As I progressed in the program, I found it easier and more efficient to user question marks and then execute the queries with the necessary parameters. Having now finished the program, I have gone back and changed all the SQL queries to use the question marks, and these can be seen below.

```
password = hashlib.sha256(password.encode()).hexdigest() ## HASHES THE PASSWORD FOR STORING IN THE DATABASE
query = 'INSERT INTO Teachers (Username,Password,Title,Surname) VALUES (?,?,?,?);' ## INSERTS NEW ACCOUNT INTO DATABASE WITH CORRECT DATA
cur.execute(query,(username,password,title,surname,)) ## EXECUTES THE QUERY
con.commit() ## WRITES CHANGES TO THE DATABASE
CreateOutbox("Account Created","Your account has been created!","",QtGui.QMessageBox.Information) ## CREATES ACCOUNT CREATED MESSAGE BOX
```

```
word = "%" + self.searchedit.text() + "%" ## GETS STRING FROM SEARCH BAR
for char in ["'",";",'"']: ## LOOPS THROUGH THE DISALLOWED CHARACTERS
    word = word.replace(char,"") ## REMOVES CHARACTERS FROM USER'S ENTRY
query = 'SELECT SetName,Author,SID FROM Sets WHERE SetName LIKE ? OR Author LIKE ?;'
cur.execute(query,(word,word,)) ## EXECUTES QUERY
```

```
        query = '''SELECT Question,Correct,Wrong1,Wrong2,Wrong3 FROM Questions INNER JOIN Sets ON
QSLinks.SID=Sets.SID INNER JOIN QSLinks ON Questions.QID=QSLinks.QID WHERE
(Sets.SetName=? AND Sets.Author=?);''' ## FETCHES ALL QUESTIONS IN THE SET
        cur.execute(query,(setname,author,)) ## EXECUTES THE QUERY
        questions = cur.fetchall() ## FETCHES ALL THE QUESTIONS FROM THE DATABASE
```

An error that did occur was when trying to use this method when accessing different tables with a single loop, which occurs when logging in, and checking to see if a username is taken. In order to prevent this, I added an extra line to replace the first question mark with the table name before executing the query.

BEFORE:

```
tables = ["Teachers","Students"] ## LIST CONTAINING TABLE NAMES TO BE SEARCHED
for x in tables: ## STARTS A FOR LOOP FOR SEARCHING THE TABLES
    query = "SELECT Username,Password FROM ? WHERE Username=?;" ## CREATES THE QUERY FOR SEARCHING THE CURRENT TABLE
    cur.execute(query,(x,username,)) ## EXECUTES THE QUERY
    temp = cur.fetchall() ## FETCHES ANY DATA FROM THE DATABASE THAT WAS SELECTED
```

```
Traceback (most recent call last):
  File "E:\Computing\Quizo - Development File\Quizo - Developing.py", line 70, i
n Login
    cur.execute(query,(x,username,)) ## EXECUTES THE QUERY
sqlite3.OperationalError: near "?": syntax error
```

AFTER:

```
query = "SELECT Username,Password FROM ? WHERE Username=?;"
query = query.replace("?",x,1) ## REPLACES THE X VALUE
cur.execute(query,(username,)) ## EXECUTES THE QUERY
```

HUNGARIAN NOTATION

Hungarian notation is a method of prefixing variables so that it is easy to see the variable's data type when looking through the code. I have written a key of the prefixes I have decided on, and these changes can be seen in the full code on the final pages of development.

- str = String
- int = Integer
- flt = Float
- lst = List
- obj = Object
- bln = Boolean
- chr = Character

DOUBLE CLICKS AND ENTER PRESSES

In my program, there are multiple times where I have incorporated double clicking and being able to press enter in text boxes and tables to do the same function as a button. To keep things consistent, I have gone back and added the ability to do this in more entry boxes and tables, and there is an example of this below. The rest of the changes can be seen in the full code at the end of the development section.

```
self.loginbutton.clicked.connect(self.Login) ## WHEN LOGIN BUTTON PRESSED
self.registerbutton.clicked.connect(self.Register) ## WHEN REGISTER BUTTON PRESED
self.passedit.returnPressed.connect(self.Login) ## RUNS THE CLICK FUNCTION WHEN ENTER IS PRESSED IN PASSWORD BOX
self.useredit.returnPressed.connect(self.Login) ## RUNS THE CLICK FUNCTION WHEN ENTER IS PRESSED IN USERNAME BOX
```

CAPITALISING CONSTANTS

To make it easier to see what are variables and what are constants, I have decided that all constants in my program will be fully capitalised, whereas variables will remain majorly lowercase. When going back through my program, I found only two constants, which were the floor and ceiling constants when making the graphs in the program. Nevertheless, I capitalised them, and they can be seen below.

```
self.INT_FLOOR = 1 ## MINIMUM Y CO-ORDINATE OF POINT - CAPITALISED FOR CONSTANT
self.FLT_CEILING = 28.5 ## MAXIMUM Y CO-ORDINATE OF POINT - CAPITALISED FOR CONSTANT
```

FINAL CODE FOR THE PROTOTYPE

Now that I have gone over my first prototype for the RGS again, I am happy with its amount of content and functionality. Although I have shown all the code previously as I've developed the program, I have taken screenshots of the entire code again so it is easy to view and follow. This code includes all the changes made that I have documented up to this point, and this marks the end of development for the first prototype of Quizo.

```
## IMPORTED ALL MODULES NEEDED FOR THIS WINDOW
import sys,os
import sqlite3 as lite
import hashlib,random,time,re
from PyQt4 import QtCore,QtGui,uic
from functools import partial
from tkinter import *

obj_con = lite.connect('QuizoBase') ## CONNECTING TO THE DATABASE
obj_cur = obj_con.cursor()

Login_class = uic.loadUiType("LoginWindow.ui")[0] ## LOADS UI FILE
TeacherRegistration_class = uic.loadUiType("TeacherRegistration.ui")[0] ## LOADS UI FILE
StudentHome_class = uic.loadUiType("StudentHome.ui")[0] ## LOADS UI FILE
TeacherHome_class = uic.loadUiType("TeacherHome.ui")[0] ## LOADS UI FILE
Play_class = uic.loadUiType("PlayGame.ui")[0] ## LOADS UI FILE
Easy_class = uic.loadUiType("EasyMode.ui")[0] ## LOADS UI FILE
Hard_class = uic.loadUiType("HardMode.ui")[0] ## LOADS UI FILE
Results_class = uic.loadUiType("Results.ui")[0] ## LOADS UI FILE
Leaderboard_class = uic.loadUiType("Leaderboard.ui")[0] ## LOADS UI FILE
SetAnalysis_class = uic.loadUiType("SetAnalysis.ui")[0] ## LOADS UI FILE
ViewStudents_class = uic.loadUiType("ViewStudents.ui")[0] ## LOADS UI FILE
ResetPassword_class = uic.loadUiType("ResetPassword.ui")[0] ## LOADS UI FILE
StudentRegistration_class = uic.loadUiType("StudentRegistration.ui")[0] ## LOADS UI FILE
ViewResults_class = uic.loadUiType("ViewResults.ui")[0] ## LOADS UI FILE
ViewSets_class = uic.loadUiType("ViewSets.ui")[0] ## LOADS UI FILE
Importing_class = uic.loadUiType("Importing.ui")[0] ## LOADS UI FILE
AddQuestions_class = uic.loadUiType("AddQuestions.ui")[0] ## LOADS UI FILE
PrintLeaders_class = uic.loadUiType("PrintLeaders.ui")[0] ## LOADS UI FILE


def CreateOutbox(str_title,str_text,str_infoText,str_icon):
    obj_outbox = QtGui.QMessageBox() ## CREATES THE MESSAGE BOX
    obj_outbox.setWindowTitle(str_title) ## SETS THE TITLE OF THE MESSAGE BOX
    obj_outbox.setText(str_text) ## SETS THE MAIN TEXT OF THE MESSAGE BOX
    obj_outbox.setInformativeText(str_infoText) ## SETS ADDITIONAL TEXT OF THE MESSAGE BOX
    obj_outbox.setIcon(str_icon) ## SETS THE ICON OF THE MESSAGE BOX
    obj_outbox.exec() ## EXECUTES THE MESSAGE BOX

class LoginWindow(QtGui.QMainWindow,Login_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.setupUi(self)
        self.loginbutton.clicked.connect(self.Login) ## WHEN LOGIN BUTTON PRESSED
        self.registerbutton.clicked.connect(self.Register) ## WHEN REGISTER BUTTON PRESED
        self.passedit.returnPressed.connect(self.Login) ## RUNS THE CLICK FUNCTION WHEN ENTER IS PRESSED IN PASSWORD BOX
        self.useredit.returnPressed.connect(self.Login) ## RUNS THE CLICK FUNCTION WHEN ENTER IS PRESSED IN USERNAME BOX

    def Register(self):
        TeacherRegistration_Window.show() ## SHOWS TEACHER REGISTRATION WINDOW
        Login_Window.hide() ## HIDES LOGIN WINDOW
        self.useredit.clear() ## CLEARS THE USERNAME BOX
        self.passedit.clear() ## CLEARS THE PASSWORD BOX

    def Login(self):
        global str_access,str_curuser ## GLOBALISES ACCESS LEVEL AND USERNAME SO ALL WINDOWS CAN USE IT
        str_username = self.useredit.text() ## GETS TEXT FROM USERNAME LINE EDIT
        str_password = self.passedit.text() ## GETS TEXT FROM PASSWORD LINE EDIT

        str_access = "" ## RESETS CURRENT USER'S ACCESS
        if str_username.isalnum() == True: ## CHECKS THAT THE USERNAME CONTAINS ONLY ALPHANUMERICAL VALUES
            lst_tables = ["Teachers","Students"] ## LIST CONTAINING TABLE NAMES TO BE SEARCHED
            for str_x in lst_tables: ## STARTS A FOR LOOP FOR SEARCHING THE TABLES
                str_query = "SELECT Username,Password FROM ? WHERE Username=?;" ## CREATES THE QUERY FOR SEARCHING THE CURRENT TABLE
                str_query = str_query.replace("?",str_x,1) ## REPLACES THE X VALUE
                obj_cur.execute(str_query,(str_username,)) ## EXECUTES THE QUERY
                lst_temp = obj_cur.fetchall() ## FETCHES ANY DATA FROM THE DATABASE THAT WAS SELECTED
                if lst_temp!= []: ## CHECKS IF ANY DATA WAS FETCHED
                    str_access = str_x ## SETS USER'S ACCESS LEVEL
                    lst_tempuser = lst_temp[0] ## SETS THE ACCOUNT TO THE NEW TEMPORARY USER
                    break ## BREAKS OUT OF THE LOOP IF THE USERNAME HAS BEEN FOUND

        if str_access == "": ## CHECKS TO SEE IF THE ACCOUNT WAS NOT FOUND
            CreateOutbox("Account Not Found","Account not found?","The account you have entered could not be found in our database, please try again.",
                         QtGui.QMessageBox.Question)
        else:
            str_hashedattempt = hashlib.sha256(str_password.encode()).hexdigest() ## HASHES USER'S ATTEMPT
            if str_hashedattempt == lst_tempuser[1]: ## CHECKS TO SEE IF THE HASHED ATTEMPT IS THE SAME AS THE HASHED PASSWORD
                str_curuser = str_username ## SETS CURRENT USER TO THE GIVEN USERNAME
                if str_hashedattempt == hashlib.sha256("Quizo123".encode()).hexdigest(): ## CHECKS IF USER IS LOGGING IN WITH DEFAULT PASSWORD
                    ResetPassword_Window.show() ## SHOWS THE RESET PASSWORD WINDOW
                else:
                    CreateOutbox("Logged In","Login was successful!",("You have successfully logged in as "+str_username+"."),QtGui.QMessageBox.Information)
                    if str_access == "Students":
                        StudentHome_Window.show() ## SHOWS STUDENT HOME SCREEN
                    else:
                        TeacherHome_Window.show() ## SHOWS TEACHER HOME SCREEN
                    Login_Window.hide() ## HIDES LOGIN SCREEN
            else:
                CreateOutbox("Unsuccessful Login","Login was unsuccessful!",
                             "The password you have entered is incorrect, please try again, or talk to your teacher if you can't remember your password.",
                             QtGui.QMessageBox.Critical)
```

```
class ResetPassword(QtGui.QMainWindow,ResetPassword_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.setupUi(self)
        self.resetbutton.clicked.connect(self.reset) ## RUNS RESET FUNCTION WHEN BUTTON PRESSED
        self.newedit.returnPressed.connect(self.reset) ## RUNS RESET FUNCTION WHEN RETURN PRESSED IN EITHER BOX
        self.confirmedit.returnPressed.connect(self.reset)

    def reset(self):
        str_new = self.newedit.text() ## GETS NEW PASSWORD
        str_confirm = self.confirmedit.text() ## GETS CONFIRMED PASSWORD
        obj_valid = re.match("^(?=.*[a-z])(?=.*[A-Z])[a-zA-Z\d]{6,}$",str_new) ## USES REGULAR EXPRESSION TO CHECK IF VALID
        if obj_valid and str_new == str_confirm: ## CHECKS IF PASSWORDS ARE VALID
            str_query = 'SELECT FirstName,Surname,Username FROM Students WHERE Username=?;' ## QUERY TO FETCH DETAILS OF STUDENT
            obj_cur.execute(str_query,(str_curuser,)) ## EXECUTES QUERY
            lst_details = obj_cur.fetchall()[0] ## FETCHES DETAILS
            str_password = hashlib.sha256(str_new.encode()).hexdigest() ## HASHES NEW PASSWORD
            str_query = 'REPLACE INTO Students (Username,Password,FirstName,Surname) VALUES (?,?,?,?);' ## QUERY TO REPLACE PASSWORD
            obj_cur.execute(str_query,(lst_details[2],str_password,lst_details[0],lst_details[1],)) ## EXECUTES QUERY
            obj_con.commit() ## WRITES CHANGES TO DATABASE
            CreateOutbox("Password Reset","Your password has been successfully changed.","",QtGui.QMessageBox.Information)
            ResetPassword_Window.hide() ## HIDES THE RESET WINDOW
            Login_Window.hide() ## HIDES THE LOGIN WINDOW
            StudentHome_Window.show() ## SHOWS THE STUDENT HOME WINDOW
        else:
            CreateOutbox("Password Not Reset",'''There has been an error with changing your password.
These are the conditions that must be met:''','''Both passwords must be identical
Password must only have alphanumerical characters
Password must contain at least 1 uppercase and 1 lowercase letter
Password must be at least 6 characters long''',QtGui.QMessageBox.Critical)


class TeacherRegistration(QtGui.QMainWindow,TeacherRegistration_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.setupUi(self)
        self.backbutton.clicked.connect(self.Back) ## STARTS BACK FUNCTION WHEN BACK BUTTON CLICKED
        self.createbutton.clicked.connect(self.addteacher) ## STARTS ADDTEACHER FUNCTION WHEN CREATE BUTTON CLICKED
        self.codeedit.returnPressed.connect(self.addteacher) ## STARTS ADDTEACHER FUNCTION WHEN ENTER IS PRESSED IN ANY BOX
        self.useredit.returnPressed.connect(self.addteacher)
        self.passedit.returnPressed.connect(self.addteacher)
        self.suredit.returnPressed.connect(self.addteacher)

    def addteacher(self):
        str_teachercode = self.codeedit.text() ## FETCHES CONTENTS OF TEACHER CODE BOX
        str_username = self.useredit.text() ## FETCHES CONTENTS OF USERNAME BOX
        str_password = self.passedit.text() ## FETCHES CONTENTS OF PASSWORD BOX
        str_title = self.comboBox.currentText() ## FETCHES SELECTED ITEM IN THE TITLE DROP DOWN
        str_surname = self.suredit.text() ## FETCHES CONTENTS OF SURNAME BOX
        lst_errors = [] ## RESETS THE ERRORS WHEN ADDING A NEW ACCOUNT

        obj_valid = re.match("^[a-zA-Z\d]{3,15}$",str_username) ## CHECKS USERNAME IS VALID
        if not obj_valid:
            lst_errors.append("Username must be alphanumerical only\nUsername must be between 3 and 15 characters long")
        else:
            lst_tables = ["Teachers","Students"] ## LIST CONTAINING TABLE NAMES TO BE SEARCHED
            for str_x in lst_tables: ## STARTS A FOR LOOP FOR SEARCHING THE TABLES
                str_query = "SELECT Username FROM ? WHERE Username=?;" ## CREATES THE QUERY FOR SEARCHING THE CURRENT TABLE
                str_query = str_query.replace("?",str_x,1) ## REPLACES THE X VALUE
                obj_cur.execute(str_query,(str_username,)) ## EXECUTES THE QUERY
                lst_temp = obj_cur.fetchall() ## FETCHES ANY DATA FROM THE DATABASE THAT WAS SELECTED
                if lst_temp!= []: ## CHECKS IF ANY DATA WAS FETCHED
                    lst_errors.append("Username already taken")
                    break ## BREAKS OUT OF THE LOOP IF THE USERNAME HAS BEEN FOUND


        obj_valid = re.match("^(?=.*[a-z])(?=.*[A-Z])[a-zA-Z\d]{6,}$",str_password) ## USES REGULAR EXPRESSION TO CHECK IF VALID
        if not obj_valid:
            lst_errors.append('''Password must only contain alphanumerical characters
Password must contain at least 1 uppercase and 1 lowercase letter
Password must be at least 6 characters long''')

        obj_valid = re.match("(?=[a-zA-Z-]{1,}$)^[a-zA-Z]+(-[a-zA-Z]+)*$",str_surname) ## USES REGEX TO CHECK IF VAID
        if not obj_valid:
            lst_errors.append('''Surname must only contain alphanumerical characters
Surname can contain a maximum of one dash between two barrels
Surname must be filled in''')
        else:
            str_surname = str_surname.title() ## TITLISES THE SURNAME

        if str_title == "Select": ## CHECKS TO SEE IF THE SELECT OPTION HAS BEEN SELECTED
            lst_errors.append("A title must be selected from the drop down menu")

        if str_teachercode != "YC71N": ## CHECKS IF THE ENTERED CODE MATCHES
            lst_errors.append("Teacher code is incorrect")

        if lst_errors != []: ## CHECKS IF USER HAS HAD ANY ERRORS
            str_errormessage = "\n".join(lst_errors) ## CONVERTS LIST OF ERRORS INTO SINGLE STRING
            CreateOutbox("Errors Found","There has been an error with creating your account:",str_errormessage,QtGui.QMessageBox.Critical) ## CREATES ERROR MESSAGE BOX
        else:
            str_password = hashlib.sha256(str_password.encode()).hexdigest() ## HASHES THE PASSWORD FOR STORING IN THE DATABASE
            str_query = 'INSERT INTO Teachers (Username,Password,Title,Surname) VALUES (?,?,?,?);' ## INSERTS NEW ACCOUNT INTO DATABASE WITH CORRECT DATA
            obj_cur.execute(str_query,(str_username,str_password,str_title,str_surname,)) ## EXECUTES THE QUERY
            obj_con.commit() ## WRITES CHANGES TO THE DATABASE
            CreateOutbox("Account Created","Your account has been created!","",QtGui.QMessageBox.Information) ## CREATES ACCOUNT CREATED MESSAGE BOX


    def Back(self):
        Login_Window.show() ## SHOWS LOGIN WINDOW
        TeacherRegistration_Window.hide() ## HIDES TEACHER REGISTRATION WINDOW
        self.codeedit.clear() ## CLEARS THE TEACHER CODE BOX
        self.useredit.clear() ## CLEARS THE USERNAME BOX
        self.suredit.clear() ## CLEARS THE SURNAME BOX
        self.passedit.clear() ## CLEARS THE PASSWORD BOX
        self.comboBox.setCurrentIndex(0) ## RESETS THE TITLE DROP DOWN TO SELECT
```

```
class StudentHome(QtGui.QMainWindow,StudentHome_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.setupUi(self)
        self.playbutton.clicked.connect(self.play) ## STARTS PLAY FUNCTION WHEN PLAY BUTTON CLICKED
        self.resultsbutton.clicked.connect(self.results) ## STARTS RESULTS FUNCTION WHEN RESULTS BUTTON IS CLICKED
        self.logoutbutton.clicked.connect(self.logout) ## STARTS LOGOUT FUNCTION WHEN LOGOUT BUTTON CLICKED

    def logout(self):
        Login_Window.show() ## SHOWS THE LOGIN WINDOW
        StudentHome_Window.hide() ## HIDES THE STUDENT HOME WINDOW

    def play(self):
        PlayGame_Window.show() ## SHOWS THE PLAY GAME WINDOW
        StudentHome_Window.hide() ## HIDES THE STUDENT HOME WINDOW

    def results(self):
        ViewResults.displayresults(ViewResults_Window,str_curuser) ## BEGINS THE FUNCTION TO DISPLAY HIGH SCORES IN THE VIEW RESULTS WINDOW
        ViewResults_Window.show() ## SHOWS THE VIEW RESULTS WINDOW
        StudentHome_Window.hide() ## HIDES THE STUDENT HOME WINDOW

class TeacherHome(QtGui.QMainWindow,TeacherHome_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.setupUi(self)
        self.playbutton.clicked.connect(self.play) ## STARTS PLAY FUNCTION WHEN PLAY BUTTON CLICKED
        self.studbutton.clicked.connect(self.stud) ## STARTS STUD FUNCTION WHEN VIEW STUDENTS BUTTON CLICKED
        self.setbutton.clicked.connect(self.sets) ## STARTS SET FUNCTION WHEN VIEW QUESTION SETS BUTTON IS CLICKED
        self.logoutbutton.clicked.connect(self.logout) ## STARTS LOGOUT FUNCTION WHEN LOGOUT BUTTON CLICKED

    def logout(self):
        Login_Window.show() ## SHOWS THE LOGIN WINDOW
        TeacherHome_Window.hide() ## HIDES THE TEACHER HOME WINDOW

    def play(self):
        PlayGame_Window.show() ## SHOWS THE PLAY GAME WINDOW
        TeacherHome_Window.hide() ## HIDES THE TEACHER HOME WINDOW

    def sets(self):
        ViewSets.search(ViewSets_Window) ## DISPLAYS THE TABLE BEFORE THE WINDOW OPENS
        ViewSets_Window.show() ## SHOWS THE VIEW SETS WINDOW
        TeacherHome_Window.hide() ## HIDES THE TEACHER HOME WINDOW

    def stud(self):
        ViewStudents.search(ViewStudents_Window) ## DISPLAYS THE TABLE
        ViewStudents_Window.show() ## SHOWS THE VIEW STUDENTS WINDOW
        TeacherHome_Window.hide() ## HIDES THE TEACHER HOME WINDOW

class PlayGame(QtGui.QMainWindow,Play_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        QtGui.QMainWindow.showEvent(self,parent) ## RUNS SHOWEVENT FUNCTION WHENEVER WINDOW IS SHOWN
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.setupUi(self)
        self.backbutton.clicked.connect(self.Back) ## CHECKS IF BACK BUTTON IS PRESSED
        self.playbutton.clicked.connect(self.findselection) ## CHECKS IF PLAY BUTTON IS PRESSED
        self.randombutton.clicked.connect(self.randomquiz) ## CHECKS IF RANDOM BUTTON IS PRESSED
        self.searchedit.textChanged.connect(self.search) ## RUNS SEARCH FUNCTION WHENEVER SEARCH BAR CONTENTS CHANGE
        self.setstable.doubleClicked.connect(self.findselection) ## STARTS QUIZ WHEN A ROW IS DOUBLE CLICKED

    def showEvent(self,parent=None):
        self.search()

    def search(self):
        str_word = "%" + self.searchedit.text() + "%" ## GETS STRING FROM SEARCH BAR
        for chr_char in ["'",";",'"']: ## LOOPS THROUGH THE DISALLOWED CHARACTERS
            str_word = str_word.replace(chr_char,"") ## REMOVES CHARACTERS FROM USER'S ENTRY
        str_query = 'SELECT SetName,Author,SID FROM Sets WHERE SetName LIKE ? OR Author LIKE ?;' ## QUERY TO FETCH SET NAMES WITH SIMILAR TITLES OR AUTHORS TO USER'S SEARCH
        obj_cur.execute(str_query,(str_word,str_word,)) ## EXECUTES QUERY
        self.lst_allsets = obj_cur.fetchall() ## FETCHES DATA FROM DATABASE
        obj_model = QtGui.QStandardItemModel(len(self.lst_allsets),2) ## CREATES A MODEL TO PUT THE DATA IN
        for int_row in range(0,len(self.lst_allsets)): ## LOOPS THROUGH EACH SET
            for int_column in range(0,2): ## LOOPS THROUGH SET NAME AND AUTHOR FOR THE CURRENT SET
                obj_model.setData(obj_model.index(int_row,int_column),str(self.lst_allsets[int_row][int_column])) ## SETS THE MODEL DATA ACCORDING TO THE FETCHED DATA
        self.setstable.setModel(obj_model) ## SETS THE MODEL TO THE TABLE VIEW
        self.setstable.setColumnWidth(0,500) ## SETS COLUMN WIDTH FOR SETS COLUMN
        self.setstable.setColumnWidth(1,265) ## SETS COLUMN WIDTH FOR AUTHOR COLUMN
        self.setstable.show() ## SHOWS THE TABLE

    def Back(self):
        if str_access == "Teachers": ## CHECKS IF USER IS A TEACHER
            TeacherHome_Window.show() ## SHOWS THE TEACHER HOME WINDOW
        else:
            StudentHome_Window.show() ## SHOWS THE STUDENT HOME WINDOW
        self.searchedit.clear() ## CLEARS THE SEARCH BAR
        PlayGame_Window.hide() ## HIDES THE PLAY GAME WINDOW

    def findselection(self):
        lst_selection = self.setstable.selectionModel().selectedRows() ## FETCHES THE SELECTED ROWS FROM THE TABLE
        if lst_selection == []: ## CHECKS IF ANY ROW HAS BEEN SELECTED
            CreateOutbox("Selection Not Found","You must select a question set to play!","",QtGui.QMessageBox.Critical)
        else:
            int_selectedrow = lst_selection[0].row() ## FINDS INDEX OF SELECTED ROW
            lst_playset = self.lst_allsets[int_selectedrow] ## SETS THE CURRENT PLAYSET TO THE SELECTED SET
            self.startgame(lst_playset) ## PASSES THE SELECTED QUESTION SET TO THE START GAME FUNCTION

    def startgame(self,lst_playset):
        if self.multichoice.isChecked() == True: ## CHECKS IF THE MULTIPLE CHOICE OPTION IS CHECKED
            Easy_Window.show() ## SHOWS THE MULTI-CHOICE WINDOW
            Easy.startnew(Easy_Window,lst_playset) ## PASSES THE WINDOW AND CHOSEN PLAYSET TO THE MULTI-CHOICE WINDOW
        else:
            Hard_Window.show() ## SHOWS THE NON-MULTI-CHOICE WINDOW
            Hard.startnew(Hard_Window,lst_playset) ## PASSES THE WINDOW AND CHOSEN PLAYSET TO THE MULTI-CHOICE WINDOW
        PlayGame_Window.hide() ## HIDES THE PLAY GAME WINDOW

    def randomquiz(self):
        str_query = "SELECT SetName,Author,SID FROM Sets;" ## QUERY TO FETCH ALL SETS FROM DATABASE
        obj_cur.execute(str_query) ## EXECUTES QUERY
        lst_data = obj_cur.fetchall() ## FETCHES DATA FROM DATABASE
        randsetnum = random.randint(1,len(lst_data)) ## CHOOSES RANDOM NUMBER BETWEEN 1 AND THE NUMBER OF SETS IN THE DATABASE
        lst_playset = lst_data[randsetnum-1] ## GETS THE SETNAME AND AUTHOR OF THE CHOSEN SET (-1 AS INDEXES START FROM 0)
        self.startgame(lst_playset) ## PASSES THE CHOSEN QUESTION SET TO THE START GAME FUNCTION
```

```
class Easy(QtGui.QMainWindow,Easy_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setupUi(self)
        self.ans1button.clicked.connect(partial(self.click,0)) ## RUNS CLICK FUNCTION WHEN BUTTON 1 PRESSED
        self.ans2button.clicked.connect(partial(self.click,1)) ## RUNS CLICK FUNCTION WHEN BUTTON 2 PRESSED
        self.ans3button.clicked.connect(partial(self.click,2)) ## RUNS CLICK FUNCTION WHEN BUTTON 3 PRESSED
        self.ans4button.clicked.connect(partial(self.click,3)) ## RUNS CLICK FUNCTION WHEN BUTTON 4 PRESSED

    def startnew(self,lst_playset):
        global lst_questions
        self.lst_playset = lst_playset ## SETS THE CURRENT QUESTION SET FOR THE WINDOW
        str_setname = lst_playset[0] ## GETS THE NAME OF THE QUESTION SET
        str_author = lst_playset[1] ## GETS THE AUTHOR OF THE QUESTION SET

        str_query = '''SELECT Question,Correct,Wrong1,Wrong2,Wrong3 FROM Questions INNER JOIN Sets ON
QSLinks.SID=Sets.SID INNER JOIN QSLinks ON Questions.QID=QSLinks.QID WHERE
(Sets.SetName=? AND Sets.Author=?);''' ## FETCHES ALL QUESTIONS IN THE SET
        obj_cur.execute(str_query,(str_setname,str_author,)) ## EXECUTES THE QUERY
        lst_questions = obj_cur.fetchall() ## FETCHES ALL THE QUESTIONS FROM THE DATABASE
        random.shuffle(lst_questions) ## SHUFFLES THE ORDER OF THE QUESTIONS IN THE LIST

        self.lst_incorrect = [] ## RESETS THE LIST OF USER'S INCORRECT ANSWERS
        self.flt_total = 0 ## SETS USER'S TOTAL SCORE TO 0
        self.obj_timer = QtCore.QElapsedTimer() ## CREATES A TIMER OBJECT
        self.obj_timer.start() ## STARTS THE TIMER
        self.ask() ## DISPLAYS THE QUESTION TO THE USER

    def ask(self):
        if len(lst_questions) != 0: ## CHECKS THAT THERE ARE STILL QUESTIONS LEFT TO ASK
            current = lst_questions[0] ## SETS THE CURRENT QUESTION
            self.questionlabel.setText(current[0]) ## DISPLAYS THE QUESTION TO THE USER
            self.choices = [1,2,3,4] ## THE POSSIBLE BUTTONS THAT CAN BE PRESSED
            random.shuffle(self.choices) ## SHUFFLES THE ORDER OF THE BUTTONS
            self.ans1button.setText(current[self.choices[0]]) ## SETS THE TEXT OF BUTTON 1
            self.ans2button.setText(current[self.choices[1]]) ## SETS THE TEXT OF BUTTON 2
            self.ans3button.setText(current[self.choices[2]]) ## SETS THE TEXT OF BUTTON 3
            self.ans4button.setText(current[self.choices[3]]) ## SETS THE TEXT OF BUTTON 4
        else:
            self.obj_timer.invalidate() ## STOPS THE TIMER
            Results.newresults(Results_Window,self.lst_playset,self.lst_incorrect,self.flt_total) ## PASSES THE CURRENT SET TO THE NEXT WINDOW
            Results_Window.show() ## SHOWS THE RESULTS WINDOW
            Easy_Window.hide() ## HIDES THE CURRENT WINDOW

    def click(self,number):
        int_correct = int(self.choices.index(1)) ## FINDS THE BUTTON NUMBER WITH THE CORRECT ANSWER
        int_points = round(1000000/self.obj_timer.restart()) ## SETS POINTS TO THE ELAPSED TIME
        if number == int_correct: ## CHECKS IF THE BUTTON PRESSED IS THE SAME AS THE BUTTON WITH THE CORRECT ANSWER
            self.flt_total += int_points ## ADDS POINTS TO TOTAL
        else:
            self.flt_total -= int_points ## DEDUCTS POINTS FROM TOTAL
            self.lst_incorrect.append([lst_questions[0][0],lst_questions[0][self.choices[number]],lst_questions[0][1]]) ## ADDS QUESTION TO INCORRECT LIST
        lst_questions.pop(0) ## REMOVES THE FIRST QUESTION FROM THE QUESTION LIST
        self.ask() ## AFTER USER CHOOSES AN ANSWER, NEXT QUESTION IS ASKED

class Hard(QtGui.QMainWindow,Hard_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setupUi(self)
        self.enterbutton.clicked.connect(self.click)
        self.ansedit.returnPressed.connect(self.click) ## RUNS THE CLICK FUNCTION WHEN ENTER IS PRESSED

    def startnew(self,lst_playset):
        global lst_questions
        self.lst_playset = lst_playset ## SETS THE CURRENT QUESTION SET FOR THE WINDOW
        str_setname = lst_playset[0] ## GETS THE TITLE FOR THE QUESTION SET
        str_author = lst_playset[1] ## GETS THE AUTHOR OF THE QUESTION SET

        str_query = '''SELECT Question,Correct,Wrong1,Wrong2,Wrong3 FROM Questions INNER JOIN Sets
ON QSLinks.SID=Sets.SID INNER JOIN QSLinks ON Questions.QID=QSLinks.QID WHERE
(Sets.SetName=? AND Sets.Author=?);''' ## FETCHES ALL QUESTIONS IN THE SET
        obj_cur.execute(str_query,(str_setname,str_author,)) ## EXECUTES THE QUERY
        lst_questions = obj_cur.fetchall() ## FETCHES ALL THE QUESTIONS FROM THE DATABASE
        random.shuffle(lst_questions) ## SHUFFLES THE ORDER OF THE QUESTIONS IN THE LIST

        self.lst_incorrect = [] ## RESETS THE LIST OF USER'S INCORRECT ANSWERS
        self.flt_total = 0 ## SETS USER'S TOTAL SCORE TO 0
        self.obj_timer = QtCore.QElapsedTimer() ## CREATES A TIMER OBJECT
        self.obj_timer.start() ## STARTS THE TIMER
        self.ask() ## DISPLAYS THE QUESTION TO THE USER

    def ask(self):
        if len(lst_questions)!=0: ## CHECKS IF THERE ARE ANY MORE QUESTIONS LEFT
            self.questionlabel.setText(lst_questions[0][0]) ## DISPLAYS THE CURRENT QUESTION IN THE LABEL
        else:
            self.obj_timer.invalidate() ## STOPS THE TIMER
            Results.newresults(Results_Window,self.lst_playset,self.lst_incorrect,self.flt_total) ## PASSES THE CURRENT SET TO THE NEXT WINDOW
            Results_Window.show() ## SHOWS THE RESULTS WINDOW
            Hard_Window.hide() ## HIDES THE CURRENT WINDOW


    def click(self):
        str_answer = self.ansedit.text() ## GETS USER'S ANSWER FROM TEXT BOX
        int_correct = lst_questions[0][1] ## CREATES VARIABLE TO STORE THE CORRECT ANSWER
        int_points = round(1000000/self.obj_timer.restart()) ## SETS POINTS TO THE ELAPSED TIME
        if str_answer.lower() == int_correct.lower(): ## CHECKS IF THE USER'S ATTEMPT IS THE SAME AS THE ANSWER - NOT CASE SENSITIVE
            self.flt_total += int_points ## ADDS POINTS TO TOTAL
        else:
            self.flt_total -= int_points ## DEDUCTS POINTS FROM TOTAL
            self.lst_incorrect.append([lst_questions[0][0],str_answer,lst_questions[0][1]]) ## ADDS QUESTION TO INCORRECT LIST
        self.ansedit.clear() ## CLEARS THE TEXT BOX
        lst_questions.pop(0) ## POPS THE FIRST QUESTION FROM THE QUESTION LIST
        self.ask() ## AFTER USER CHOOSES AN ANSWER, NEXT QUESTION IS ASKED
```

```
class Results(QtGui.QMainWindow,Results_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.setupUi(self)
        self.incorrecttable.doubleClicked.connect(self.showincorrect) ## RECOGNISES A DOUBLE CLICK IN THE TABLE
        self.leaderboardbutton.clicked.connect(self.showleaders)
        self.homebutton.clicked.connect(self.home)
        self.newbutton.clicked.connect(self.playagain)
        self.reviewbutton.clicked.connect(self.review)

    def newresults(self,lst_playset,lst_incorrect,flt_total):
        self.lst_incorrect = lst_incorrect ## SETS THE INCORRECT QUESTIONS FOR THE CURRENT SET
        self.lst_playset = lst_playset ## SETS THE CURRENT QUESTION SET

        str_query = 'INSERT INTO Scores (Username,SID,Score,Date) VALUES (?,?,?,datetime("now"));' ## QUERY FOR INSERTING SCORE INTO DATABASE
        obj_cur.execute(str_query,(str_curuser,str(self.lst_playset[2]),str(flt_total),)) ## FORMATS THE QUERY WITH CORRECT VARIABLES AND EXECUTES
        obj_con.commit() ## WRITES CHANGES TO THE DATABASE

        self.scorelabel.setText("YOUR SCORE: "+str(flt_total)) ## DISPLAYS USER'S TOTAL
        obj_model = QtGui.QStandardItemModel(len(lst_incorrect),3) ## CREATES MODEL FOR THE TABLE
        for int_x in range(0,len(lst_incorrect)): ## LOOPS THROUGH EACH INCORRECT QUESTION
            for int_i in range(0,3): ## LOOPS THROUGH THE QUESTION, USER'S ANSWER AND CORRECT ANSWER FOR THE QUESTION
                obj_model.setData(obj_model.index(int_x,int_i),str(lst_incorrect[int_x][int_i])) ## ADDS THE DATA TO THE TABLE MODEL
        self.incorrecttable.setModel(obj_model) ## SETS THE TABLE MODEL
        lst_tempwidth = [375,200,185] ## LIST OF COLUMN WIDTHS
        for int_i in range(3): ## ITERATES THROUGH EACH COLUMN
            self.incorrecttable.setColumnWidth(int_i,lst_tempwidth[int_i]) ## SETS THE WIDTH OF THE COLUMN
        self.incorrecttable.show() ## DISPLAYS THE TABLE

    def showincorrect(self,obj_index):
        lst_selection = self.incorrecttable.selectionModel().selectedRows() ## GETS THE SELECTED ROWS FROM THE TABLE
        int_selectedrow = lst_selection[0].row() ## FINDS INDEX OF SELECTED ROW
        lst_details = self.lst_incorrect[int_selectedrow] ## GETS THE DETAILS OF THE SELECTED QUESTION
        str_detailmsg = ("You answered: "+lst_details[1]+"\n"+"Correct Answer: "+lst_details[2]) ##  SHOWS THE CORRECT ANSWER AND THE USERS ANSWER
        CreateOutbox("Incorrect Answer",lst_details[0],str_detailmsg,QtGui.QMessageBox.Information) ## CREATES THE MESSAGE BOX

    def showleaders(self):
        Leaderboard_Window.show() ## SHOWS THE LEADERBOARD
        Leaderboard.fetchleaders(Leaderboard_Window,self.lst_playset)

    def home(self):
        if str_access == "Students":
            StudentHome_Window.show() ## RETURNS USER TO THE STUDENT HOME
        else:
            TeacherHome_Window.show() ## RETURNS USER TO THE TEACHER HOME
        Results_Window.hide() ## HIDES THE RESULTS WINDOW

    def playagain(self):
        PlayGame_Window.show() ## RETURNS USER TO THE PLAY GAME SCREEN
        Results_Window.hide() ## HIDES THE RESULTS WINDOW
    def review(self):
        str_query = "SELECT COUNT(Score) FROM Scores WHERE SID=? AND Username=?;" ## QUERY THAT FETCHES THE NUMBER OF SCORES BY THE USER IN A SET
        obj_cur.execute(str_query,(self.lst_playset[2],str_curuser,)) ## EXECUTES THE QUERY WITH THE CORRECT PARAMETERS
        int_scorenum = obj_cur.fetchone()[0] ## FETCHES THE ACTUAL NUMBER OF SCORES
        if int_scorenum > 1000:
            obj_outbox = QtGui.QMessageBox() ## CREATES THE MESSAGE BOX
            obj_outbox.setWindowTitle("Set Analysis") ## SETS THE TITLE OF THE MESSAGE BOX
            obj_outbox.setText("You are trying to analyse a set where you have more than 1000 scores.") ## SETS THE MAIN TEXT OF THE MESSAGE BOX
            obj_outbox.setInformativeText('''This may take some time to process, would you like to continue?
\n(If you have more than 20,000 results, it is possible the program will crash)''') ## SETS ADDITIONAL TEXT OF THE MESSAGE BOX
            obj_outbox.setIcon(QtGui.QMessageBox.Information) ## SETS THE ICON OF THE MESSAGE BOX
            obj_outbox.setStandardButtons(QtGui.QMessageBox.Ok | QtGui.QMessageBox.Cancel) ## SETS THE BUTTONS FOR THE MESSAGE BOX
            obj_choice = obj_outbox.exec() ## EXECUTES THE MESSAGE BOX AND RETRIEVES THE BUTTON PRESSED
            if obj_choice == QtGui.QMessageBox.Ok: ## CHECKS IF OK WAS PRESSED
                SetAnalysis.fetchresults(SetAnalysis_Window,self.lst_playset,str_curuser) ## PASSES THE NECESSARY PARAMETERS TO START FETCHING RESULTS FOR ANALYSIS
                SetAnalysis_Window.show() ## SHOWS THE ANALYSIS WINDOW
                Results_Window.hide() ## HIDES THE RESULTS WINDOW
        elif int_scorenum == 1: ## CHECKS IF USER HAS ONLY ATTEMPTED THE SET ONCE
            CreateOutbox("Not Enough Attempts!","You must attempt this quiz at least one more time before you can view your progress!","",QtGui.QMessageBox.Information)
        else:
            SetAnalysis.fetchresults(SetAnalysis_Window,self.lst_playset,str_curuser) ## PASSES THE NECESSARY PARAMETERS TO START FETCHING RESULTS FOR ANALYSIS
            SetAnalysis_Window.show() ## SHOWS THE ANALYSIS WINDOW


class Leaderboard(QtGui.QMainWindow,Leaderboard_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setupUi(self)
        self.backbutton.clicked.connect(self.Back)

    def fetchleaders(self,lst_playset):
        str_query = '''SELECT * FROM (SELECT Username,Score FROM Scores WHERE SID=? AND Username NOT IN
(SELECT Username FROM Teachers) ORDER BY Score ASC) AS allscores GROUP BY Username ORDER BY Score DESC LIMIT 10;''' ## FETCHES TOP 10 UNIQUE STUDENT SCORES
        obj_cur.execute(str_query,(lst_playset[2],)) ## EXECUTES THE QUERY WITH THE CORRECT SET ID
        lst_allscores = obj_cur.fetchall() ## FETCHES THE SCORES
        obj_model = QtGui.QStandardItemModel(len(lst_allscores),2) ## CREATES A TABLE MODEL
        for int_x in range(len(lst_allscores)): ## LOOPS THROUGH EACH SCORE
            for int_i in range(2): ## LOOPS TROUGH EACH COLUMN
                obj_model.setData(obj_model.index(int_x,int_i),str(lst_allscores[int_x][int_i])) ## SETS THE DATA FROM THE FETCHED SCORES TO THE MODEL
        self.leaderboard.setModel(obj_model) ## SETS THE MODEL TO THE LEADERBOARD
        self.leaderboard.setColumnWidth(0,175) ## SETS THE FIRST COLUMN'S WIDTH
        self.leaderboard.setColumnWidth(1,175) # SETS THE SECOND COLUMN'S WIDTH
        self.leaderboard.show() ## SHOWS THE LEADERBOARD

    def Back(self):
        Leaderboard_Window.hide() ## HIDES THE LEADERBOARD
```

```
class SetAnalysis(QtGui.QMainWindow,SetAnalysis_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.setupUi(self)
        self.backbutton.clicked.connect(self.Back)
        self.resultstable.clicked.connect(self.detailview) ## PASSES SELECTED ITEM TO DETAILVIEW FUNCTION

    def detailview(self,obj_index): ## INDEX IS THE ACTUAL ITEM WIDGET SELECTED
        int_score = int(obj_index.data()) ## GETS THE TEXT OF ITEM WIDGET SELECTED (THE SCORE)
        flt_scoreline = (((self.FLT_CEILING-self.INT_FLOOR)*(int_score-self.int_smallest))/(self.int_largest-self.int_smallest))+self.INT_FLOOR ## FINDS THE Y CO-ORDINATE OF THE SCORE LINE
        self.drawgraph(self.lst_pointslist,self.flt_zeroline,self.flt_xoffset,flt_scoreline,int_score)

    def fetchresults(self,lst_playset,str_user):
        self.str_user = str_user ## ALLOWS OTHER FUNCTIONS WITHIN CLASS TO USE USERNAME
        self.setlabel.setText(lst_playset[0]) ## SETS LABEL TEXT TO THE NAME OF THE SET
        self.resultstable.clear() ## CLEARS THE CURRENT LIST
        str_query = "SELECT Score FROM Scores WHERE SID=? AND Username=? ORDER BY Date DESC;" ## FETCHES ALL THE USER'S RESULTS IN THE CURRENT SET
        obj_cur.execute(str_query,(lst_playset[2],str_user,)) ## EXECUTES THE QUERY
        lst_allscores = obj_cur.fetchall() ## FETCHES THE DATA

        lst_plotlist = [] ## CONTAINS THE Y CO-ORDINATES OF POINTS TO BE PLOTTED

        for lst_score in lst_allscores: ## LOOPS THROUGH EACH SCORE
            obj_item = QtGui.QListWidgetItem() ## CREATES A NEW ITEM IN THE LIST
            obj_item.setText(str(lst_score[0])) ## SETS THE TEXT OF THE ITEM TO THE SCORE
            self.resultstable.addItem(obj_item) ## ADDS THE ITEM TO THE LIST WIDGET
        for lst_score in reversed(lst_allscores):
           lst_plotlist.append(lst_score[0]) ## ADDS THE CURRENT SCORE TO THE PLOTLIST

        self.flt_xoffset = (450/len(lst_plotlist)) ## GETS THE HORIZONTAL SPACING BETWEEN EACH POINT

        self.int_largest = max(lst_plotlist) ## FINDS THE HIGHEST SCORE
        self.int_smallest = min(lst_plotlist) ## FINDS THE LOWEST SCORE
        self.INT_FLOOR = 1 ## MINIMUM Y CO-ORDINATE OF POINT - CAPITALISED FOR CONSTANT
        self.FLT_CEILING = 28.5 ## MAXIMUM Y CO-ORDINATE OF POINT - CAPITALISED FOR CONSTANT

        self.flt_zeroline = (((self.FLT_CEILING-self.INT_FLOOR)*(0-self.int_smallest))/(self.int_largest-self.int_smallest))+self.INT_FLOOR
        for int_i in range(0,len(lst_plotlist)): ## LOOPS THROUGH EACH SCORE IN THE PLOTLIST
            lst_plotlist[int_i] = (((self.FLT_CEILING-self.INT_FLOOR)*(lst_plotlist[int_i]-self.int_smallest))/(self.int_largest-self.int_smallest))+self.INT_FLOOR

        self.lst_pointslist=",".join([str(int_x) for int_x in lst_plotlist]) ## JOINS TOGETHER ALL THE POINTS INTO ONE STRING SO IT'S FORMATTED CORRECTLY
        self.drawgraph(self.lst_pointslist,self.flt_zeroline,self.flt_xoffset,0,0)

    def drawgraph(self,lst_points,flt_zero,flt_xset,flt_scoreline,int_score):
        obj_file = open("GraphFile.txt","rt") ## OPENS THE GRAPH FILE
        str_contents = obj_file.read() ## SAVES THE CONTENTS OF THE FILE TO A VARIABLE
        str_contents = str_contents.replace("POINTS-LIST-HERE",lst_points) ## ADDS THE POINTSLIST TO THE HTML FILE
        str_contents = str_contents.replace("ZERO-COORDS-HERE",str(flt_zero)) ## ADDS THE ZEROLINE CO-ORDINATE TO THE HTML FILE
        str_contents = str_contents.replace("X-OFFSET-HERE",str(flt_xset)) ## ADDS THE XOFFSET TO THE HTML FILE
        if int_score != 0 and flt_scoreline != 0:
            str_contents = str_contents.replace("// SCORE-COORDS-HERE","var scorelocation="+str(flt_scoreline)+";") ## INSERTS THE Y-VALUE FOR THE SCORE
            str_contents = str_contents.replace("// SCORE-HERE","var score="+str(int_score)+";") ## INSERTS THE SCORE VALUE
            str_contents = str_contents.replace("// DRAW-SCORE-LINE",'''ctx.moveTo(x*10,canvas.height-yoffset*scorelocation*.5);
\n ctx.lineTo((x*10)+5,canvas.height-yoffset*scorelocation*.5);''') ## INSERTS THE CODE FOR DRAWING THE LINE
            str_contents = str_contents.replace("// WRITE-SCORE",
                                                "ctx.fillText(score,0,(canvas.height-yoffset*scorelocation*.5));") ## INSERTS THE CODE FOR WRITING THE SCORE
        self.webView.setHtml(str_contents) ## SETS THE VIEW OF THE WEB VIEWER IN THE WINDOW

    def Back(self):
        SetAnalysis_Window.hide() ## HIDES THE SET ANALYSIS WINDOW
```

```
class ViewStudents(QtGui.QMainWindow,ViewStudents_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.setupUi(self)
        self.backbutton.clicked.connect(self.Back)
        self.searchedit.textChanged.connect(self.search)
        self.addtoclassbutton.clicked.connect(self.addtoclass)
        self.removestudbutton.clicked.connect(self.removefromclass)
        self.resetbutton.clicked.connect(self.resetpass)
        self.addstudbutton.clicked.connect(self.add)
        self.mystudstable.doubleClicked.connect(self.viewresults)

    def findselection(self):
        lst_studentdetails = [] ## NO STUDENT SELECTED
        if self.studentTabs.currentIndex() == 0: ## CHECKS WHICH TAB IS CURRENTLY BEING VIEWED
            lst_selection = self.allstudstable.selectionModel().selectedRows() ## FINDS SELECTED ROWS IN ALL TABLE
            if lst_selection != []: ## CHECKS IF ANYTHING WAS SELECTED
                int_selectedrow = lst_selection[0].row() ## FINDS INDEX OF SELECTED ROW
                lst_studentdetails = self.lst_allstudents[int_selectedrow] ## GETS DETAILS OF SELECTED STUDENT
        else:
            lst_selection = self.mystudstable.selectionModel().selectedRows() ## FINDS SELECTED ROWS IN MY TABLE
            if lst_selection != []: ## CHECKS IF ANYTHING WAS SELECTED
                int_selectedrow = lst_selection[0].row() ## FINDS INDEX OF SELECTED ROW
                lst_studentdetails = self.lst_mystudents[int_selectedrow] ## GETS DETAILS OF SELECTED STUDENT
        return lst_studentdetails ## RETURNS THE DETAILS OF THE SELECTED STUDENT, OR AN EMPTY LIST

    def viewresults(self):
        str_studentusername = self.findselection()[2] ## FETCHES THE USERNAME OF THE CHOSEN STUDENT
        ViewResults.displayresults(ViewResults_Window,str_studentusername) ## BEGINS THE FUNCTION TO DISPLAY HIGH SCORES IN THE VIEW RESULTS WINDOW
        ViewResults_Window.show() ## SHOWS THE VIEW RESULTS WINDOW
        ViewStudents_Window.hide() ## HIDES THE VIEW STUDENTS WINDOW

    def addtoclass(self):
        lst_studentdetails = self.findselection() ## FINDS THE SELECTED STUDENT'S DETAILS
        if lst_studentdetails == []: ## CHECKS IF A STUDENT WAS SELECTED
            CreateOutbox("Selection Not Found","You must select a student to add to your class!","",QtGui.QMessageBox.Critical)
        else:
            str_query = 'SELECT * FROM STLinks WHERE SUser=? AND TUser=?;' ## TRIES TO FIND EXISTING LINK IN TABLE
            obj_cur.execute(str_query,(lst_studentdetails[2],str_curuser,)) ## EXECUTES QUERY
            lst_link = obj_cur.fetchone() ## FETCHES ANY DATA FROM DATABASE
            if lst_link != None: ## CHECKS IF A LINK WAS FOUND
                CreateOutbox("Conflict Found","This student is already in your class.","",QtGui.QMessageBox.Information)
            else:
                str_query = 'INSERT INTO STLinks (TUser,SUser) VALUES (?,?);' ## CREATES NEW LINK
                obj_cur.execute(str_query,(str_curuser,lst_studentdetails[2],)) ## EXECUTES QUERY
                obj_con.commit() ## WRITES CHANGES TO DATABASE
                CreateOutbox("Student Added","The selected student has been added to your class.","",QtGui.QMessageBox.Information)
                self.search() ## UPDATES THE TABLES

    def removefromclass(self):
        lst_studentdetails = self.findselection() ## FINDS THE SELECTED STUDENT'S DETAILS
        if lst_studentdetails == []: ## CHECKS IF A STUDENT WAS SELECTED
            CreateOutbox("Selection Not Found","You must select a student to remove from your class!","",QtGui.QMessageBox.Critical)
        else:
            str_query = 'DELETE FROM STLinks WHERE SUser=? AND TUser=?;' ## DELETES ANY LINK WITH THE SELECTED STUDENT AND TEACHER
            obj_cur.execute(str_query,(lst_studentdetails[2],str_curuser,)) ## EXECUTES QUERY
            obj_con.commit() ## WRITES CHANGES TO DATABASE
            CreateOutbox("Student Removed","The selected student has been removed from your class.","",QtGui.QMessageBox.Information)
            self.search() ## UPDATES THE TABLES

    def resetpass(self):
        lst_studentdetails = self.findselection() ## FIND THE SELECTED STUDENT'S DETAILS
        if lst_studentdetails == []: ## CHECKS IF A STUDENT WAS SELECTED
            CreateOutbox("Selection Not Found","You must select a student to reset their password!","",QtGui.QMessageBox.Critical)
        else:
            str_password = "Quizo123" ## SETS TEMPORARY PASSWORD IN PLAIN TEXT
            str_password = hashlib.sha256(str_password.encode()).hexdigest() ## HASHES PASSWORD FOR DATABASE
            str_query = 'REPLACE INTO Students (Username,Password,FirstName,Surname) VALUES (?,?,?,?);' ## QUERY TO CHANGE PASSWORD
            obj_cur.execute(str_query,(lst_studentdetails[2],str_password,lst_studentdetails[0],lst_studentdetails[1],)) ## EXECUTES THE QUERY
            obj_con.commit() ## WRITES CHAGNES TO DATABASE
            CreateOutbox("Password Reset","Password has been reset to 'Quizo123'.","",QtGui.QMessageBox.Information)

    def viewtable(self,obj_table,lst_students):
        obj_model = QtGui.QStandardItemModel(len(lst_students),3) ## CREATES A MODEL TO PUT THE DATA IN
        for int_row in range(0,len(lst_students)): ## LOOPS THROUGH EACH STUDENT
            for int_column in range(0,3): ## LOOPS THROUGH USERNAME, FIRST NAME, AND SURNAME
                obj_model.setData(obj_model.index(int_row,int_column),str(lst_students[int_row][int_column])) ## SETS THE MODEL DATA ACCORDING TO THE FETCHED DATA
        obj_table.setModel(obj_model) ## SETS THE MODEL TO THE TABLE VIEW
        lst_tempwidth = [250,250,225] ## COLUMN WIDTHS FOR THE TABLE
        for int_i in range(3): ## LOOPS THROUGH EACH COLUMN
            obj_table.setColumnWidth(int_i,lst_tempwidth[int_i]) ## SETS THE WIDTH OF THE COLUMN
        obj_table.show() ## SHOWS THE TABLE

    def search(self):
        str_searchword = "%"+self.searchedit.text()+"%" ## FORMATS THE SEARCH WORD FOR USE IN THE SQL STATEMENT
        str_query = 'SELECT FirstName,Surname,Username FROM Students WHERE Students.Username LIKE ? or Students.FirstName LIKE ? or Students.Surname LIKE ?;' ## QUERY TO FETCH ALL STUDENTS
        obj_cur.execute(str_query,(str_searchword,str_searchword,str_searchword,)) ## EXECUTES QUERY
        self.lst_allstudents = obj_cur.fetchall() ## FETCHES DATA FROM THE DATABASE
        str_query = '''SELECT Students.FirstName,Students.Surname,Students.Username FROM Students INNER JOIN STLinks on Students.Username=STLinks.SUser WHERE STLinks.TUser LIKE ?
and (Students.Username LIKE ? or Students.FirstName LIKE ? or Students.Surname LIKE ?);''' ## QUERY TO FETCH ALL STUDENTS FROM DATABASE
        obj_cur.execute(str_query,(str_curuser,str_searchword,str_searchword,str_searchword,)) ## EXECUTES QUERY
        self.lst_mystudents = obj_cur.fetchall() ## FETCHES DATA FROM THE DATABASE
        self.viewtable(self.mystudstable,self.lst_mystudents) ## DISPLAYS THE MY STUDENTS TABLE
        self.viewtable(self.allstudstable,self.lst_allstudents) ## DISPLAYS THE ALL STUDENTS TABLE

    def add(self):
        StudentRegistration_Window.show() ## SHOWS THE STUDENT REGISTRATION WINDOW
        ViewStudents_Window.hide() ## HIDES THE VIEW STUDENTS WINDOW

    def Back(self):
        self.searchedit.setText("") ## CLEARS THE SEARCH BAR TEXT
        TeacherHome_Window.show() ## SHOWS THE TEACHER HOME WINDOW
        ViewStudents_Window.hide() ## HIDES THE VIEW STUDENTS WINDOW
```

```
class ViewResults(QtGui.QMainWindow,ViewResults_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.setupUi(self)
        self.backbutton.clicked.connect(self.Back)
        self.viewbutton.clicked.connect(self.analyse)
        self.resultstable.doubleClicked.connect(self.analyse) ## SHOWS RESULTS WHEN SET IS DOUBLE CLICKED

    def analyse(self):
        lst_selection = self.resultstable.selectionModel().selectedRows() ## FINDS SELECTED ROWS IN TABLE
        if lst_selection != []: ## CHECKS IF ANYTHING WAS SELECTED
            int_selectedrow = lst_selection[0].row() ## FINDS INDEX OF SELECTED ROW
            lst_setdetails = self.attemptedsets[int_selectedrow] ## GETS THE SET DETAILS FOR SELECTED SET (Name,Score,ID)
            str_query = "SELECT COUNT(Score) FROM Scores WHERE SID=? AND Username=?;" ## QUERY THAT FETCHES THE NUMBER OF SCORES BY THE USER IN A SET
            obj_cur.execute(str_query,(lst_setdetails[2],self.str_username,)) ## EXECUTES THE QUERY WITH THE CORRECT PARAMETERS
            int_scorenum = obj_cur.fetchone()[0] ## FETCHES THE ACTUAL NUMBER OF SCORES
            if int_scorenum > 1000:
                obj_outbox = QtGui.QMessageBox() ## CREATES THE MESSAGE BOX
                obj_outbox.setWindowTitle("Set Analysis") ## SETS THE TITLE OF THE MESSAGE BOX
                obj_outbox.setText("You are trying to analyse a set where there are more than 1000 scores.") ## SETS THE MAIN TEXT OF THE MESSAGE BOX
                obj_outbox.setInformativeText('''This may take some time to process, would you like to continue?
\n(If there are more than 20,000 results, it is possible the program will crash)''') ## SETS ADDITIONAL TEXT OF THE MESSAGE BOX
                obj_outbox.setIcon(QtGui.QMessageBox.Information) ## SETS THE ICON OF THE MESSAGE BOX
                obj_outbox.setStandardButtons(QtGui.QMessageBox.Ok | QtGui.QMessageBox.Cancel) ## SETS THE BUTTONS FOR THE MESSAGE BOX
                obj_choice = obj_outbox.exec() ## EXECUTES THE MESSAGE BOX AND RETRIEVES THE BUTTON PRESSED
                if obj_choice == QtGui.QMessageBox.Ok: ## CHECKS IF OK WAS PRESSED
                    SetAnalysis.fetchresults(SetAnalysis_Window,lst_setdetails,self.str_username) ## PASSES THE NECESSARY PARAMETERS TO START FETCHING RESULTS FOR ANALYSIS
                    SetAnalysis_Window.show() ## SHOWS THE ANALYSIS WINDOW
                    Results_Window.hide() ## HIDES THE RESULTS WINDOW
            elif int_scorenum == 1: ## CHECKS IF USER HAS ONLY ATTEMPTED THE SET ONCE
                CreateOutbox("Not Enough Attempts!","At least two attempts are required to review progress.","",QtGui.QMessageBox.Information)
            else:
                SetAnalysis.fetchresults(SetAnalysis_Window,lst_setdetails,self.str_username) ## PASSES THE NECESSARY PARAMETERS TO START FETCHING RESULTS FOR ANALYSIS
                SetAnalysis_Window.show() ## SHOWS THE ANALYSIS WINDOW
        else:
            CreateOutbox("No Set Selected","You must select a question set to analyse!","",QtGui.QMessageBox.Critical)

    def displayresults(self,str_username):
        self.str_username = str_username ## ALLOWS ALL FUNCTIONS WITHIN THE CLASS TO USE THE USERNAME
        str_query = '''SELECT * FROM (SELECT Sets.SetName,Scores.Score,Scores.SID FROM Scores INNER JOIN Sets ON Scores.SID=Sets.SID WHERE Username=?
ORDER BY Score ASC) AS allscores GROUP BY SID ORDER BY Score DESC;''' ## FETCHES A USER'S HIGHEST SCORE IN ALL ATTEMPTED SETS
        obj_cur.execute(str_query,(str_username,)) ## EXECUTES QUERY
        self.attemptedsets = obj_cur.fetchall() ## FETCHS DATA FROM DATABASE
        obj_model = QtGui.QStandardItemModel(len(self.attemptedsets),2) ## CREATES A MODEL TO PUT THE DATA IN
        for int_row in range(0,len(self.attemptedsets)): ## LOOPS THROUGH EACH SET
            for int_column in range(0,2): ## LOOPS THROUGH SET NAME AND SCORE FOR THE CURRENT SET
                obj_model.setData(obj_model.index(int_row,int_column),str(self.attemptedsets[int_row][int_column])) ## SETS THE MODEL DATA ACCORDING TO THE FETCHED DATA
        self.resultstable.setModel(obj_model) ## SETS THE MODEL TO THE TABLE VIEW
        lst_tempwidth = [500,265] ## COLUMN WIDTHS FOR THE TABLE
        for int_i in range(2): ## LOOPS THROUGH EACH COLUMN
            self.resultstable.setColumnWidth(int_i,lst_tempwidth[int_i]) ## SETS THE WIDTH OF THE COLUMN
        self.resultstable.show() ## SHOWS THE TABLE

    def Back(self):
        if str_curuser == self.str_username: ## CHECKS IF USER IS A STUDENT OR TEACHER
            StudentHome_Window.show() ## SHOWS THE STUDENT HOME WINDOW
        else:
            ViewStudents_Window.show() ## SHOWS THE VIEW STUDENTS WINDOW
        ViewResults_Window.hide() ## HIDES THE VIEW RESULTS WINDOW
```

```
class StudentRegistration(QtGui.QMainWindow,StudentRegistration_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.setupUi(self)
        self.backbutton.clicked.connect(self.Back)
        self.createbutton.clicked.connect(self.addstudent) ## STARTS ADD STUDENT FUNCTION WHEN CREATE PRESSED
        self.useredit.returnPressed.connect(self.addstudent) ## STARTS ADD STUDENT FUNCTION WHEN ENTER IS PRESSED
        self.firstedit.returnPressed.connect(self.addstudent)
        self.suredit.returnPressed.connect(self.addstudent)

    def addstudent(self):
        str_username = self.useredit.text() ## FETCHES CONTENTS OF USERNAME BOX
        str_firstname = self.firstedit.text() ## FETCHES CONTENTS OF FIRSTNAME BOX
        str_surname = self.suredit.text() ## FETCHES CONTENTS OF SURNAME BOX
        lst_errors = [] ## RESETS THE ERRORS WHEN ADDING A NEW ACCOUNT
        obj_valid = re.match("^[a-zA-Z\d]{3,15}$",str_username) ## CHECKS USERNAME IS VALID
        if not obj_valid:
            lst_errors.append("Username must be alphanumerical only\nUsername must be between 3 and 15 characters long")
        else:
            lst_tables = ["Teachers","Students"] ## LIST CONTAINING TABLE NAMES TO BE SEARCHED
            for str_x in lst_tables: ## STARTS A FOR LOOP FOR SEARCHING THE TABLES
                str_query = "SELECT Username FROM ? WHERE Username=?;" ## CREATES THE QUERY FOR SEARCHING THE CURRENT TABLE
                str_query = str_query.replace("?",str_x,1) ## REPLACES THE X VALUE
                obj_cur.execute(str_query,(str_username,)) ## EXECUTES THE QUERY
                lst_temp = obj_cur.fetchall() ## FETCHES ANY DATA FROM THE DATABASE THAT WAS SELECTED
                if lst_temp!= []: ## CHECKS IF ANY DATA WAS FETCHED
                    lst_errors.append("Username already taken")
                    break ## BREAKS OUT OF THE LOOP IF THE USERNAME HAS BEEN FOUND
        obj_valid = re.match("^[a-zA-Z]{1,}$",str_firstname) ## CHECKS IF FIRST NAME IS VALID WITH REGEX
        if not obj_valid:
            lst_errors.append('''First name must contain only alphanumerical characters\nFirst name must be filled in''')
        else:
            str_firstname = str_firstname.title() ## TITLISES FIRST NAME
        obj_valid = re.match("(?=[a-zA-Z-]{1,}$)^[a-zA-Z]+(-[a-zA-Z]+)*$",str_surname) ## USES REGEX TO CHECK IF VALID
        if not obj_valid:
            lst_errors.append('''Surname must only contain alphanumerical characters
Surname can contain a maximum of one dash between two barrels
Surname must be filled in''')
        else:
            str_surname = str_surname.title() ## TITLISES THE SURNAME
        if lst_errors != []: ## CHECKS IF USER HAS HAD ANY ERRORS
            str_errormessage = "\n".join(lst_errors) ## CONVERTS LIST OF ERRORS INTO SINGLE STRING
            CreateOutbox("Errors Found","There has been an error with creating this account:",str_errormessage,QtGui.QMessageBox.Critical) ## CREATES ERROR MESSAGE BOX
        else:
            str_password = hashlib.sha256("Quizo123".encode()).hexdigest() ## HASHES THE PASSWORD FOR STORING IN THE DATABASE
            str_query = 'INSERT INTO Students (Username,Password,FirstName,Surname) VALUES (?,?,?,?);' ## INSERTS NEW ACCOUNT INTO DATABASE WITH CORRECT DATA
            obj_cur.execute(str_query,(str_username,str_password,str_firstname,str_surname,)) ## EXECUTES THE QUERY
            str_query = 'INSERT INTO STLinks (TUser,SUser) VALUES (?,?);' ## INSERTS NEW LINK BETWEEN TEACHER AND STUDENT
            obj_cur.execute(str_query,(str_curuser,str_username,)) ## EXECUTES THE QUERY
            obj_con.commit() ## WRITES CHANGES TO THE DATABASE
            CreateOutbox("Account Created","This new account has been created!","",QtGui.QMessageBox.Information) ## CREATES ACCOUNT CREATED MESSAGE BOX

    def Back(self):
        self.useredit.setText("") ## CLEARS THE USERNAME FIELD
        self.firstedit.setText("") ## CLEARS THE FIRSTNAME FIELD
        self.suredit.setText("") ## CLEARS THE SURNAME FIELD
        StudentRegistration_Window.hide() ## HIDES THE STUDENT REGISTRATION WINDOW
        ViewStudents.search(ViewStudents_Window) ## DISPLAYS THE TABLE
        ViewStudents_Window.show() ## SHOWS THE VIEW STUDENTS WINDOW
'
```

```
class ViewSets(QtGui.QMainWindow,ViewSets_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.setupUi(self)
        self.backbutton.clicked.connect(self.Back) ## RUNS BACK FUNCTION WHEN BACK BUTTON PRESSED
        self.addsetbutton.clicked.connect(self.addset) ## RUNS ADD SET FUNCTION WHEN ADD BUTTON PRESSED
        self.editsetbutton.clicked.connect(self.editset) ## RUNS EDIT SET FUNCTION WHEN EDIT BUTTON PRESSED
        self.searchedit.textChanged.connect(self.search) ## RUNS SEARCH FUNCTION WHENEVER SEARCH BAR CONTENTS CHANGE
        self.leaderboardbutton.clicked.connect(self.leaders) ## RUNS LEADERS FUNCTION WHEN LEADERBOARD BUTTON PRESSED
        self.setstable.doubleClicked.connect(self.editset) ## STARTS EDIT SET FUNCTION WHEN SET DOUBLE CLICKED

    def leaders(self):
        lst_selection = self.setstable.selectionModel().selectedRows() ## FETCHES THE SELECTED ROWS FROM THE TABLE
        if lst_selection == []: ## CHECKS IF ANY ROW HAS BEEN SELECTED
            CreateOutbox("Selection Not Found","You must select a question set to view the leaderboard!","",QtGui.QMessageBox.Critical)
        else:
            int_selectedrow = lst_selection[0].row() ## FINDS INDEX OF SELECTED ROW
            lst_setdetails = self.lst_allsets[int_selectedrow] ## SETS THE CURRENT PLAYSET TO THE SELECTED SET
            PrintLeaders.displaytable(PrintLeaders_Window,lst_setdetails) ## DISPLAYS THE LEADERBOARD
            PrintLeaders_Window.show() ## SHOWS PRINT LEADERS WINDOW
            ViewSets_Window.hide() # HIDES VIEW SETS WINDOW

    def editset(self):
        lst_selection = self.setstable.selectionModel().selectedRows() ## FETCHES THE SELECTED ROWS FROM THE TABLE
        if lst_selection == []: ## CHECKS IF ANY ROW HAS BEEN SELECTED
            CreateOutbox("Selection Not Found","You must select a question set to edit!","",QtGui.QMessageBox.Critical)
        else:
            int_selectedrow = lst_selection[0].row() ## FINDS INDEX OF SELECTED ROW
            lst_setdetails = self.lst_allsets[int_selectedrow] ## SETS THE CURRENT PLAYSET TO THE SELECTED SET
            if lst_setdetails[1] == str_curuser: ## CHECKS THAT THE CURRENT USER IS THE AUTHOR OF THE SET
                str_query = '''SELECT Questions.Question,Questions.Correct,Questions.Wrong1,Questions.Wrong2,Questions.Wrong3
    FROM Questions INNER JOIN QSLinks ON Questions.QID=QSLinks.QID WHERE QSLinks.SID=?;''' ## FETCHES QUESTION FOR GIVEN SET
                obj_cur.execute(str_query,(lst_setdetails[2],)) ## EXECUTES QUERY
                lst_setquestions = obj_cur.fetchall() ## FETCHES ALL QUESTIONS
                AddQuestions.displayquestions(AddQuestions_Window,lst_setdetails[0],lst_setquestions) ## CREATES EMPTY TABLE
                AddQuestions_Window.show() ## SHOWS THE ADD QUESTIONS WINDOW
                Importing_Window.hide() ## HIDES THE IMPORTING WINDOW
            else:
                CreateOutbox("Selection Error","You may only edit sets that you have created!","",QtGui.QMessageBox.Information)

    def addset(self):
        Importing_Window.show() ## SHOWS THE IMPORTING WINDOW
        ViewSets_Window.hide() ## HIDES THE VIEW SETS WINDOW

    def search(self):
        str_word = "%" + self.searchedit.text() + "%" ## GETS STRING FROM SEARCH BAR
        for chr_char in ["'",";",'"']: ## LOOPS THROUGH THE DISALLOWED CHARACTERS
            str_word = str_word.replace(chr_char,"") ## REMOVES CHARACTERS FROM USER'S ENTRY
        str_query = 'SELECT SetName,Author,SID FROM Sets WHERE SetName LIKE ? OR Author LIKE ?;' ## QUERY TO FETCH SET NAMES WITH SIMILAR TITLES OR AUTHORS TO USER'S SEARCH
        obj_cur.execute(str_query,(str_word,str_word,)) ## EXECUTES QUERY
        self.lst_allsets = obj_cur.fetchall() ## FETCHES DATA FROM DATABASE
        obj_model = QtGui.QStandardItemModel(len(self.lst_allsets),2) ## CREATES A MODEL TO PUT THE DATA IN
        for int_row in range(0,len(self.lst_allsets)): ## LOOPS THROUGH EACH SET
            for int_column in range(0,2): ## LOOPS THROUGH SET NAME AND AUTHOR FOR THE CURRENT SET
                obj_model.setData(obj_model.index(int_row,int_column),str(self.lst_allsets[int_row][int_column])) ## SETS THE MODEL DATA ACCORDING TO THE FETCHED DATA
        self.setstable.setModel(obj_model) ## SETS THE MODEL TO THE TABLE VIEW
        self.setstable.setColumnWidth(0,500) ## SETS COLUMN WIDTH FOR SETS COLUMN
        self.setstable.setColumnWidth(1,265) ## SETS COLUMN WIDTH FOR AUTHOR COLUMN
        self.setstable.show() ## SHOWS THE TABLE

    def Back(self):
        self.searchedit.setText("") ## CLEARS THE SEARCH BAR
        TeacherHome_Window.show() ## SHOWS THE TEACHER HOME WINDOW
        ViewSets_Window.hide() ## HIDES THE VIEW SETS WINDOW

class Importing(QtGui.QMainWindow,Importing_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.setupUi(self)
        self.importbutton.clicked.connect(self.imp) ## RUNS THE IMP FUNCTION WHEN IMPORT IS CLICKED
        self.createbutton.clicked.connect(self.create) ## RUNS THE CREATE FUNTION WHEN CREATE IS CLICKED
        self.backbutton.clicked.connect(self.back) ## RUNS THE BACK FUNCTION WHEN BACK IS CLICKED

    def back(self):
        ViewSets_Window.show() ## SHOWS THE VIEW SETS WINDOW
        Importing_Window.hide() ## HIDES THE IMPORTING WINDOW

    def imp(self):
        if (self.tab.isChecked() != True) and (self.comma.isChecked() != True): ## CHECKS WHETHER A CHECKBOX HAS BEEN TICKED
            CreateOutbox("Notification","Please select an import option.","",QtGui.QMessageBox.Critical)
        else:
            obj_browser = Tk() ## CREATES A TKINTER WINDOW
            obj_browser.withdraw() ## REMOVES THE WINDOW SO IT EXISTS BUT CAN'T BE SEEN
            obj_browser.filename = filedialog.askopenfilename(initialdir = "/",title = "Select File",filetypes = (("Text File","*.txt"),("CSV File","*.csv")))
            if obj_browser.filename != "": ## CHECKS IF A FILE HAS BEEN SELECTED
                obj_file = open(obj_browser.filename,"rt") ## OPENS THE SELECTED FILE
                str_contents = obj_file.read() ## READS THE CONTENTS OF THE FILE TO A STRING
                lst_temp = str_contents.split("\n") ## SPLITS THE CONTENTS INTO EACH QUESTION BY LINE
                lst_importedquestions = [] ## LIST OF ALL IMPORTED QUESTIONS
                for str_each in lst_temp: ## LOOPS THROUGH EACH LINE OF THE FILE
                    if str_each !='': ## CHECKS IF LINE IS EMPTY
                        if self.tab.isChecked() == True: ## CHECKS WHICH TYPE OF DELIMITER HAS BEEN CHOSEN
                            lst_each = str_each.split("\t") ## SPLITS THE ROW BY TAB
                        else:
                            lst_each = str_each.split(",") ## SPLITS THE ROW BY COMMA
                        lst_importedquestions.append(lst_each) ## ADDS THE LIST TO THE QUESTION LIST
                AddQuestions.displayquestions(AddQuestions_Window,"",lst_importedquestions) ## CREATES TABLE WITH IMPORTED QUESTIONS
                AddQuestions_Window.show() ## SHOWS THE ADD QUESTIONS WINDOW
                Importing_Window.hide() ## HIDES THE IMPORTING WINDOW

    def create(self):
        AddQuestions.displayquestions(AddQuestions_Window,"",[]) ## CREATES EMPTY TABLE
        AddQuestions_Window.show() ## SHOWS THE ADD QUESTIONS WINDOW
        Importing_Window.hide() ## HIDES THE IMPORTING WINDOW
```

```
class AddQuestions(QtGui.QMainWindow,AddQuestions_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.setupUi(self)
        self.finishbutton.clicked.connect(self.finish)
        self.addbutton.clicked.connect(self.add)
        self.deletebutton.clicked.connect(self.delete)
        self.backbutton.clicked.connect(self.Back)

    def displayquestions(self,str_name,lst_questions):
        self.titleedit.setText(str_name) ## SETS TITLE OF SET
        self.obj_model = QtGui.QStandardItemModel(len(lst_questions),5) ## CREATES A TABLE MODEL
        try: ## ATTEMPTS TO EXECUTE THE FOLLOWING CODE, OTHERWISE WILL JUMP TO EXCEPT
            for int_x in range(0,len(lst_questions)): ## LOOPS THROUGH EACH QUESTION IN QUESTIONS LIST
                for int_i in range(0,5): ## LOOPS THROUGH EACH COLUMN
                    self.obj_model.setData(self.obj_model.index(int_x,int_i),str(lst_questions[int_x][int_i])) ## SETS DATA IN TABLE
        except:
            self.obj_model = QtGui.QStandardItemModel(0,5) ##CREATES EMPTY MODEL
            CreateOutbox("Notification","There was an issue importing your questions, ensure they are in the format you specified and please try again.",
                         "",QtGui.QMessageBox.Critical)
        self.questiontable.setModel(self.obj_model) ## SETS MODEL TO TABLE
        lst_tempwidth = [210,153,139,139,139] ## LIST OF COLUMN WIDTHS
        for int_i in range(5): ## LOOPS THROUGH EACH COLUMN
            self.questiontable.setColumnWidth(int_i,lst_tempwidth[int_i]) ## SETS WIDTH OF COLUMN
        self.questiontable.show() ## DISPLAYS TABLE

    def Back(self):
        ViewSets_Window.show() ## SHOWS THE VIEW SETS WINDOW
        AddQuestions_Window.hide() ## HIDES THE ADD QUESTIONS WINDOW

    def add(self):
        self.obj_model.appendRow(QtGui.QStandardItem()) ## ADDS NEW ROW TO TABLE

    def delete(self):
        lst_selection = self.questiontable.selectionModel().selectedIndexes() ## FETCHES THE SELECTED ITEMS FROM THE TABLE
        if lst_selection == []: ## CHECKS IF ANY ROW HAS BEEN SELECTED
            CreateOutbox("Notification","You must select a question to remove!","",QtGui.QMessageBox.Critical)
        else:
            int_selectedrow = lst_selection[0].row() ## GETS ROW NUMBER OF SELECTED ITEM
            self.obj_model.removeRow(int_selectedrow) ## REMOVES ROW FROM TABLE MODEL

    def finish(self):
        str_title = self.titleedit.text() ## GET TEXT FROM TITLE ENTRY BOX
        bln_valid = re.match("^[A-Za-z0-9](?!.*-{2})(?!.* {2})[A-Za-z0-9 -]{2,39}$",str_title) ## USES REGULAR EXPRESSION TO CHECK IF VALID
        if not bln_valid:
            CreateOutbox("Invalid Title",
                         '''The title you have entered is invalid. All titles must:","Contain only alphanumerical characters, spaces, or dashes
Have no double dashes
Have no double spaces
Begin with an alphanumerical character
Be between 3 and 40 characters in length''',QtGui.QMessageBox.Critical)

        else:
            str_query = 'SELECT SID,SetName,Author FROM Sets WHERE SetName=? AND Author=?;' ## CHECKS IF USER HAS PREVIOUSLY MADE A SET WITH THIS NAME
            obj_cur.execute(str_query,(str_title,str_curuser,)) ## EXECUTES QUERY
            lst_setdetails = obj_cur.fetchall() ## FETCHES ANY FOUND DETAILS
            if lst_setdetails != []: ## CHECKS IF A SET WAS FOUND
                obj_outbox = QtGui.QMessageBox() ## CREATES A MESSAGE BOX
                obj_outbox.setStandardButtons(QtGui.QMessageBox.Ok | QtGui.QMessageBox.Cancel) ## SETS BUTTONS FOR BOX
                obj_outbox.setWindowTitle("Overwrite Quiz") ## SETS TITLE
                obj_outbox.setIcon(QtGui.QMessageBox.Critical) ## SETS ICON
                obj_outbox.setText("You already have a set with this title, would you like to replace the existing questions?") ## SETS TEXT
                if obj_outbox.exec() == QtGui.QMessageBox.Ok: ## CHECKS IF OK WAS PRESSED
                    str_query = 'DELETE FROM Questions WHERE Questions.QID IN (SELECT QSLinks.QID FROM QSLinks WHERE QSLinks.SID=?);' ## DELETES OLD QUESTIONS
                    obj_cur.execute(str_query,(lst_setdetails[0][0],))
                    str_query = 'DELETE FROM Scores WHERE SID=?;' ## DELETES ANY OLD SCORES
                    obj_cur.execute(str_query,(lst_setdetails[0][0],))
                    str_query = 'DELETE FROM QSLinks WHERE QSLinks.SID=?;' ## DELETES OLD LINKS
                    obj_cur.execute(str_query,(lst_setdetails[0][0],))
                    str_query = 'DELETE FROM Sets WHERE SID=?;' ## DELETES OLD SET
                    obj_cur.execute(str_query,(lst_setdetails[0][0],))
                    obj_con.commit() ## COMMITS CHANGES TO DATABASE
                    self.writechanges(str_title)
            else:
                self.writechanges(str_title)
```

```
    def writechanges(self,str_title):
        lst_questions = [] ## LIST OF QUESTIONS TO WRITE TO DATABASE
        for int_row in range(self.obj_model.rowCount()): ## LOOPS THROUGH EACH ROW IN TABLE
            lst_tempquestion = [] ## MAKES A TEMPORARY LIST FOR CURRENT ROW
            for int_column in range(5): ## LOOPS THROUGH EACH COLUMN IN TABLE
                obj_index = self.obj_model.index(int_row,int_column) ## FINDS INDEX OF CURRENT CELL
                str_item = str(self.obj_model.data(obj_index)) ## GETS TEXT FROM CURRENT CELL
                str_item = str_item.replace('"','').replace("'","").replace(";","") ## REMOVES ANY APOSTROPES,QUOTE MARKS, OR SEMI-COLONS FOR VALIDATION
                lst_tempquestion.append(str_item) ## ADDS CELL DATA TO TEMPORARY LIST
            if ("None" not in lst_tempquestion) and ("" not in lst_tempquestion): ## CHECKS IF ROW HAS ANY INCOMPLETE DATA
                lst_questions.append(lst_tempquestion) ## ADDS QUESTION TO LIST OF QUESTIONS

        str_query = "SELECT SID FROM Sets;" ## FETCHES ALL SET IDS FROM DATABASE
        obj_cur.execute(str_query) ## EXECUTES QUERY
        lst_allsids = [lst_sub[0] for lst_sub in obj_cur.fetchall()] ## LIST OF ALL SET IDS IN A USABLE FORMAT
        int_checkid = 1 ## SETS STARTING ID FOR FINDING AN ID FOR THE NEW SET
        bln_allocated = False
        while bln_allocated == False: ## CHECKS IF AN ID HAS BEEN FOUND FOR THE SET
            if int_checkid not in lst_allsids: ## CHECKS IF THE CURRENT ID IS ALREADY TAKEN
                int_setid = str(int_checkid) ## SETS THE ID OF THE SET
                str_query = "INSERT INTO Sets VALUES (?,?,?);" ## ADDS NEW SET TO DATABASE
                obj_cur.execute(str_query,(int_setid,str_title,str_curuser,)) ## EXECUTES QUERY
                obj_con.commit() ## WRITES CHANGES TO DATABASE
                bln_allocated = True ## PREVENTS FURTHER SEARCHING FOR AN ID
            else:
                int_checkid += 1 ## INCREMENTS THE ID TO CHECK

        str_query = "SELECT QID FROM Questions;" ## FETCHES ALL QUESTION IDS FROM DATABASE
        obj_cur.execute(str_query) ## EXECUTES QUERY
        lst_allqids = [lst_sub[0] for lst_sub in obj_cur.fetchall()] ## LIST OF ALL QUESTION IDS IN A USABLE FORMAT
        int_checkid = 1 ## SETS STARTING ID FOR FINDING AN ID FOR THE QUESTIONS
        for lst_question in lst_questions: ## LOOPS THROUGH EACH QUESTION THAT NEEDS TO BE ADDED TO THE DATABASE
            bln_allocated = False
            while bln_allocated == False: ## CHECKS IF AN ID HAS BEEN FOUND FOR THIS QUESTION YET
                if int_checkid not in lst_allqids:
                    str_query = "INSERT INTO Questions VALUES (?,?,?,?,?,?);" ## ADDS QUESTION TO DATABASE
                    obj_cur.execute(str_query,(str(int_checkid),lst_question[0],lst_question[1],lst_question[2],lst_question[3],lst_question[4],)) ## EXECUTES QUERY
                    str_query = "INSERT INTO QSLinks VALUES (?,?);" ## ADDS LINK TO DATABASE
                    obj_cur.execute(str_query,(int_setid,str(int_checkid),)) ## EXECUTED QUERY
                    obj_con.commit() ## WRITES CHANGES TO DATABASE
                    bln_allocated = True ## PREVENTS FURTHER SEARCHING FOR AN ID FOR THIS QUESTION
                int_checkid += 1 ## INCREMENTS THE ID TO CHECK
        CreateOutbox("Question Set Added","Your questions have been added to the database!","",QtGui.QMessageBox.Information)
        ViewSets.search(ViewSets_Window) ## UPDATES THE TABLE BEFORE THE WINDOW OPENS
        ViewSets_Window.show() ## SHOWS THE VIEW SETS WINDOW
        AddQuestions_Window.hide() ## HIDES THE ADD QUESTIONS WINDOW



class PrintLeaders(QtGui.QMainWindow,PrintLeaders_class):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.setupUi(self)
        self.printButton.clicked.connect(self.printing)
        self.previewButton.clicked.connect(self.preview)
        self.backButton.clicked.connect(self.Back)

    def displaytable(self,lst_setdetails):
        self.obj_cursor = self.textEdit.textCursor() ## CREATES CURSOR FOR TEXT EDIT
        obj_titleformat = QtGui.QTextCharFormat() ## CREATES FORMAT FOR THE TITLE
        obj_titleformat.setFontUnderline(True) ## UNDERLINES TITLE
        obj_titleformat.setFontPointSize(30) ## SETS FONT SIZE TO 30
        obj_tableformat = QtGui.QTextTableFormat() ## CREATES FORMAT FOR TABLE
        obj_tableformat.setAlignment(QtCore.Qt.AlignHCenter) ## ALIGNS ALL TEXT IN CENTRE
        self.textEdit.setAlignment(QtCore.Qt.AlignHCenter) ## ALIGNS ALL TEXT IN CENTRE
        self.obj_cursor.insertText(lst_setdetails[0],obj_titleformat) ## INSERTS TITLE

        str_query = '''SELECT * FROM (SELECT Students.Username,FirstName,Surname,Score FROM Scores
INNER JOIN Students ON Scores.Username=Students.Username WHERE SID=? AND Students.Username NOT IN
(SELECT Username FROM Teachers) ORDER BY Score ASC) AS allscores GROUP BY Username ORDER BY Score DESC;'''
        obj_cur.execute(str_query,(lst_setdetails[2],)) ## EXECUTES QUERY
        lst_scores = obj_cur.fetchall() ## FETCHES ALL SCORES
        int_studentnum = len(lst_scores) ## GETS NUMBER OF STUDENTS
        lst_scores.insert(0,('0','0','0','0')) ## NEEDED AS INDEXING STARTS FROM 0

        int_tempnum = int_studentnum - 78 ## TEMPORARY NUMBER OF STUDENTS FOR FINDING HOW MANY PAGES ARE NEEDED
        int_pages = 0 ## RESETS HOW MANY PAGES ARE NEEDED
        while int_tempnum > 0:
            int_tempnum -= 84 ## REMOVES ONE PAGE'S WORTH OF STUDENTS FROM TEMPNUM
            int_pages += 1 ## ADDS A PAGE
        int_totalrows = 41 + (int_pages*42) ## FINDS TOTAL NUMBER OF ROWS NEEDED

        self.obj_cursor.insertTable(int_totalrows,7,obj_tableformat) ## INSERTS TABLE
        self.obj_cursor.insertText("Position")
        self.obj_cursor.movePosition(self.obj_cursor.NextCell)
        self.obj_cursor.insertText("Name")
        self.obj_cursor.movePosition(self.obj_cursor.NextCell)
        self.obj_cursor.insertText("Score")
        self.obj_cursor.movePosition(self.obj_cursor.NextCell)
        self.obj_cursor.insertText("\t")
        self.obj_cursor.movePosition(self.obj_cursor.NextCell)
        self.obj_cursor.insertText("Position")
        self.obj_cursor.movePosition(self.obj_cursor.NextCell)
        self.obj_cursor.insertText("Name")
        self.obj_cursor.movePosition(self.obj_cursor.NextCell)
        self.obj_cursor.insertText("Score")
        self.obj_cursor.movePosition(self.obj_cursor.NextCell)
```

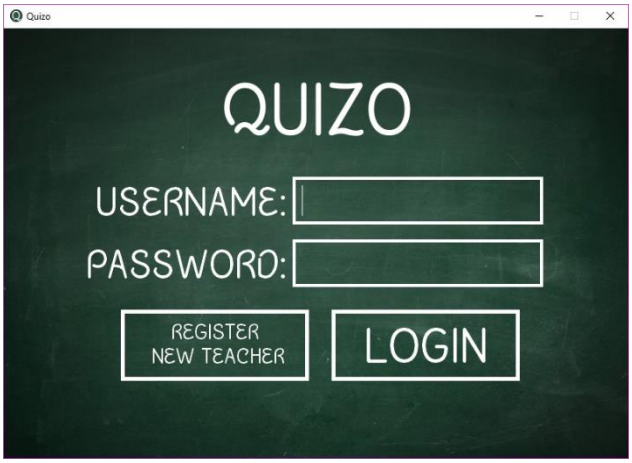
```
        int_page = 1 ## SETS PAGE NUMBER
        int_columnvalue = 1 ## STARTS ON LEFT COLUMN
        int_rowsperpage = 39 ## ONLY 39 ROWS OF STUDENTS ON FIRST PAGE
        while int_studentnum > 0:
            if int_page == 1: ## CHECKS IF ON FIRST PAGE
                int_lowerbound = int_columnvalue ## STARTS FROM ROW 1
                int_upperbound = int_lowerbound + int_rowsperpage ## ENDS AT ROW 40
            else:
                int_lowerbound = int_columnvalue + 78 + ((int_page-2)*84) ## FORMULA FOR FINDING STARTING ROW FOR PAGE
                int_upperbound = int_lowerbound + int_rowsperpage ##  42 ROWS OF STUDENTS ON EVERY REMAINING PAGE
            for int_i in range(int_lowerbound,int_upperbound): ## LOOPS THROUGH EACH ROW NUMBER ON PAGE
                if int_studentnum > 0:
                    self.obj_cursor.insertText(str(int_i)) ## I IS POSITION OF USER IN LEADERBOARD
                    self.obj_cursor.movePosition(self.obj_cursor.NextCell)
                    str_name = lst_scores[int_i][1]+" "+lst_scores[int_i][2] ## FINDS FULL NAME OF USER
                    self.obj_cursor.insertText(str_name)
                    self.obj_cursor.movePosition(self.obj_cursor.NextCell)
                    self.obj_cursor.insertText(str(lst_scores[int_i][3])) ## INSERTS THE ACTUAL SCORE FOR THE USER
                    self.obj_cursor.movePosition(self.obj_cursor.NextRow)
                    if int_columnvalue == int_rowsperpage+1: ## CHECKS IF AT END OF PAGE
                        self.obj_cursor.movePosition(self.obj_cursor.NextCell)
                        self.obj_cursor.movePosition(self.obj_cursor.NextCell)
                        self.obj_cursor.movePosition(self.obj_cursor.NextCell)
                        self.obj_cursor.movePosition(self.obj_cursor.NextCell)
                    int_studentnum -= 1 ## DECREASES NUMBER OF STUDENTS LEFT TO INSERT
                else:
                    break
            if int_columnvalue == 1: ## CHECKS IF ON FIRST COLUMN
                int_columnvalue += int_rowsperpage ## GOES TO SECOND COLUMN
                for int_i in range(int_rowsperpage): ## LOOPS THROUGH ALL ROWS TO GO TO TOP OF PAGE
                    self.obj_cursor.movePosition(self.obj_cursor.PreviousRow)
                self.obj_cursor.movePosition(self.obj_cursor.PreviousCell)
                self.obj_cursor.movePosition(self.obj_cursor.PreviousCell) ## MOVES TO SECOND COLUMN
            else:
                int_page += 1 ## GOES TO NEXT PAGE
                int_columnvalue = 1 ## RESETS TO FIRST COLUMN
                int_rowsperpage = 42 ## ENSURES NUMBER OF ROWS ON PAGE IS 42
                self.obj_cursor.movePosition(self.obj_cursor.PreviousCell) ## MOVES TO FIRST COLUMN
                self.obj_cursor.movePosition(self.obj_cursor.PreviousCell)
                self.obj_cursor.movePosition(self.obj_cursor.PreviousCell)
                self.obj_cursor.movePosition(self.obj_cursor.PreviousCell)

    def Back(self):
        self.textEdit.clear() ## CLEARS TEXT EDIT
        PrintLeaders_Window.hide() ## HIDES THE PRINT LEADERS WINDOW
        ViewSets_Window.show() ## SHOWS THE VIEW SETS WINDOW

    def printing(self):
        obj_dialog = QtGui.QPrintDialog() ## CREATES THE PRINT DIALOG BOX
        if obj_dialog.exec_() == QtGui.QDialog.Accepted:
            self.textEdit.document().print_(obj_dialog.printer()) ## GETS TEXT FROM TEXT EDIT FOR PRINTING

    def preview(self):
        obj_dialog = QtGui.QPrintPreviewDialog() ## CREATES THE PREVIEW DIALOG BOX
        obj_dialog.paintRequested.connect(self.textEdit.print_) ## GETS PRINT FROM TEXT EDIT TO MAKE A PREVIEW
        obj_dialog.exec_()


app = QtGui.QApplication(sys.argv)

Login_Window = LoginWindow(None)
TeacherRegistration_Window = TeacherRegistration(None)
StudentHome_Window = StudentHome(None)
TeacherHome_Window = TeacherHome(None)
PlayGame_Window = PlayGame(None)
Easy_Window = Easy(None)
Hard_Window = Hard(None)
Results_Window = Results(None)
Leaderboard_Window = Leaderboard(None)
SetAnalysis_Window = SetAnalysis(None)
ViewStudents_Window = ViewStudents(None)
ResetPassword_Window = ResetPassword(None)
StudentRegistration_Window = StudentRegistration(None)
ViewResults_Window = ViewResults(None)
ViewSets_Window = ViewSets(None)
Importing_Window = Importing(None)
AddQuestions_Window = AddQuestions(None)
PrintLeaders_Window = PrintLeaders(None)

Login_Window.show() ## SHOWS THE LOGIN WINDOW
app.exec_()
```

EVALUATION

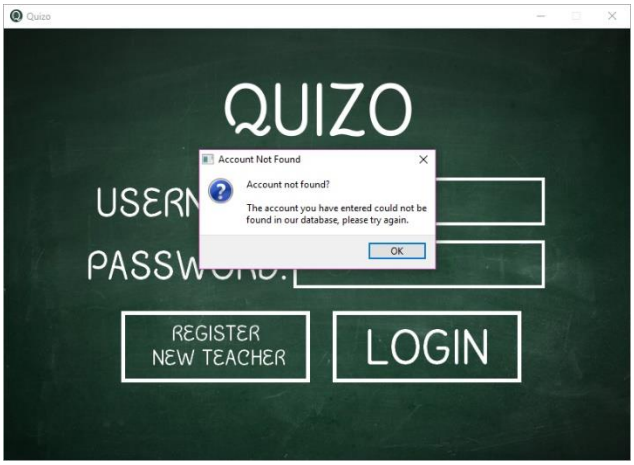
POST-DEVELOPMENT TESTING

Now that I have finished developing the first prototype for my program, I will test all its features in order to check that it is all working as expected. I will be testing each window at a time, starting with the login window.

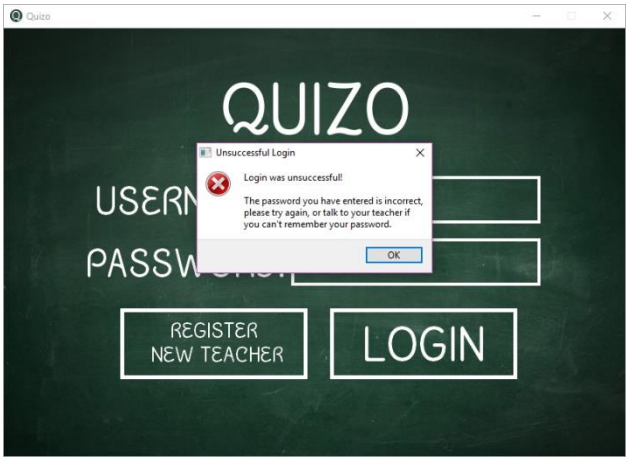
LOGIN WINDOW TESTS



When executing the program, the login screen successfully appears first.

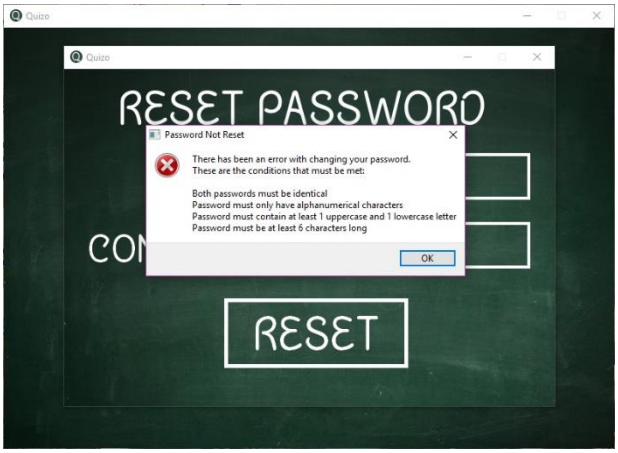


When trying to login with an account that doesn't exist in the database, a prompt appears informing the user that the entered user does not exist.



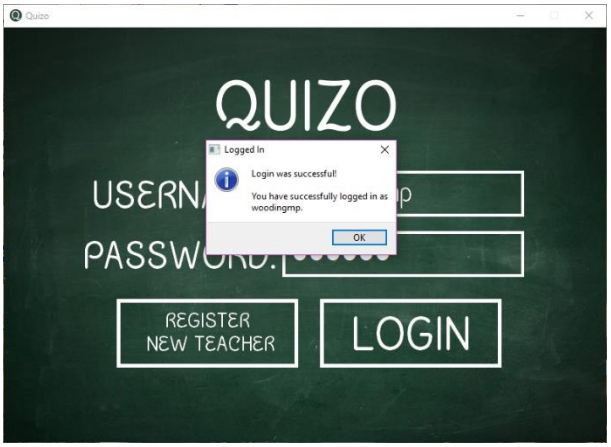
When trying to login to an account that does exist, if the password entered is incorrect, a prompt informing the user that their login attempt was unsuccessful is shown.

When logging in, if the user has had their password reset, and they enter "Quizo123" as their password, they will be shown the reset password screen.



When resetting a password, if the passwords do not match, or they do not meet the password requirements, this prompt is correctly displayed.

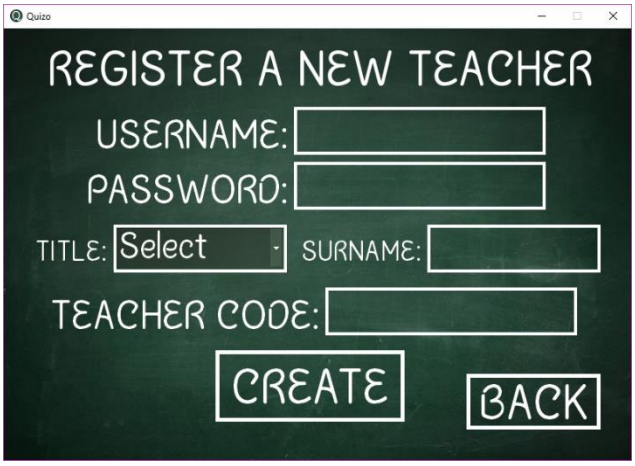
If the password entered is valid, and the user confirms it by entering the same string in both boxes, the password will be reset in the database.



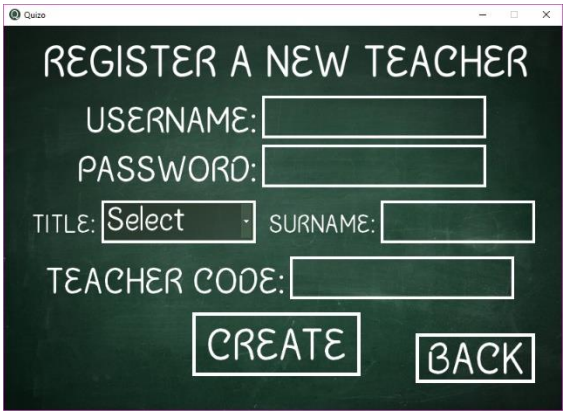
As shown to the left, after resetting a password, to show that it is correctly updated in the database, I tried to login as “woodingmp” again with the new password, and the program successfully informed me that I successfully logged in. This is the same message that is shown when any user logs in successfully.

When the register new teacher button is pressed on the login window, the user is successfully shown the teacher registration window, and the login window is hidden.

All functionality for the login and reset password windows is working as intended, and I will now test the teacher registration window.

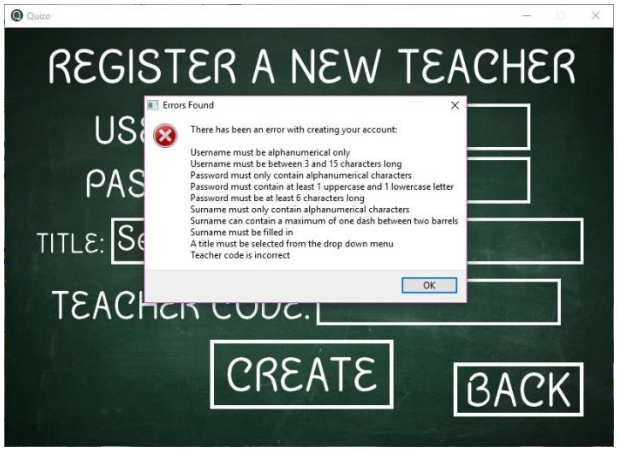


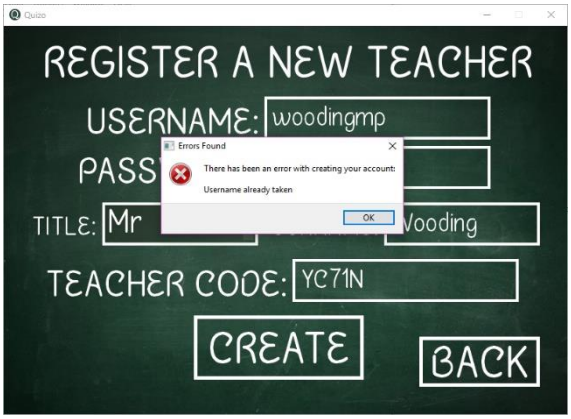
TEACHER REGISTRATION WINDOW TESTS



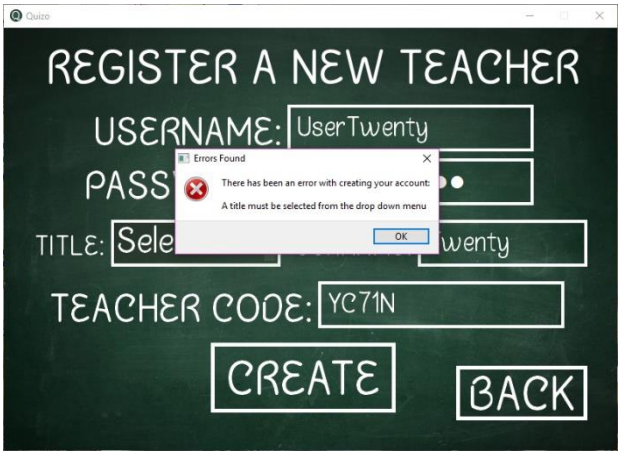
When loading this window, the GUI is created successfully.

When trying to create an account without entering any details, every possible error is shown, except “username taken”.

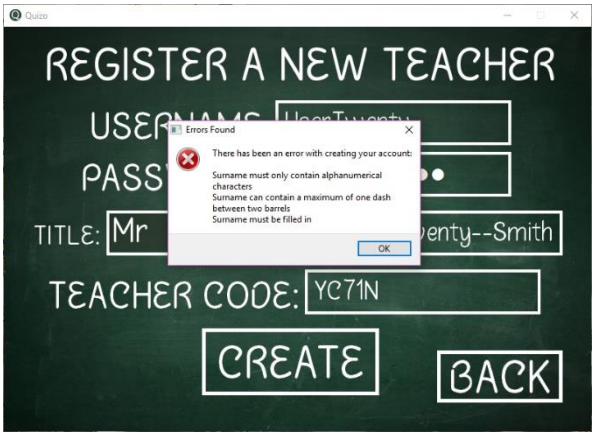




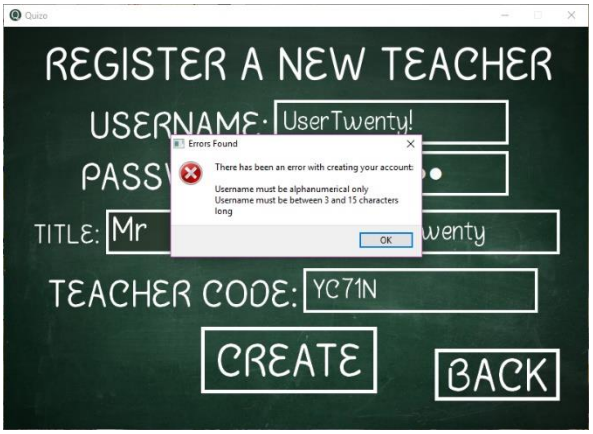
When all details are entered correctly, but the username entered is taken, the error “Username taken” is shown.



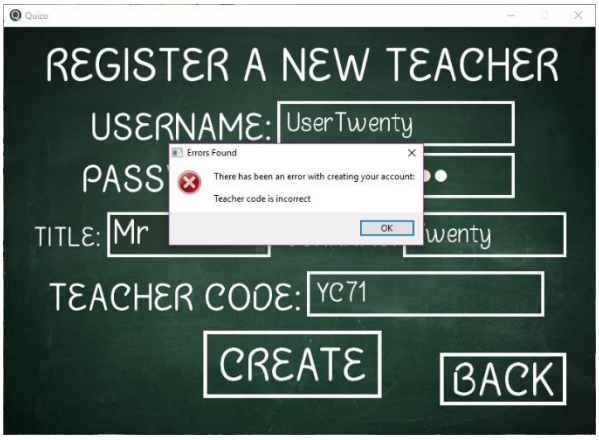
When all details are entered correctly, except the user hasn’t selected a title from the drop-down box, the error “A title must be selected” is shown.



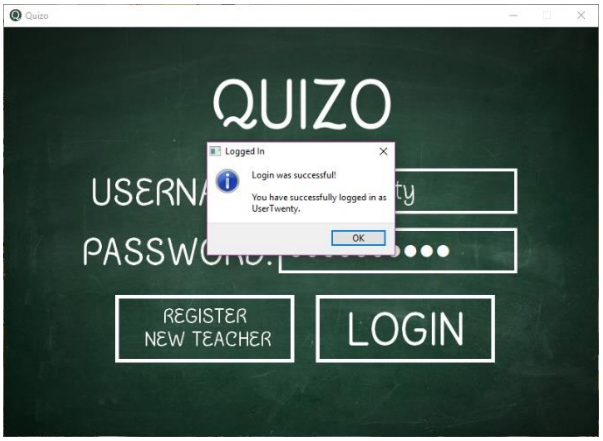
When all details are entered correctly, except that the surname is not valid due to having multiple dashes, null barrels, or invalid characters, an error appears informing the user of the conditions that need to be met for a valid surname.



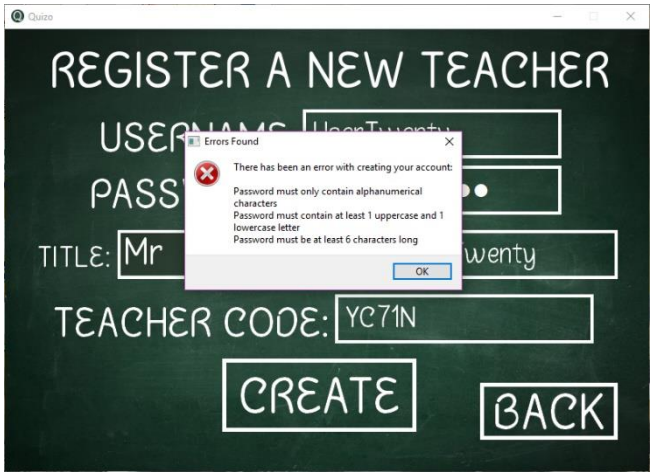
When all details are entered correctly except for an invalid password, an error appears informing the user that their password must only contain alphanumerical characters, with at least one uppercase and one lowercase letter, and must be at least six characters long.



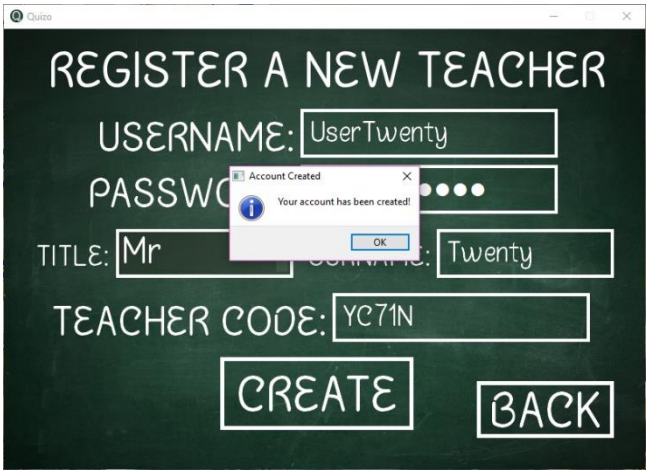
When all entries are valid, the teacher code is correct, and the create button is pressed, the account is created and added to the database, and a message informs the user of this.



When all details are correctly entered except for an invalid username, an error appears informing the user that their username must be between 3 and 15 characters long, and cannot contain non-alphanumerical characters.



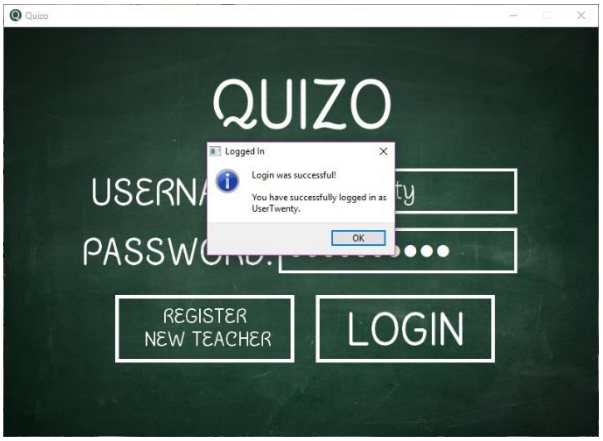
When all details are entered correctly except for an incorrect teacher code, an error appears informing the user that the teacher code is incorrect.



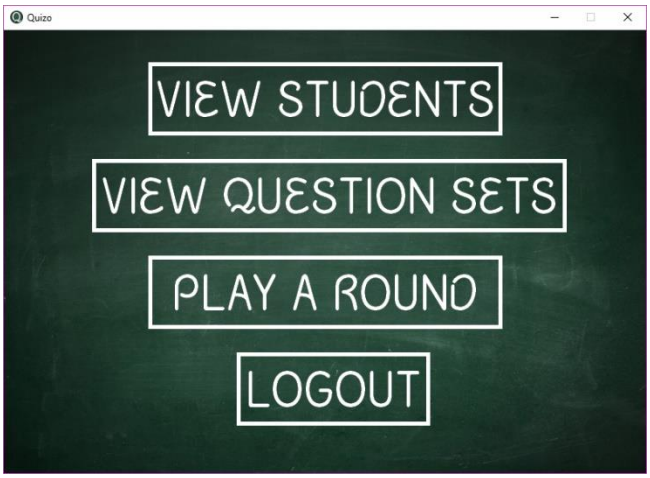
To show that the account is successfully added, I tried to login as this new user, and I was informed that my login was successful.

Now that I have tested the teacher registration window fully, I will move onto the student and teacher home screens.

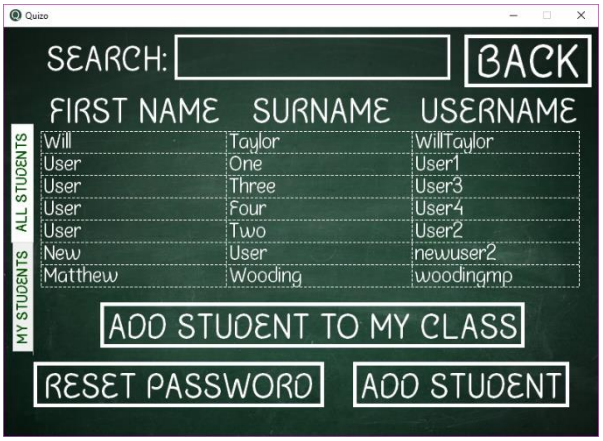
TEACHER HOME TESTS



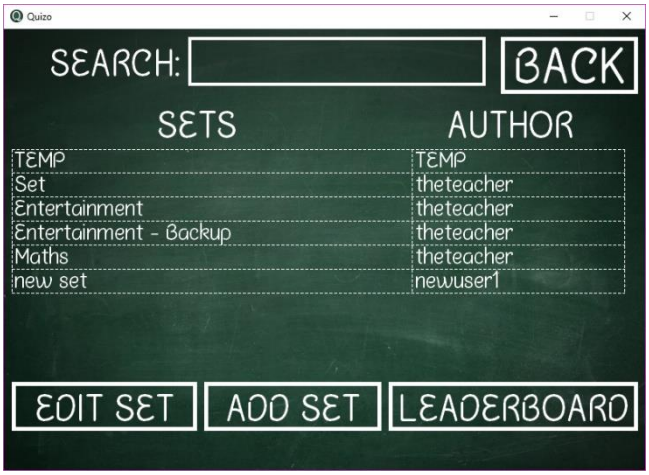
When attempting to login with a teacher account, the teacher home window is successfully shown, and the login window is hidden.



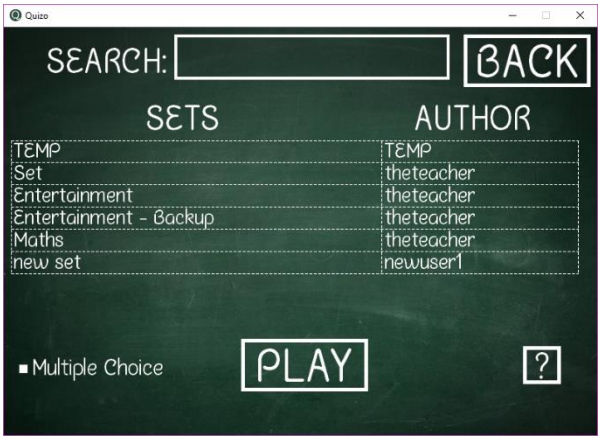
When pressing the view students button, the view students window is successfully shown, and the teacher home window is hidden.



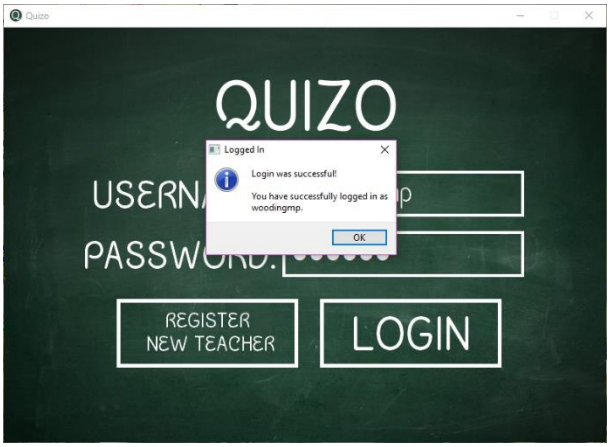
When pressing the view sets button, the view sets window is successfully shown, and the teacher home window is hidden.



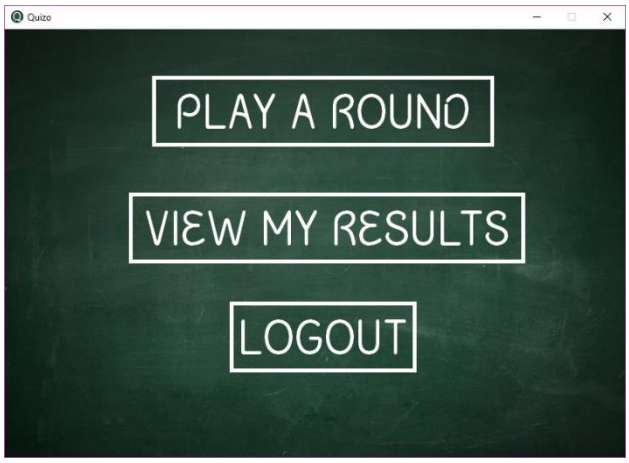
When pressing the play a round button, the play game window is successfully shown, and the teacher home window is hidden.



STUDENT HOME TESTS



When logging in as a student, the student home window is successfully shown, and the login window is hidden.

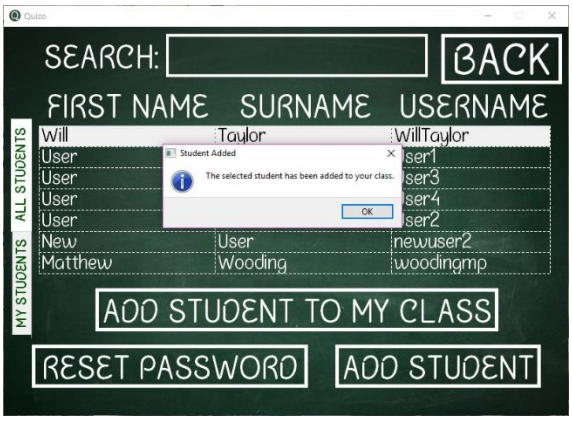


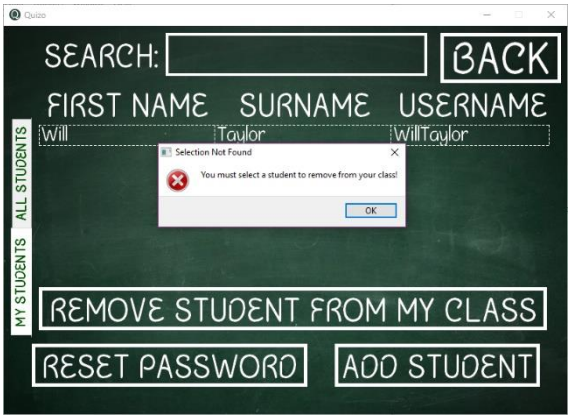
When the play a round button is pressed, the play game window is shown, and the student home window is hidden.

When the view my results button is pressed, the view results window is shown, and the student home window is hidden. The results shown are correct for the current user.

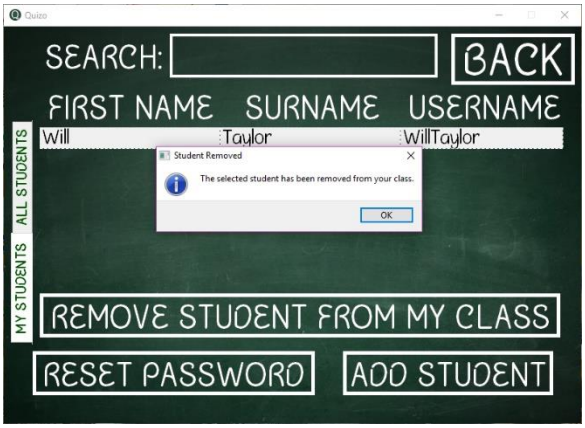
Now that I have tested the home screens, and they are both working as intended, I will move onto testing the view students window.

VIEW STUDENTS WINDOW TESTS

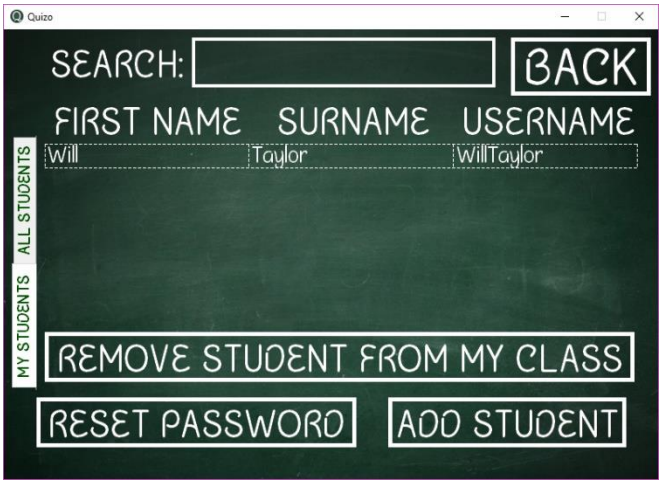


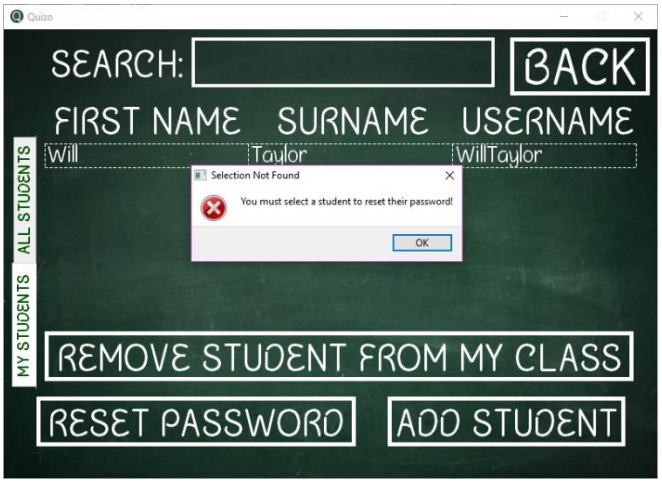


When trying to press any button that requires a student to be selected from the table, such as the remove button, add button, and reset password button, if no student is selected, an error will appear informing the user to select a student first.

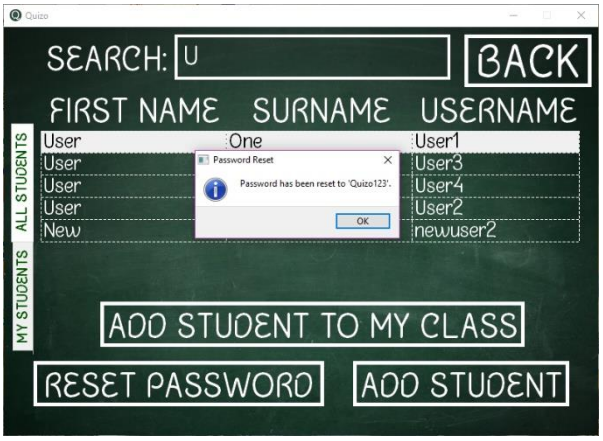


When loading the view students window as a teacher, all current students in the database are successfully shown in a table. When selecting a student and pressing the add student button, the student is successfully added to that teacher's class.



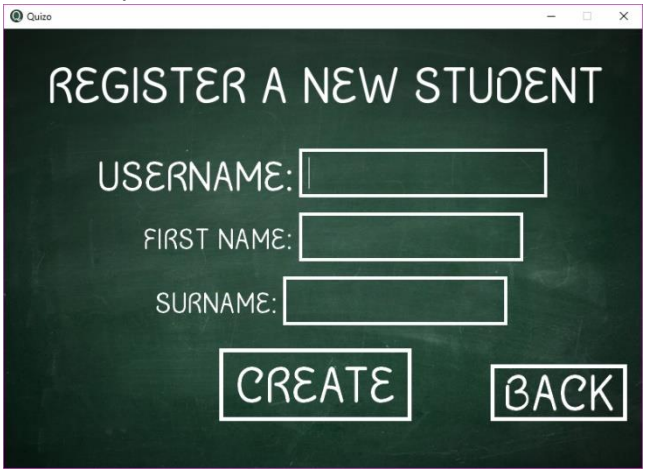


When selecting a student and pressing the remove student button, the student is successfully removed from that the logged in teacher's class.

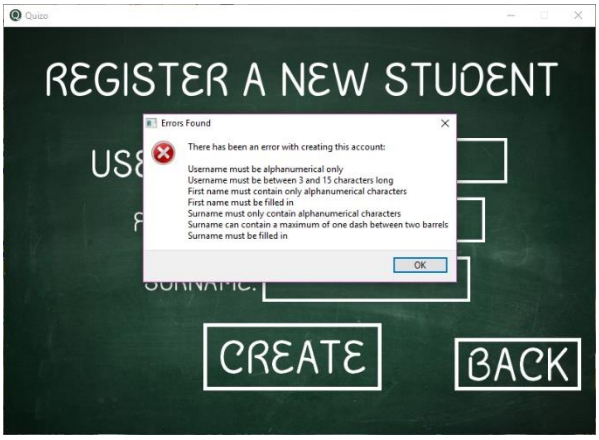


When any student is selected, and the reset password button is pressed, their password is successfully set to the default “Quizo123” and a message is shown to confirm this.
As can also be seen in the screenshot opposite, the search bar works as intended, and has successfully filtered all students to show just those that have a “U” in any column.

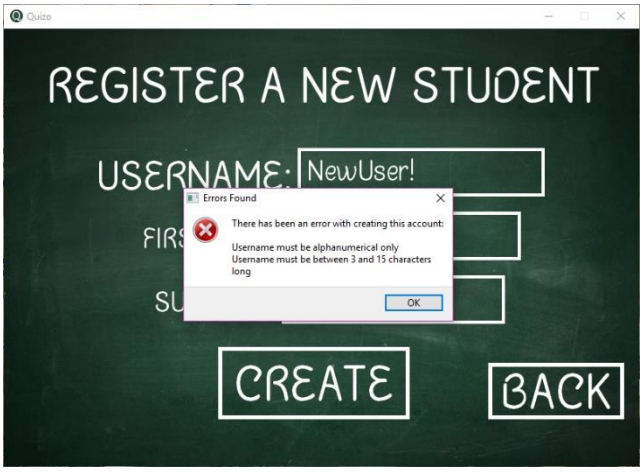
When the add student button is pressed, the student registration window is shown, and the view students window is successfully hidden.



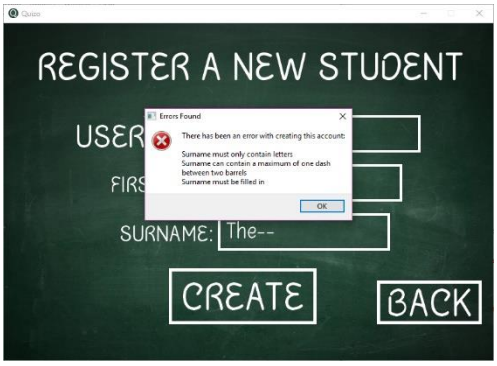
When all entry boxes have invalid or no inputs, and the create button is pressed, a message with all errors is shown.



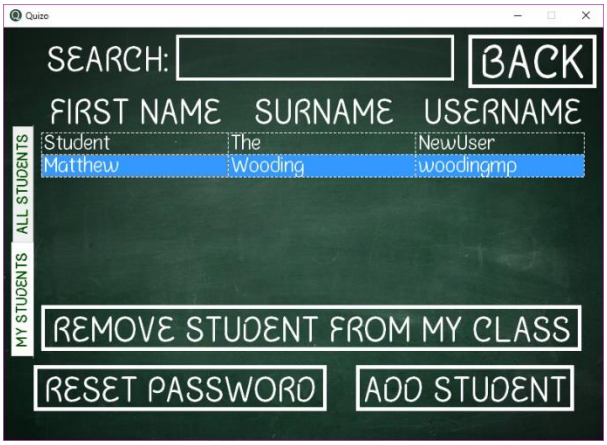
When all entries are valid except for the username, an error message appears informing the user that their username must be between 3 and 15 characters long, and must only contain alphanumerical characters.



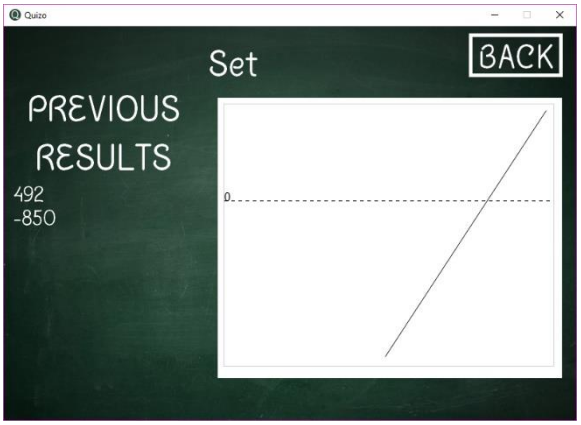
When all entries are valid except for the first name, an error message appears informing the user that first names must be made up of only letters. The same validation is used for the surname field, except there can be a maximum of one dash.



The new account can be seen in the table of this teacher's students.

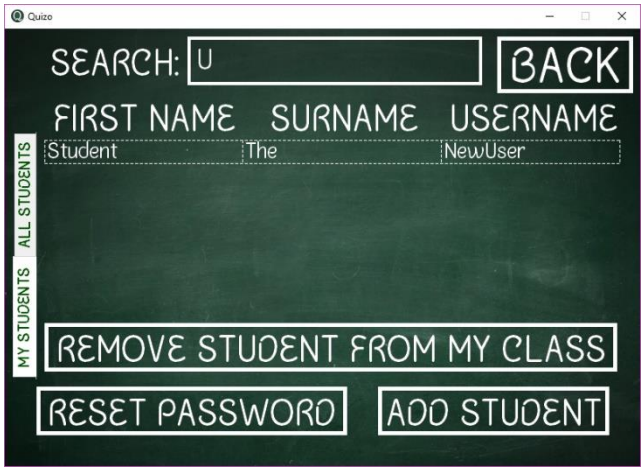


The view results window shows all the sets attempted by the selected student, as well as their highest score in each set.

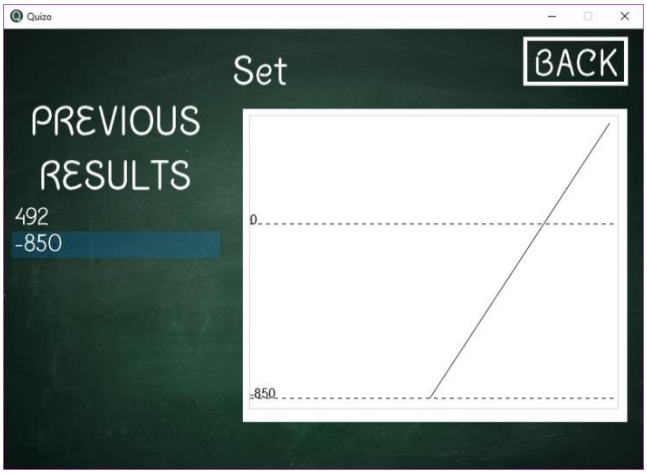


When a set is either double clicked, or selected and then the view more button is pressed, the set analysis window is shown. This window shows a graph of all the attempts made over time, as well as a list of all the scores obtained. When a score is selected, it is drawn onto the graph as shown.

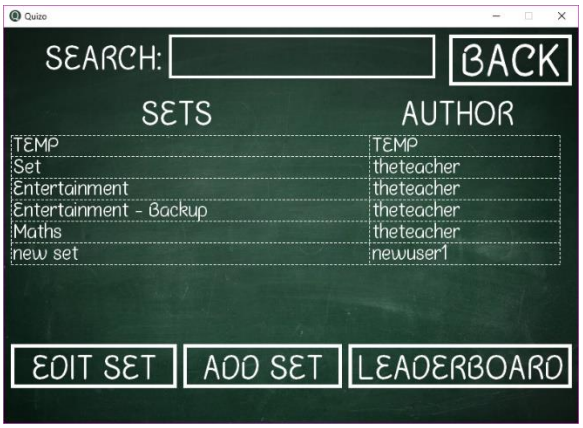
When all entries are valid, and the create button is pressed, the new student is added to the database with the default password "Quizo123".



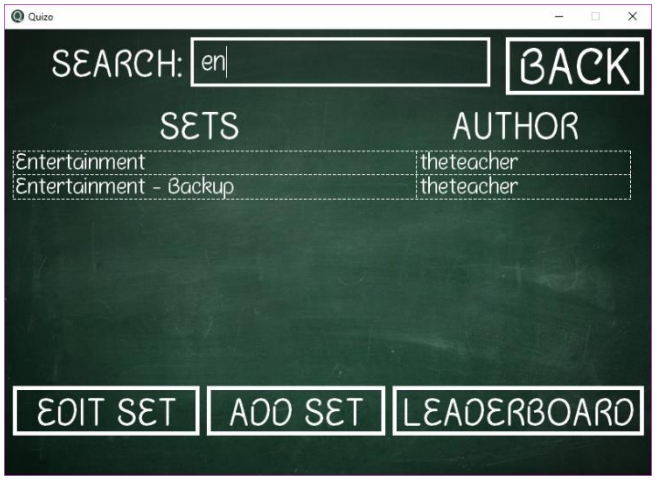
When a student from a teacher's class is double clicked, the view results window is shown as if logged in as the selected student. When the view results window is shown, the view students window is hidden.



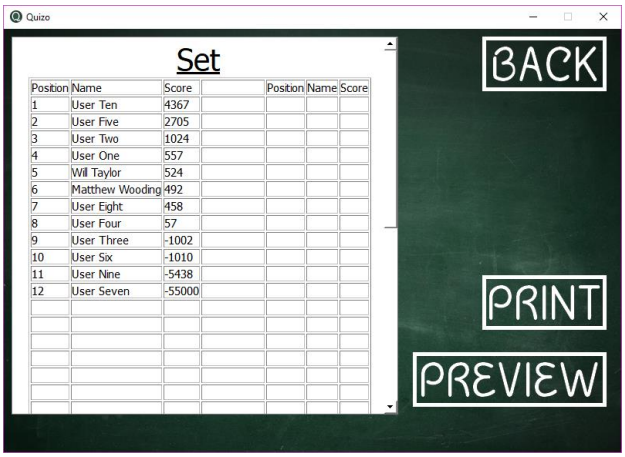
VIEW SETS WINDOW TESTS



When the view sets window is shown, a table of all the sets created is shown, alongside their authors.

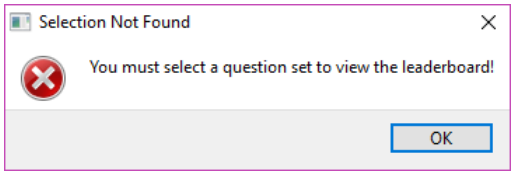


When typing into the search bar, the table correctly filters out sets that do not contain the entered text in either their set name or author.

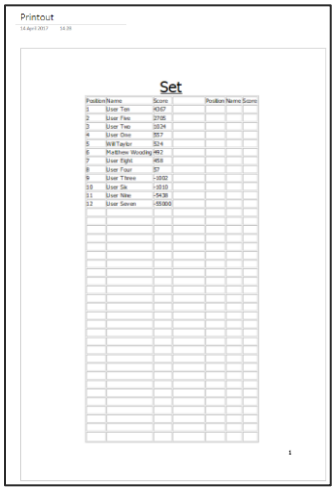


Position	Name	Score		Position	Name	Score
1	User Ten	4367				
2	User Five	2705				
3	User Two	1024				
4	User One	557				
5	Will Taylor	524				
6	Matthew Wooding	492				
7	User Eight	458				
8	User Four	57				
9	User Three	-1002				
10	User Six	-1010				
11	User Nine	-5438				
12	User Seven	-55000				

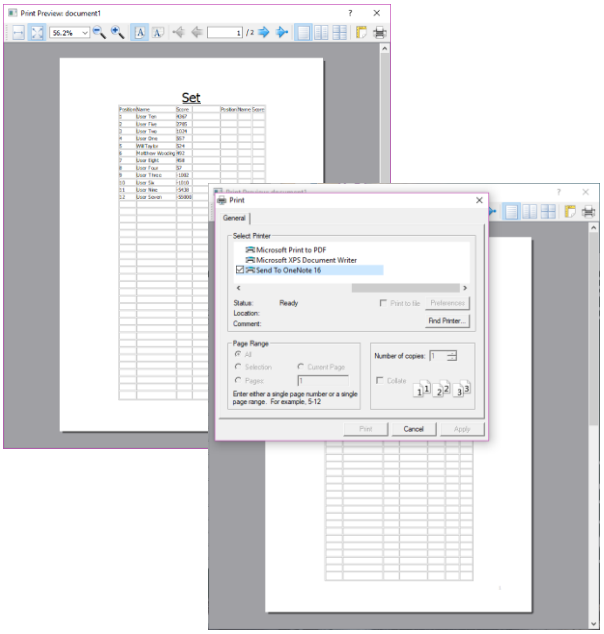
Before viewing the leaderboard for the "Set" quiz, I added some new users to the program, and logged in as them to add some more results to the database, in order to showcase the leaderboards.
When a set is selected and the leaderboard button is pressed, a leaderboard showing every user that has attempted the set is shown. If a set is not selected and the leaderboard button is pressed, an error appears informing the user to select a set.



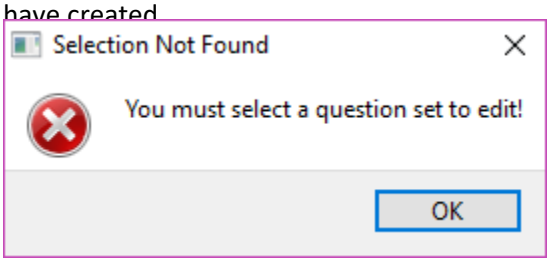
When the preview button is pressed, a print preview is shown, displaying the full leaderboard. From here the user can go to the print window, which is the same as if the print button was pressed from the initial screen.

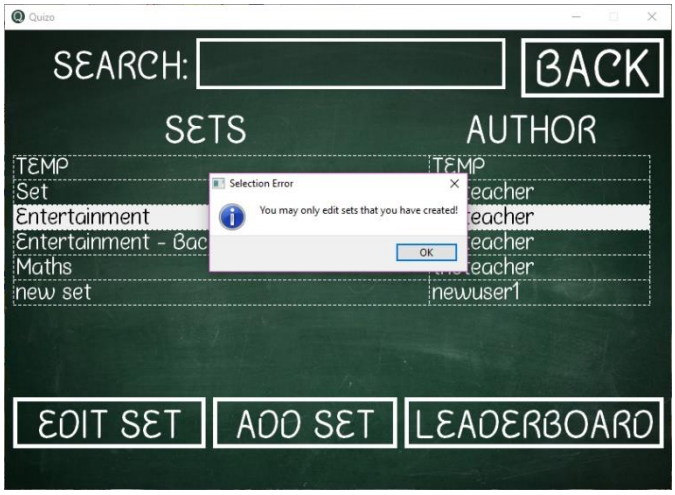


When the print button is pressed, the program can print to the desired printer, and to test, I printed to the "Send to OneNote 16" printer which sends it to a OneNote page, and the image printed can be seen to the side.

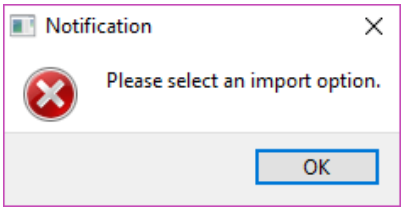


When trying to edit a set, if no set has been selected, an error will appear. If the set selected is one that the current user has not created, a different error will appear to inform the user that they can only edit sets that they have created.

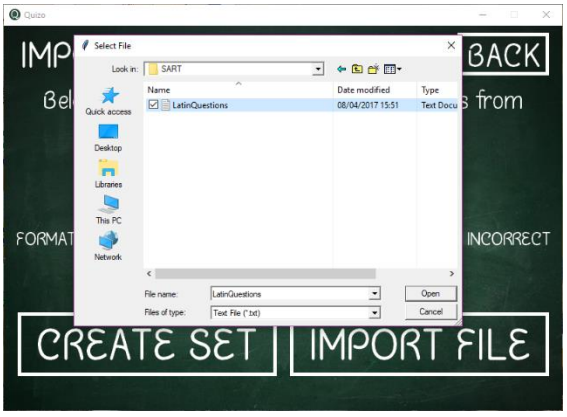




When the add set button is pressed, the importing window is shown successfully.

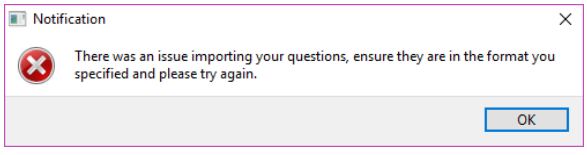


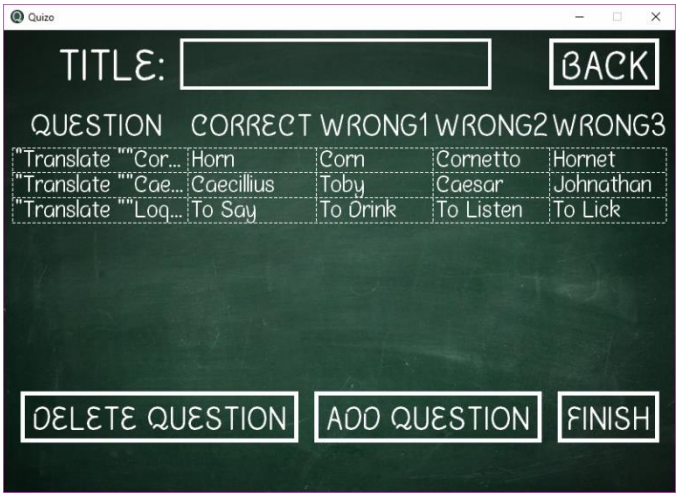
If the import file button is pressed without selecting an import option, the above error message will appear.

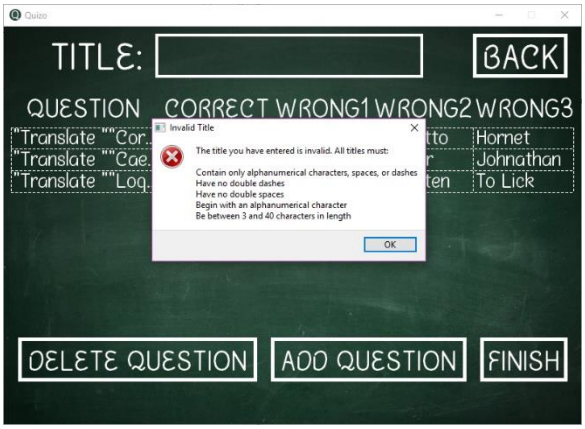


If an option is selected, a file browser will appear to allow the user to select a file from their computer. Only text files or csv files can be displayed in the file browser. When a file is selected, the program tries to import its contents into the question table.

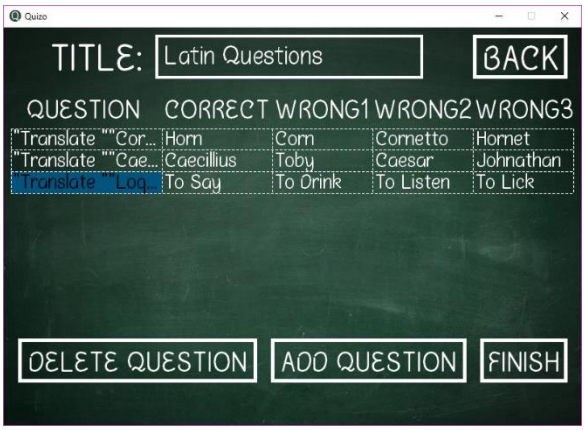
If the import is successful, the questions will be displayed as shown to the right. However, if the file is in an incorrect format and cannot be imported into the table, the below message will appear.



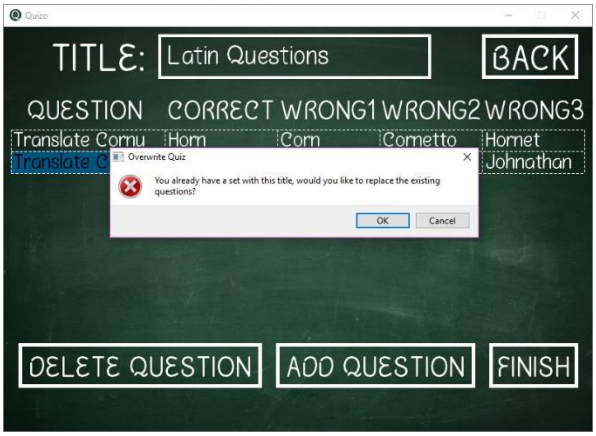




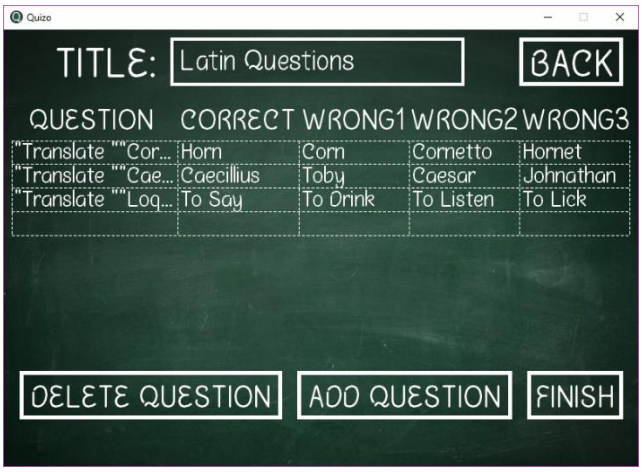
If the add question button is pressed, a new empty row is added to the bottom of the table.



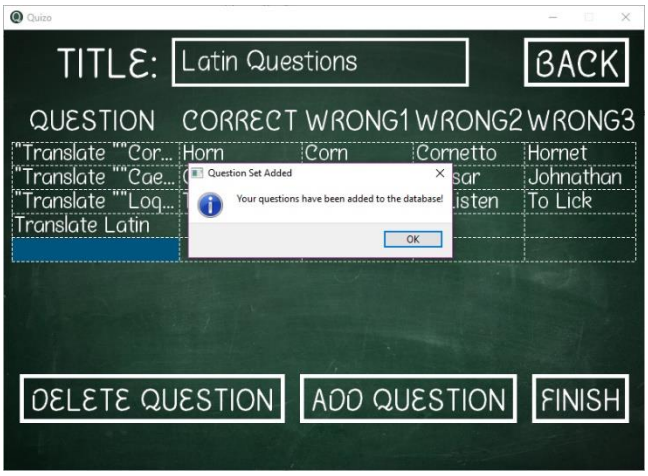
If the title is valid, and the finish button is pressed, as long as the current user has not created a set with this name before, the set and its questions will be added to the database. Any questions that are not fully entered and have fields missing will not be added in any capacity to the database.



If the finish button is pressed without a valid title, an error appears, explaining what conditions a valid title must meet.



If a question is selected, and the delete question button is pressed, the selected row is removed from the table. If no question is selected and the button is pressed, an error message will appear.



When trying to create a question set with the same name as one already created by the current user, a message will appear asking whether they would like to overwrite their previous set or not. If so, the program will delete the old set, and write the new set to the database.

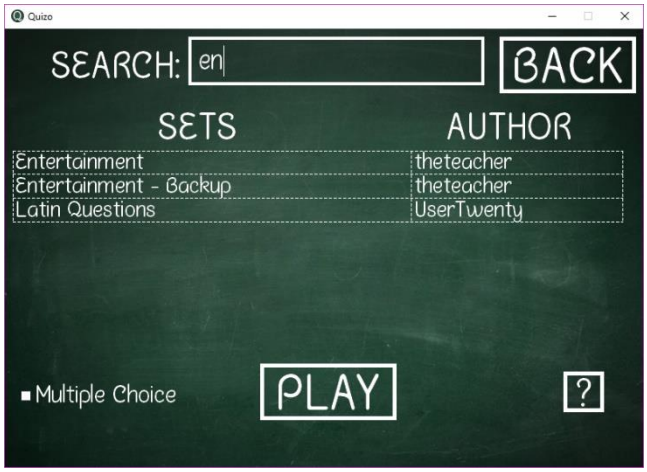
PLAY GAME WINDOW TESTS

As can be seen to the right, when entering text into the search bar, the table is successfully filtered to only show sets that contain the entered text in their name or in their author's name.

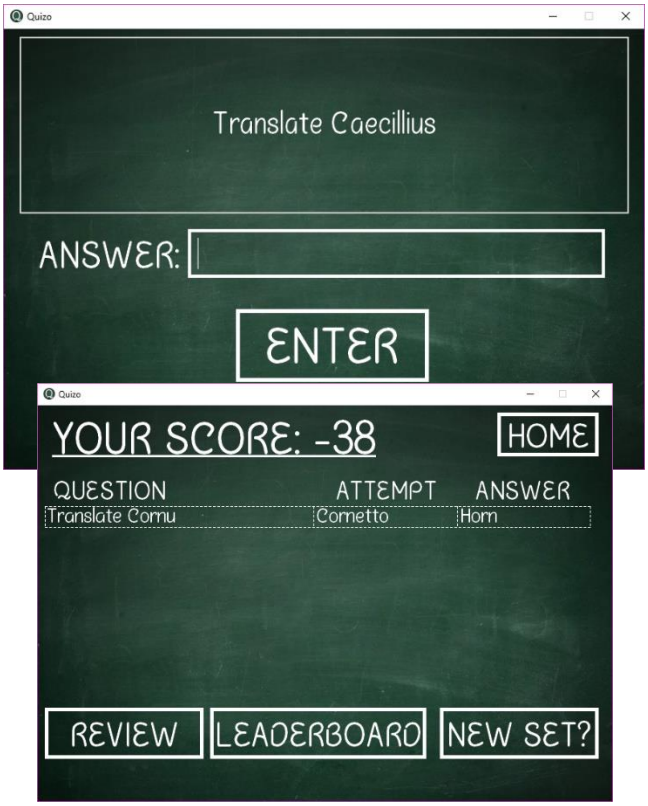
If the multiple-choice box is not checked, the hard mode window is successfully shown, and the first randomly selected question from the set is shown. If the enter button is pressed, or if the user presses the enter key on their keyboard, the question will end and then next question will be displayed.

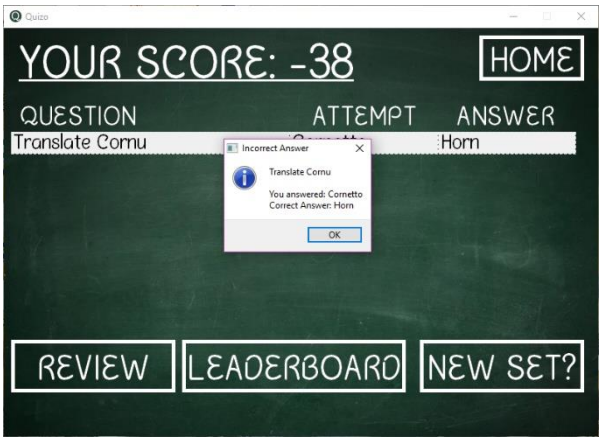
When the user answers their final question, the results window is shown, and the hard mode window is hidden. When the results window is shown, all the questions that were answered incorrectly are displayed in a table, and the score achieved by the user is displayed at the top.

When the play a round button is pressed by either a teacher or student, the play game window appears. When it appears, the window successfully displays all the existing sets in the program that can be attempted by users.



When the play button is pressed, if a set has been selected, the program will go to the selected mode's window. If no set is selected when the button is pressed, an error message will appear to inform the user to select a set. The quiz will also start if a set is double clicked.



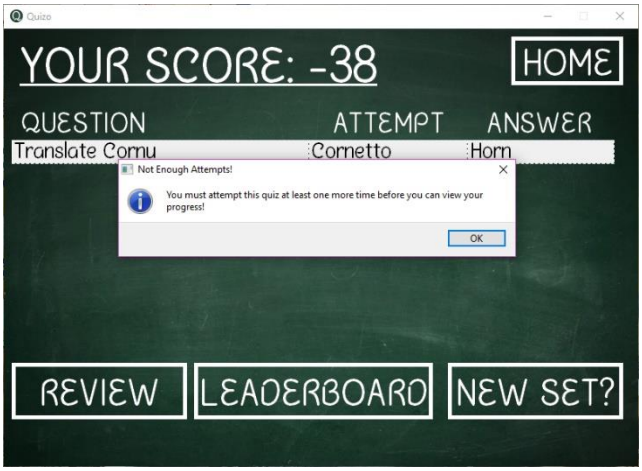


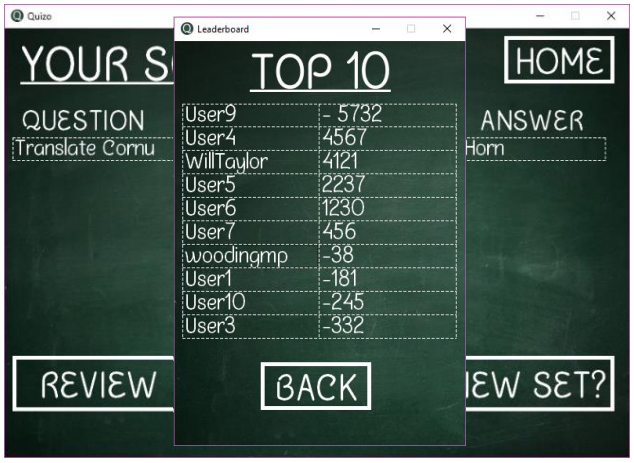
If the review button is pressed, if the user has only attempted this set once, an error will appear informing them that they need to attempt the set more times to view their progress.

If the leaderboard button is pressed, the users with the ten best scores in the set will be displayed in the leaderboard window that is shown. If the back button is pressed, the leaderboard window is successfully hidden.

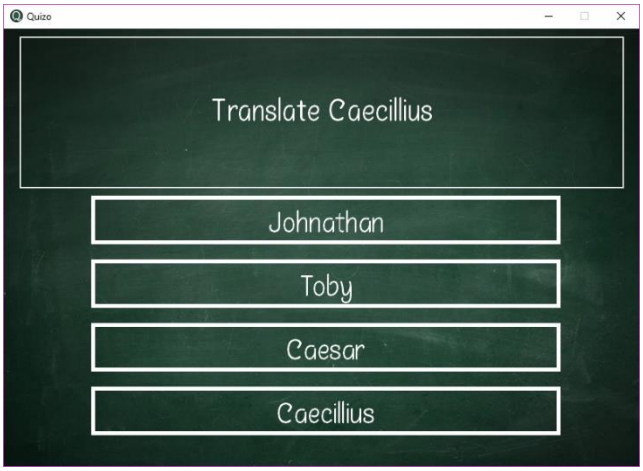
When one of the answer buttons is pressed, the quiz will move onto the next question until there are no more questions left, and the results screen will be shown.

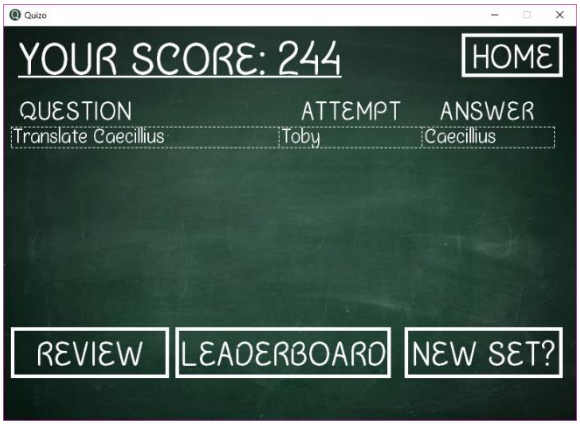
When a question is double clicked on the table, a pop up box will appear, displaying all the details.



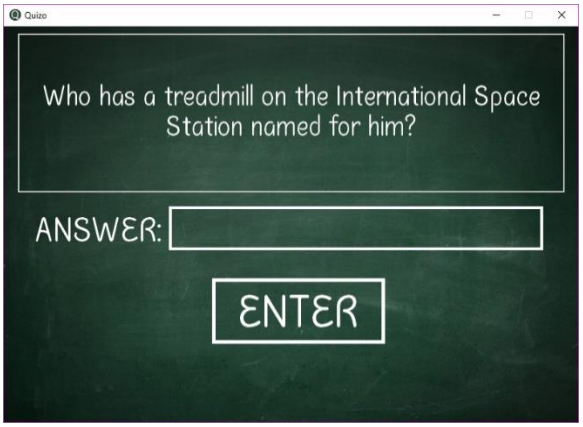


If the multiple-choice box is checked when the user tries to start a quiz, the easy mode window will be shown, and similarly, the first randomly selected question will be displayed, along with all the possible answers.

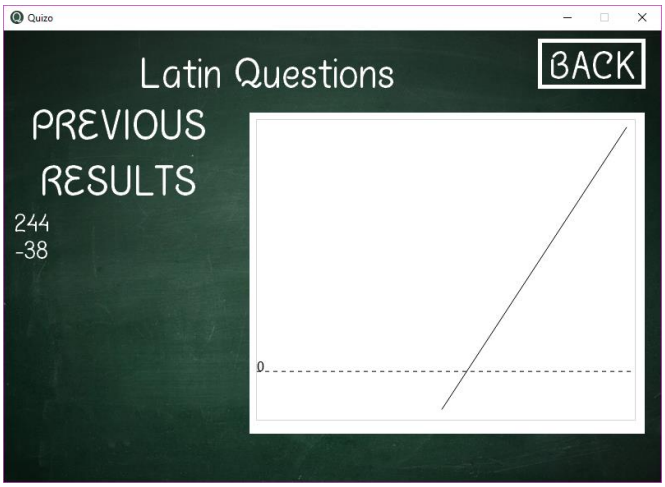




When the user has more than one attempt in a set, when the review button is pressed, the set analysis window is shown.



When all the questions are answered, the results window is successfully shown, and the easy mode window is hidden.



When on the play game window, if the question mark button is pressed, the program will randomly select a quiz and start it for the user.

Now that I have finished testing the play game and surrounding windows, I have tested all parts of my program and have encountered no errors in doing so. From this I can safely say that the program is working as intended and is ready to be sent to my stakeholders for review.

TARGET AUDIENCE TESTING

With the program's first prototype complete, I returned to the RGS and gave the prototype to Mr Hamm, and he told me to let him, his students and colleagues try out the program for a few days before he came to any judgement about whether he would be planning on using the program in the future. Upon my return to the school, I managed to speak to a few of the students and members of staff to hear their thoughts of the program, and what sort of changes they would like made in future versions. Below are the responses I received, followed by my interview with Mr Hamm.

"I managed to create a quiz about the cold war for my history class this morning and we went to a computer room to let them all try it. The boys seemed thoroughly engaged with the program, and many of them continued to attempt the quiz just to try and get the highest score in the class. I ended up showing the leader board for the quiz on the projector so they could all see their progress and not just the top ten. I feel the boys would agree in saying that perhaps having a way for students to view the same leader board as teachers would be helpful." – Mr Johnson, History teacher at RGS

"When I logged into the program at lunch yesterday I could already see several quizzes I was interested in such as a chemistry quiz based on the periodic table, and I decided to just try them all! The program

was intuitive and it looked really interesting with the sleek font and background as opposed to a dull and boring design." – James Taylor 12EB, studying, Chemistry, Physics, Maths, and French

"I was pleased to see the random quiz button that I suggested last time you were here and I used it a few times because I didn't know which quiz to try first. I also thought the graph was useful for seeing how I was progressing after each attempt I made, however maybe it could have looked a bit more colourful like the rest of the program rather than just black and white. – Seb Perry 12CD, Studying Maths, Physics, and Biology

"I tried adding some maths questions to the program that I had created a few years ago for a worksheet, and found the importing tool really helpful since I had already created the questions in an excel spreadsheet; the instructions were really clear and the questions imported exactly as I had hoped." – Mrs Dean, Maths teacher at RGS

"Although before I said having a pause button would be a good idea, I found myself not needing it at all when attempting the quizzes because I ended up just doing lots of attempts at a time. Although I didn't end up using it, the search bar seemed like a really good idea, especially when even more question sets get added to the program and I want to find ones for specific topics I need help with." – Finlay Franks 13PD, Studying Maths, Further Maths, and Physics

"After I set my class a homework in the program to do a few quizzes for me, I printed out the leaderboards for them this morning so they could see how they were fairing against their peers. Although the leaderboards themselves looked good, they included students that aren't in my class, so if there was a way to filter students by class for leaderboards that would be a useful feature for me and my class." – Mr Davis, German Teacher at RGS

INTERVIEW WITH MR HAMM – HEAD OF COMPUTING

Q: I've already gotten the chance to speak to a few of your colleagues and students about the program, but from your own experience, how well is it working with your teaching?

A: Well we've been using it for a few days now and I've tried some quizzes with a number of my students and so far, they've all found the program to be a fun exercise in the classroom, and some of them came in this morning asking if we could do some more quizzes today, so in that respect, it appears as though the students are really taking a shine to your program.

Q: Have you found the program easy to use, or are there some things that you wish were included to make the experience easier?

A: I've been able to make a few quizzes with ease, and being able to track who has been attempting them and who is doing well is a useful feature. Although I personally haven't needed to import a quiz from a file, I've spoken to some of the Maths department and they assured me that it was working well for their questions. The menus are all very clear and bold, and I don't feel like there has been any issue with being confused or not knowing what to do, because as I said, the program is very clear in how to navigate the windows and how to make and attempt quizzes. In terms of making things easier, perhaps having a way of filtering students in more ways than just who I do and don't teach would be useful;

many teachers at the school are teaching students from six different year groups, so being able to split them accordingly would make things easier when looking at leaderboards.

Q: Are there any core features that you feel are missing from the program, or is it just tweaks to existing features that would improve the program?

A: In terms of core functionality I feel like it's all there; we both know that this was never meant to be a substitute for teaching or exams, and that it was meant to be used as a revision tool. As I said, more filtering options would be useful, and perhaps a way of notifying students when their teachers have created a new quiz could help, perhaps by email or just with a notification in the program when they login.

Q: Would you say that a fully functioning messaging system built into the program would be a good idea for this then?

A: I don't think it needs to go that far in my honest opinion, students just need to be able to know when there is more work to do, so going back to filters, maybe being able to filter question sets by when they were created, and who created them could be helpful.

Q: Are you happy with the aesthetic and feel of the program that I have designed, or are there some tweaks you would like to see in a future version?

A: In terms of the aesthetic, I am far beyond happy with how the program looks; it's bright and colourful, and I think that's been a huge part in trying to get the students to use it. Moving forward however, being able to resize the windows so that we can scale the program up on projectors would be useful.

Q: Given that you've had some time to get a feel for the program, would you be happy to continue working together and create a second iteration of the program for your school to use?

A: I think that given the student's reaction to the program, and the changes we've just talked about, I would be happy to implement the next version of the program into our school's system. After a soft launch on our school system, I could very well see that after making the program have network capabilities and ports to different operating systems, this could become a great revision tool for our students.

CUSTOMER FEEDBACK

Overall I would say that the feedback I received from the RGS was very positive; they found the program to be very useful and the students seemed to be really interested in using it, which is of course the main goal. Although the feedback was mostly positive, there were a few changes that I discussed with the school and Mr Hamm that will need to be considered before the program will have its official launch:

- Allow students to view leaderboards
- Graphs don't fit in with aesthetic for the rest of the program (needs colour)
- More options for filtering and searching (by class, form, year group, teacher, etc.)
- Notify students when they have new quizzes to attempt (email or built in)
- Enable resizing of windows (requires dynamic scaling interfaces)
- Network capabilities – Either a web program, or store the database in the cloud and have a client program
- Ports to different operating systems – Windows, Mac, Linux, iOS, Android

ADDITIONAL CUSTOMER TESTING

Before leaving the RGS again, I asked a few of the students if they could help me with a few tests to see how the usability of my program stacks up quantitively as opposed to similar products. By this I mean timing how long it takes for users to do similar tasks in my program, and in similar programs.

For these tests, I will be comparing my program to *Sporcle* and *Showbie*. The tests that I have come up with can be seen below, alongside the results. To make the tests fair, I will not be the person being timed. Instead, I will be timing RGS students, and the same student will be taking the same test for each application. Additionally, I will be timing how long it takes to go from the home screen (home page for the web applications) to the results I am looking for.

Test	Quizo	Sporcle	Showbie
Creating a student account	21.54 seconds	45.65 seconds	46.38 seconds
Creating a quiz (10 questions)	143.46 seconds	166.08 seconds	192.55 seconds (time taken to create a quiz file and upload to a class)
Viewing progress in a quiz	14.76 seconds	8.05 seconds	N/A
Viewing a leaderboard	12.98 seconds	7.59 seconds	N/A

As shown in the table above, Quizo was the fastest at creating new student accounts, and creating a 10-question quiz when compared to *Sporcle* and *Showbie*. Although *Showbie* doesn't have built-in functionality for leaderboards and viewing progress in quizzes, *Sporcle* was still faster than Quizo in these regards due to being able to view everything on the same screen for a quiz and only having to navigate through two windows to reach these.

From these tests, I can see that while my program may not be the fastest to do everything, it manages to come fairly close in terms of time taken to reach features when it's not the leader in a field. In future

iterations of the program, I could consider increasing the window size and making more space to be able to show multiple pieces of important data in the same window, at which point the time would be more in line with the results we obtained from *Sporcle*.

REVIEWING THE DESIGN

Although in my head, my program has had a very clear vision since its inception, since my design section it has evolved for what I think is the better. For the most part, the layout for the program has remained identical except for a few extra buttons, and the extra window for printing leaderboards.

When creating the extra window for printing leaderboards, having not created an initial design for it, it perhaps wasn't as well thought out in terms of layout as the other windows as it has a fair amount of unused space. In addition to this, the layout for the actual leaderboard was a bit last minute and doesn't really fit the aesthetic for the rest of the program given its grayscale colours. Perhaps for a future iteration of the program, making the leaderboard, as well as the graphs, fit in better with the aesthetic and feel of the program by adding colour and using the same fonts would be a welcome change to Quizo.

Another design choice I made for the program was to use PyQt's built-in message boxes for showing the user errors and information. While this was an easy and fast way to display these messages, these too didn't quite fit the aesthetic of the program, and so for a future iteration of the program, creating message boxes specifically for my program seems like a good idea.

As I mentioned in the previous section, when looking at applications like *Sporcle*, the size of the windows I created for my program are quite small (800x550 pixels), meaning that I had less space to show more data all in one window rather than having to spread data into multiple windows, such as having the graph and top ten leaderboard in separate windows to the results window. Going forward, when working on the dynamically scaling interfaces, accommodating this kind of design would help the program display more data to users in a helpful way.

Despite the issues I have mentioned above, I am more than happy with the way the design for the program turned out, and I think this is cemented by the responses that I received from the members of the RGS when I took the prototype to them.

OBJECTIVE 4 IN PROGRESS – QUIZO CURRENTLY HAS AN EASY-TO-USE AND INTUITIVE DEISGN THROUGHOUT ACCORDING TO MY STAKEHOLDERS, HOWEVER CHANGES COULD BE MADE TO IMPROVE THE EXPERIENCE

SUCCESS CRITERIA REVIEW

In my analysis section, I outlined the 73 objectives that I decided my program would need to reach for me to accommodate the needs put forward by Mr Hamm and the students I interviewed at the RGS. I then built on these objectives in my design section, and created a success criteria table, to be able to accurately check my progress as I was developing my program. Now that I have finished the first prototype for the program, I have gone back through my success criteria and objectives list to check that I met all the goals I set out to achieve. The results of my review can be seen below.

Objective Number	What is being tested	Successful? (Y/N)	Proof (page no.)
1	Load PyQt UI files from a folder	Y	80
2	Load SQLite database from a file	Y	90
3	Generate GUI from PyQt files	Y	80
4	Have an easy-to-use and intuitive design throughout	N	266
5	Load graphic resources from a file	Y	83
6	Import necessary modules into python (e.g. time, hashlib)	Y	231
7	Use object oriented programming to create a class for each window in the program	Y	231
8	Swap between windows seamlessly when a function is called to move between them	Y	114
9	Allow user to input data into text boxes	Y	87
10	Recognize double clicking a row of a table	Y	156
11	Working search bars to search through tables	Y	129
12	Validation for entry boxes throughout	Y	232
12.1	Username Validation	Y	232
12.2	Password Validation	Y	232
12.3	Search Bar Validation	Y	232
12.4	Question Input Validation	Y	232
12.5	First Name Validation	Y	232
12.6	Surname Validation	Y	232
13	When inputting a password, only display bullet points for security reasons	Y	84
14	Fetch usernames and passwords from the database when attempting to login	Y	89
15	Hash passwords for security reasons	Y	85
16	Be able to differentiate between a student and teacher account, moving the user to the according home screen	Y	89
17	Check if the user's password has been reset and if so, allow them to change it	Y	189
18	When changing password, check that the new password is confirmed before editing the database records	Y	189
19	Provide an output when an account hasn't been found	Y	92
20	Provide an output when an entered password is incorrect	Y	92
21	Create a button to allow new teachers to be registered	Y	111

22	When registering a new teacher ensure validation is working correctly	Y	104
23	Provide an output when the account has been successfully created	Y	111
24	Add new teacher accounts to the database	Y	111
25	Allow access to viewing all student accounts (Teachers Only)	Y	193
26	Allow access to viewing all question sets (Teachers Only)	Y	202
27	Allow access to playing a round	Y	124
28	Allow access to view own results (Students Only)	Y	198
29	Allow user to logout	Y	118
30	Fetch all students' data from the database	Y	181
31	Generate a table of all students, displaying name and username	Y	181
32	Be able to filter by students that the teacher teaches	Y	181
33	Allow teachers to add/remove students to/from their class	Y	184
34	Reset a student's password to a default one	Y	186
35	Ability to add new students to the database so long as a unique username is chosen	Y	193
36	Show results of any students from a teacher's class	Y	198
37	Display a student's highest score in each of their attempted tests	Y	198
38	Allow access to in-depth analysis of progress in a given test	Y	164
39	Show a list of all scores ever received in a set, ordered by date	Y	164
40	Generate a line graph showing scores over time	Y	174
41	Scale the graph depending on how many results the student has and the range of the scores	Y	174
42	Show a specific selected score on the graph	Y	178
43	Fetch all question sets and their authors, and display in a table	Y	202
44	Allow access to adding new sets	Y	213
45	Have a browser for selecting a file from storage	Y	208
46	Import contents of file into database	Y	222
47	Allow access to editing sets	Y	213
48	Allow user to upload file with updated/edited questions	Y	222
49	View a leaderboard of everyone who's attempted a quiz	Y	230
50	Allow leaderboard to be printed	Y	230
51	Fetch all question sets and their authors	Y	126
52	Create a table displaying all sets and authors	Y	126
53	Create a checkbox for whether a user wants to play multiple choice	Y	152
54	Allow playing a selected quiz by pressing 'Play'	Y	152
55	Fetch all questions from database for the given set	Y	135

56	Randomly select a question from the pool and then remove it from the pool	Y	138
57	Display the question in a text box	Y	138
58.1	In hard mode, generate an input box	Y	147
58.2	In hard mode, allow entry of answer by pressing enter or pressing 'Play'	Y	147
59.1	In easy mode, shuffle the 4 possible answers	Y	141
59.2	In easy mode allocate each answer to a button	Y	141
59.3	In easy mode, question moves on when a button is pressed	Y	141
60	When a question starts, start a timer	Y	137
61	When a question is answered, stop the timer	Y	137
62	Generate a number of points based on elapsed time (kn where k is a constant and n is elapsed time)	Y	137
63	If answer is correct, add score to total	Y	143
64	If answer is incorrect, deduct score from total	Y	143
65	When there are no more questions left, go to the results screen	Y	155
66	Display total score on the screen	Y	155
67	Display all incorrectly answered questions in a table along with the user's attempt and the correct answer	Y	155
68	If not all text fits in a row, allow user to double click to view in a separate box	Y	156
69	Fetch the top 10 unique scores for the quiz	Y	161
70	Display scores on a leaderboard	Y	161
71	Allow user to review their progress of the set if they've attempted it more than once	Y	175
72	Create a button to let the user return to the play screen	Y	163
73	Writing scores to the database	Y	158

As can be seen in the table above, I have successfully met all the success criteria that I created in my design section, except for objective four, which was to have an easy to use and intuitive interface. Although some of the students and teachers praised the design and accessibility of the program, there were still things that could be changed, and below I will outline the things that could be adapted had I more time and skill.

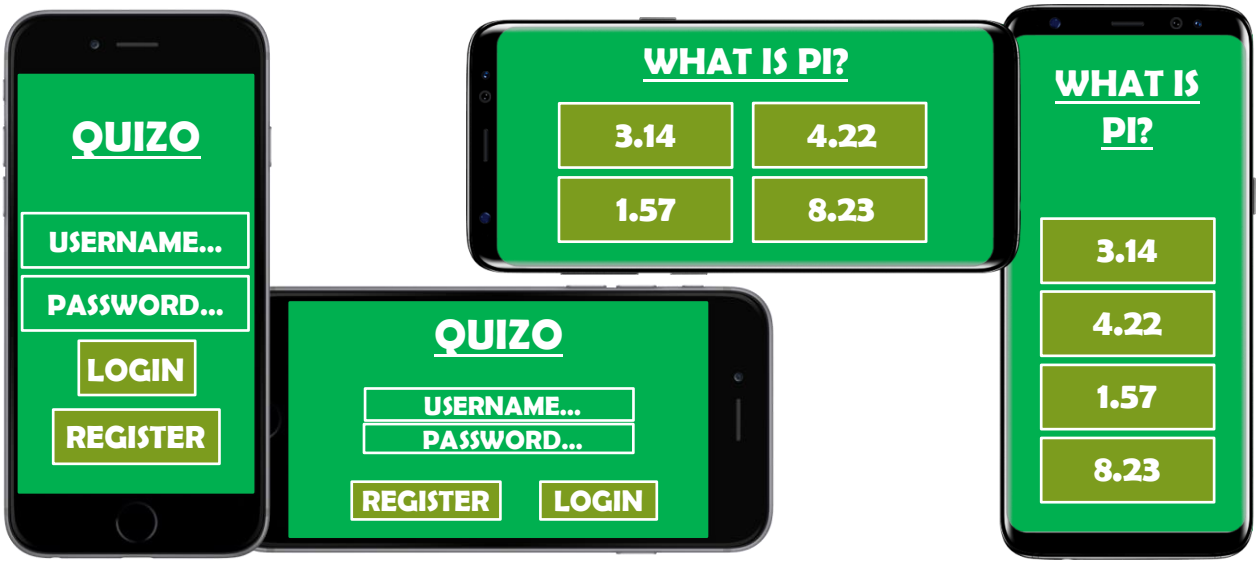
PORTS AND DISTRIBUTION

With the finished prototype, the program currently only runs on windows computers that already have Python, PyQt, and SQLite installed; obviously, this would not be desired for a public release.

Due to the extra modules that I had to install in order to create my program, just distributing the program file on its own won't allow users to run the program. Even distributing the program along with the database, and ui files from QtDesigner wouldn't be enough to be able to run the program due to not having the installed modules.

The best way to distribute programs would be as a single executable file which would install the program onto the computer. Having done a bit of research, there are multiple ways to do this, such as using tools like Py2Exe or Pyinstaller. Doing this would also require me to optimize the code for multiple platforms, such as Macintosh and Linux. Given some more time, doing this would be a viable option for the program.

In addition to other computer platforms, the overall goal is to port the program to as many different devices as possible, including mobile. In order to port to mobile, more things would need to be taken into consideration, such as how different phones have different screen sizes and resolutions, and also that phones can be used in both landscape and portrait modes. I have shown a few design ideas for how the program could look on mobile devices.



(iPhone 7 concept left, Samsung Galaxy S8 concept right)

In order to port the program to multiple platforms, the database for Quizo will need to be hosted online so that users can retrieve and write data to it no matter where they are. In order to do this, I could use MySQL as it has network capabilities and would allow the database to be stored online and for data to be read from and written to by any user, no matter the platform.

While porting to mobile would allow for a greater install base of users, the program could be developed into a web application, so that users can use their browser of choice to try the program, similar to how

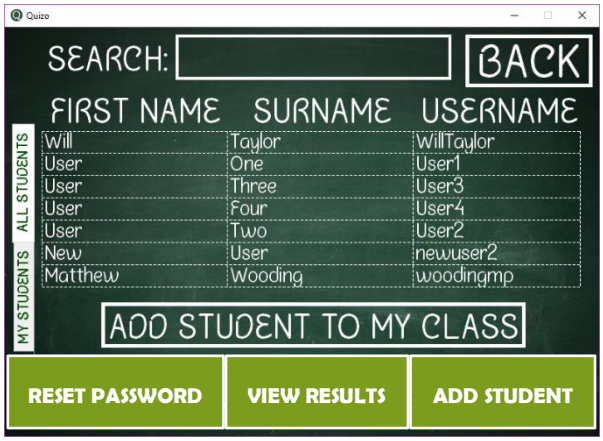
Sporcle and *Showbie* work. This would be possible, however could be more expensive due to the cost of having to host servers.

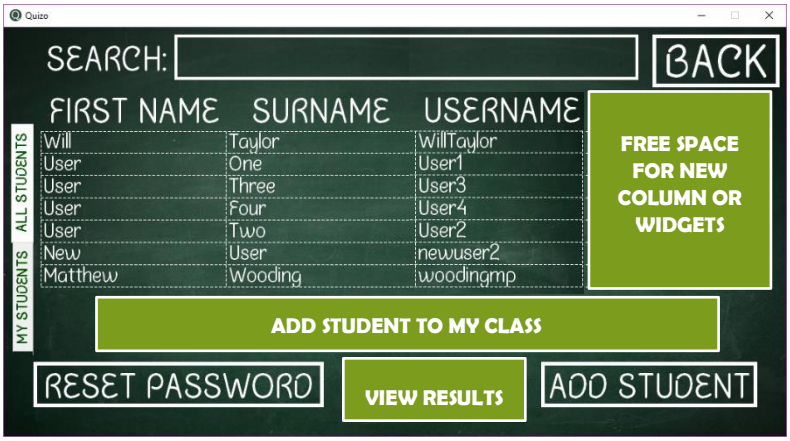
While all the ports and methods of distributing the program that I have mentioned would be possible, Python is a fairly simple language, and it would be more efficient to rewrite the program in another programming language, such as C++, so that it would be easier to maintain and support different platforms.

SUCCESS CRITERIA – FUTURE UPDATES

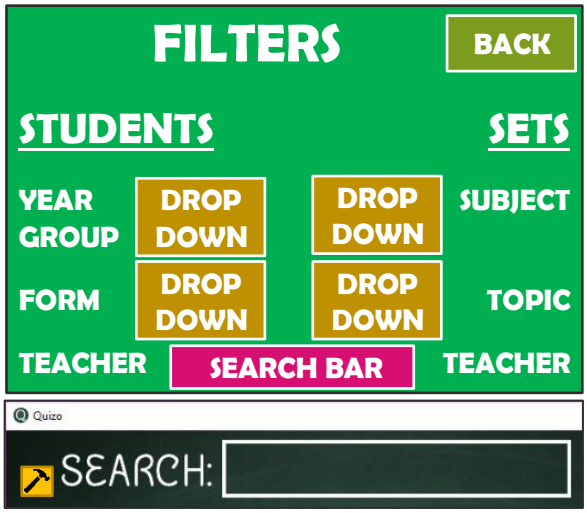
As shown in the review of my success criteria, the one objective that I did not complete was the easy-to-use and intuitive interface. The reason for this was that while the members of the RGS found it appealing, there were parts of the program that didn't fit in, and not always easy to understand at first glance.

One occurrence of this was found in the view students window when logged in as a teacher. Mr Boone, a teacher of the RGS, pointed out to me that he wasn't sure how to view the results of his students, since it requires a double click and has no corresponding button. This was something I missed in development because I was the one creating the program, so I automatically knew how to view results, where it isn't obvious to new users. In order to address this issue, I would need to add a new button in the window specifically for this, either by reducing the size of the existing widgets to make room(Left), or to increase the window size(Right). Both these ideas can be seen below.

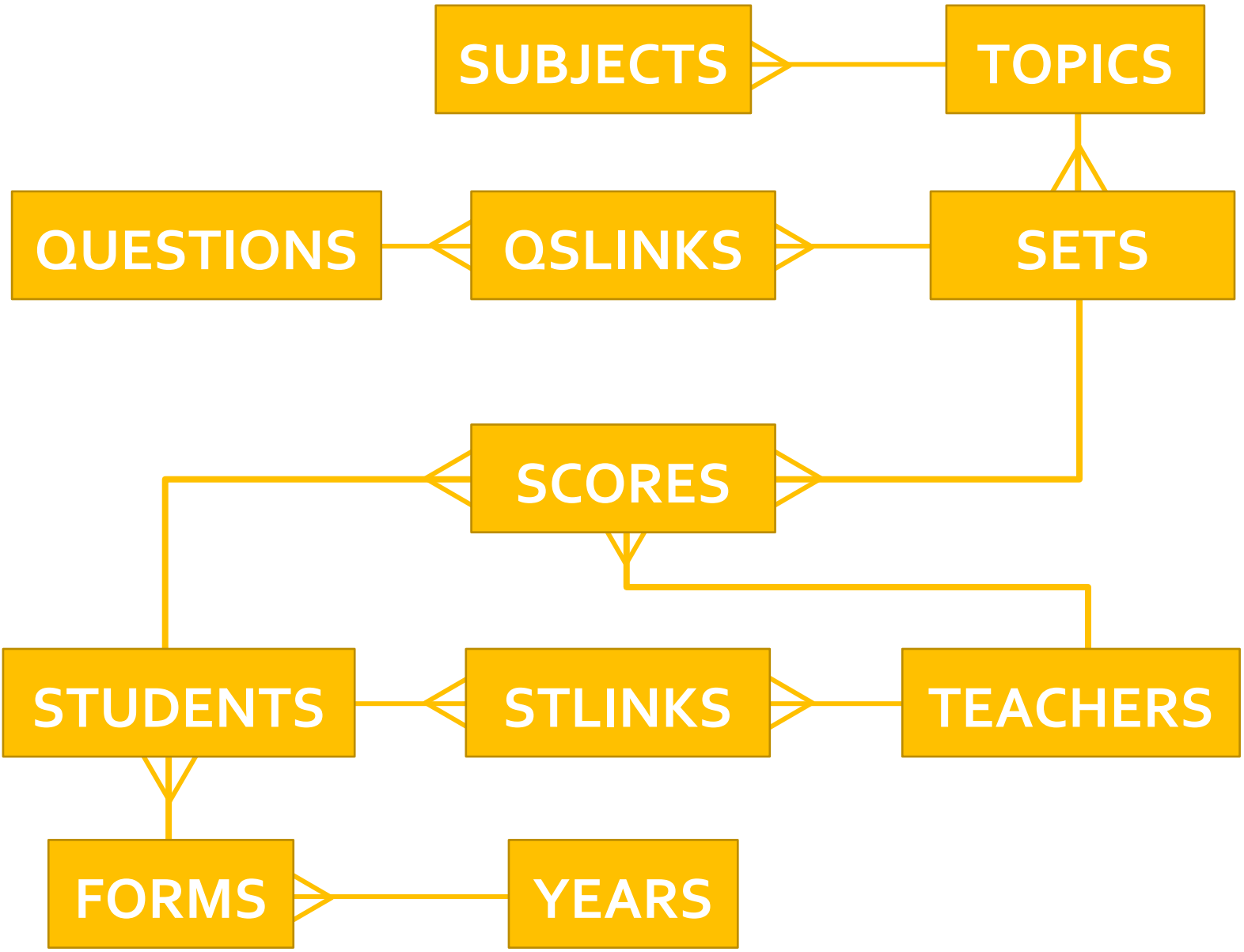




Another feature that some of the members of the RGS asked for was more filter options. Having thought about how I could integrate this in the future without cluttering the screen, I think that a new window could be used for all the possible areas where searching is used. The window could look like the design I have made to the side, with drop down menus showing all the year groups, forms, subjects, and topics available, and a search bar to search by teacher. This new window could be accessible by a small button next to search bars, as shown to the side.



The filter window mentioned above would be a useful way to filter students and question sets in the program, however in its current state, the program would be unable to accommodate this. In order to address this, the database would need to be changed to include new tables for year groups, forms, subjects and topics. Below I have created a small entity relationship diagram that shows how these new tables would be introduced into the database.



As seen above, question sets would be able to be assigned a topic (e.g. Integration) and then this topic will be a section of a subject (e.g. Maths) and so users would be able to search for relevant question sets by subjects or topics.

Similarly, each year group has a set number of forms, and students can only be members of a single form, so they will be able to be searched by year group or form.

FINAL EVALUATION OF QUIZO

As shown in the evaluation prior to this, there are some valid reasons why Quizo in its current state is unlikely to be released to the public for use in schools. However, with the additional changes that the members of the RGS have mentioned to me, and the initial thought I've put into how to go about these changes in future iterations, I think that Quizo could easily be a program used by students for revision purposes.

Mr Hamm expressed his interest in the prototype I developed, and this seems to be a good indication that other schools could follow suit in adopting the program for use in their curriculum once it reaches its first official release.