Computer Science NEA

Name: Bradley Makinson
Candidate Number:
Centre Name: Ashton Sixth Form College
Centre Number:

Contents:

I will be creating a revision app for my NEA using Python with a few other frameworks, such as PySimpleGUI. It will have similar features to some other revision apps that already exist (i.e. Quizlet), such as virtual flashcards, quizzes to test your knowledge, a progress tracker and more that will be discussed further later on. The key purpose of this app is to help students revise effectively for exams.

<u>**ANALYSIS SECTION**</u>

In this section, I will start to investigate the problem in order to start planning a suitable solution for the revision app I plan to create. The problem is students' needs to revise for upcoming exams and assessments, and my program will aim to provide a solution to that by allowing students to create their own revision sets and track their own progress.

<u>Why have I used a computational approach?</u>

A computational approach is most suitable for my solution for many reasons. Firstly, there are many mathematical processes that need to be carried out that would be much quicker and accurate if carried out by a computer system as the CPUs multiple cores can carry out many tasks quickly and simultaneously, whereas a human could only carry out one task at once. For example, calculating the average score, percentage and time taken to complete a quiz would take much longer for a human to carry out and would be less accurate - these features will be required to allow the user to track their progress. Specifically, calculating the average percentage of the user's test results will be calculated using the following calculation: (achieved score/maximum possible score) x 100, and this calculation will be carried out many times throughout the user's time on the app. It would take a human much longer to type this into a calculator, and there would be the potential for some inaccuracies in the results, so therefore a computational approach is more useful here. The purpose of the app is to provide a quick and easy way for the user to revise efficiently, and displaying statistics like these on the screen instantly helps achieve this goal by reducing the time needed to revise a topic as the user will not have to carry out these calculations themselves. By using a computational approach, repetitive, mathematical processes like these are guaranteed to be quick and accurate.

Furthermore, with virtual flashcards, the user cannot lose or damage them by accident as they are stored virtually in the computers memory, making this a much more reliable way of storing and using your revision materials. I plan to store the flashcards themselves in an external (.csv) file, meaning the user will not have to worry about having the resources to create real-life flashcards (e.g. paper, pens etc) or losing/damaging them as they are securely stored within memory rather than being a physical item.

There are also certain aspects of my revision app that would be impossible to implement in any other way than computationally. For example, sending tweets and email notifications would become extremely complicated, and therefore take much longer, if it wasn't done with the press of a button on the app itself. This is why I plan to use the Twitter API to automatically post a pre-written tweet by the press of a button.

My solution lends itself to the idea of abstraction – breaking down the problem into smaller, more manageable chunks – as will be described later in my design section. There will be a lot of different areas in my program, and breaking them down and solving them modularly via computational methods will make the solution as a whole a lot more manageable, and make my solution easier to create as the functions and subroutines I need to use will be laid out clearly.

Overall, a computational approach makes my solution easier to use, quicker to run and more convenient and organised for the user. Many of the processes that need to take place in my solution would simply be impractical and too time-consuming for a human to carry out repetitively.

Stakeholders

As my application will be for academic and revision purposes, I have identified my stakeholders to be students, mainly under the age of 25. My application will allow the user to create their own sets of revision materials, and these can be of any level of complexity, in any subject area. For that reason, the students my application is aimed at can be of any age, but mainly at GCSE and A-level students who have high amounts of content to remember, and a very short amount of time to memorise it. My application will be most useful for students who are close to sitting their exams and revise frequently, as the progress tracker feature will encourage students to return to the app frequently and build up their knowledge. Additionally, the user interface will be very simple, user-friendly and easy to understand, meaning there will be no confusion in terms of how to use the application, and therefore using the application should not be an issue for those students with very little IT knowledge.

In order to make my solution more tailored to the needs of my stakeholders, I will question three of my stakeholders about what they would expect in a revision app and how they would use it.
My three stakeholders are:

- Marcus, who is a 19 year old university student studying business.
- Vivek, who is a 17 year old A-level student studying Maths, Computer Science and English Language.
- Sam, who is a 15 year old GCSE student with an interest in studying physics at a higher level.

I feel this is a good range of ages and study levels, helping me to get a better general understanding of the needs of students at all levels, and all of these stakeholders will need to revise for upcoming exams in their various subjects.

Question 1: How would you like to log into the revision app?
Marcus: “Email and password”
Vivek: “Username and Password”
Sam: “Using the default username and password fields”

From the answers given to question 1, I am going to stick with my decision to use a username and password to log in. The users email will be entered when they sign up for email notification purposes.

Question 2: What features do you expect in a revision application?
Marcus: “Different sections for different subjects etc and a testing section for your subjects, as well as a social media page (I use Twitter most often)”
Vivek: “Creating Question Sets, Quizzes, Progress Tracker, Twitter updates, flashcards, quick and easy to use”
Sam: “topic grid so I know what’s revised so far and what’s not yet, a range of questions and topics and flashcards to revise them etc…”

Many of the responses given to question 2 link heavily with what I was already planning to include – creating revision material, revising that material using flashcards, quizzes/tests on that material, and a progress tracker/topic grid, as well as including the ability to post results to social media (namely Twitter)

Question 3: How would you use a revision application with these features (e.g. how often, how you would test your knowledge etc.)?
Marcus: “End of every week and before a test/exam”
Vivek: “Create Question Sets after writing notes, do quizzes before doing past papers on a topic in order to refresh my mind”
Sam: “Nightly sessions, test on topics I’m not as sure about and redo it for a few nights until I’m good at it”

The answers to question 3 will help me design my solution in a way that allows the user to have the best experience.

Question 4: Have you used any revision apps before, and if so, what were they and which features did you like best?
Marcus: “Yes – Quizlet. My favourite feature about it is the way you can test your knowledge with a quiz.”
Vivek: “No prefer paper. Mostly use flashcards because they are portable.”
Sam: “Quizlet and Memrise, I liked the ability to see which parts have been covered and which haven’t”

In question 4, the idea of portability was mentioned in one of the responses – my solution will be available on any computer system that meets the software and hardware requirements (discussed later). I am also planning to include a quiz feature in my app as mentioned by Marcus.

Question 5: Why would you use a revision app?
Marcus: "In case I had a test coming up (especially because of my uni exams) and it would refresh my memory about certain topics"
Vivek: "To access a way to revise on the go and to remember key definitions before exams / last minute practice"
Sam: "Easier and more convenient revision"

The idea of revision of key ideas just before an exam was brought up in question 5, suggesting the application may be used on a more short-term basis by some students, perhaps removing the need for a long-term progress tracker.

Question 6: How would you like the program to look, and what would be the best form of inputs for you?
Marcus: "Not overly complicated, clear instructions and input boxes/buttons as I don't want to spend ages figuring out how to use the app rather than revising"
Vivek: "Just be as easy to use as possible, prefer inputs via presses of a button than textual inputs for example"
Sam: "Having used many revision apps before, I always preferred apps that were simple and you could revise content or test your knowledge via the press of a button"

Its clear that my stakeholders want a simple and easy-to-use solution, with buttons seemingly being the preferred source of user input.

Question 7: How long before your exams would you start using a revision app?
Marcus: "2 weeks at a minimum"
Vivek: "6 months before"
Sam: "A few weeks before rather than months because a revision app is for more quick, snappy revision"

The idea of quick and snappy revision came up again in question 7, although the times that my stakeholders would start revision before an exam varied massively – from 2 weeks to 6 months. For this reason, I will still keep my long-term progress tracker as some students may wish to use the application over a longer period of time than others.

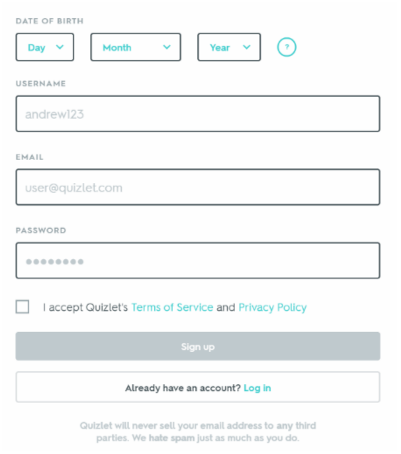
These questions and answers from my stakeholders will help me improve my solution and amend it to include anything that my stakeholders say they would find useful in a revision app. They will help me to understand which features are most useful for my stakeholders based on their past usage and experience of revision apps, and thus ensures my solution is designed with my stakeholders' needs in mind.

Existing solutions

There are many already existing revision apps available, and I will research and investigate these to ensure I understand what my stakeholders need in my solution.

Solution 1: Quizlet

Quizlet is a free-to-use revision app for students that allows the user to create sets of terms and definitions on a particular subject, and then revise these in many ways.
Link: https://quizlet.com/en-gb
Features:

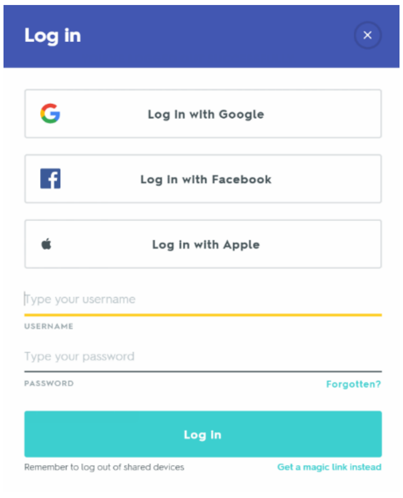


DATE OF BIRTH
Day Month Year
USERNAME
andrew123
EMAIL
user@quizlet.com
PASSWORD
••••••••
☐ I accept Quizlet's Terms of Service and Privacy Policy
Sign up
Already have an account? Log in
Quizlet will never sell your email address to any third parties. We hate spam just as much as you do.

This is the sign up screen on Quizlet. You have to enter your date of birth, choose a username and password, and then enter your email.

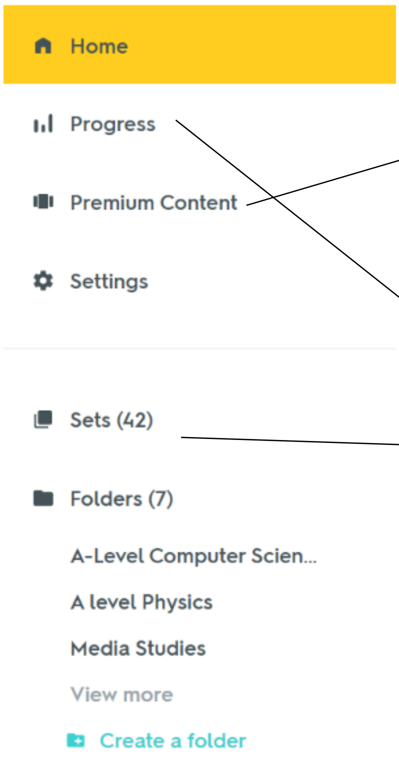
My solution will require the user to choose both a username and password for log in purposes (these will be stored in a secure .csv file), and the user's email for email notifications (this will be discussed in more detail later).

I will not require the user to enter their date of birth for any reason, and hence this will not be included. Similarly, I will not have any reason to ask the user to accept any terms or conditions, and so this will not be included either.



This is the log-in screen on Quizlet. You can either log in via Google/Facebook/Apple, or simply log in with your username and password.

My solution will only offer the option to log in with your username and password, and these will be stored in a secure .csv file. Your email will still be required later in the program for email notifications.

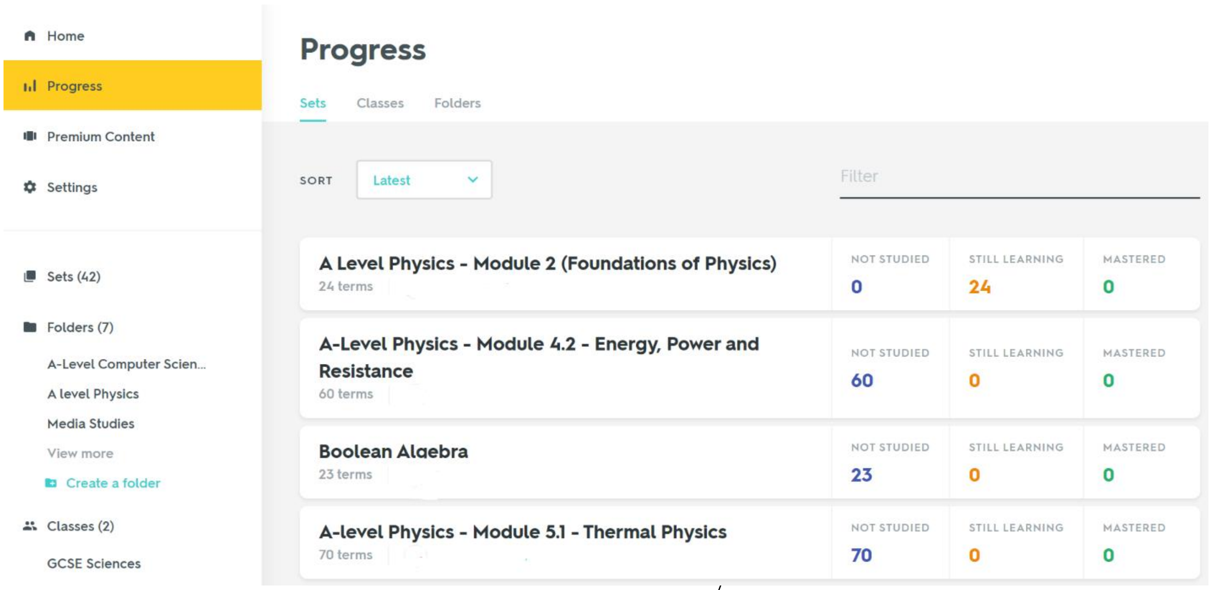


Price: Quizlet is free, but you can pay £19.99 a year to get access to premium content (e.g. images on flashcards).

My solution will be completely free (no premium content)

On the menu, the user can view all of their sets, and folders, as well as look at their progress.

This will be similar for my solution in order to create a user interface and menu which will be simple and user-friendly.



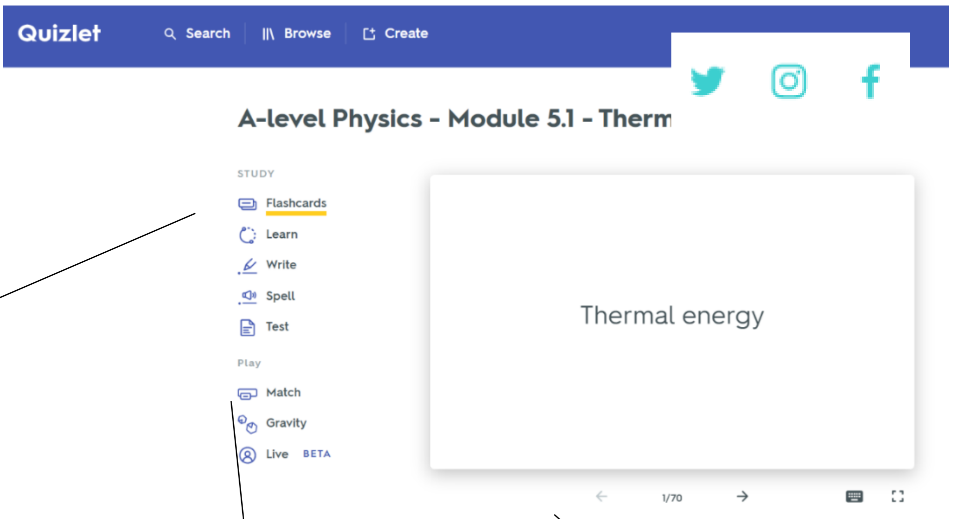
Progress

	NOT STUDIED	STILL LEARNING	MASTERED
A Level Physics - Module 2 (Foundations of Physics) 24 terms	0	24	0
A-Level Physics - Module 4.2 - Energy, Power and Resistance 60 terms	60	0	0
Boolean Algebra 23 terms	23	0	0
A-level Physics - Module 5.1 - Thermal Physics 70 terms	70	0	0

On the progress page, the user can see each of their sets, how many of the flashcards they have learnt and how many they haven't. They can then view these flashcards individually.

I will include a progress tracker in my solution too, and it will display time spent revising it, average score and average time taken to complete the quiz on a particular topic. These will be recorded as variables and so can be changed and displayed as needed.

I will also provide the option to post results on social media platforms, namely Twitter, using the Twitter API in order to create a further incentive for my stakeholders to keep competing with each other.



Users can create their own sets of flashcards, as well as browse sets of flashcards made by other people.

My solution will include the ability for the user to create their own sets of flashcards, which they can amend or delete at any point, allowing for a more personalised experience. These flashcards will be saved to a .csv file, and then I can write to and read from this file whenever necessary.

I am not planning to include the feature of sharing your sets with others, as this is not a web app, and I would like this to be a more personalised application.

Users can then revise the set of flashcards in many ways – either in the form of traditional flashcards, by writing out the definitions, or by taking a quiz to test their knowledge on that particular set.

My solution will include the ability to revise flashcards in the traditional way (look at term, say definition, check definition), as well as a quiz feature, which will allow the user to check their understanding and track their progress on a particular topic.

There are also some mini games that can be played to help the user learn in a more fun and interactive way.

I will not be including any of these mini games in my program, as I believe there are already enough features for effective and engaging revision.

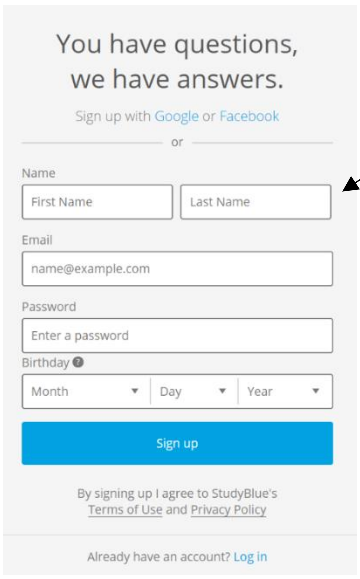
Advantages	Disadvantages
Effective flashcard feature	Only raw numbers on the progress tracker rather than a visual representation (e.g. a graph)
Simple GUI and layout	Browser app will require internet access
Unlimited sets/classes, and can share sets.	

Solution 2: StudyBlue

Link: https://www.studyblue.com/online-flashcards

StudyBlue is another free-to-use revision tool that has many similar features to Quizlet.

Features:



You have questions, we have answers.

Sign up with Google or Facebook

or

Name

First Name | Last Name

Email

name@example.com

Password

Enter a password

Birthday

Month | Day | Year

Sign up

By signing up I agree to StudyBlue's Terms of Use and Privacy Policy

Already have an account? Log in

The sign up page on StudyBlue asks for the full name of the user, their email, a password and their date of birth.

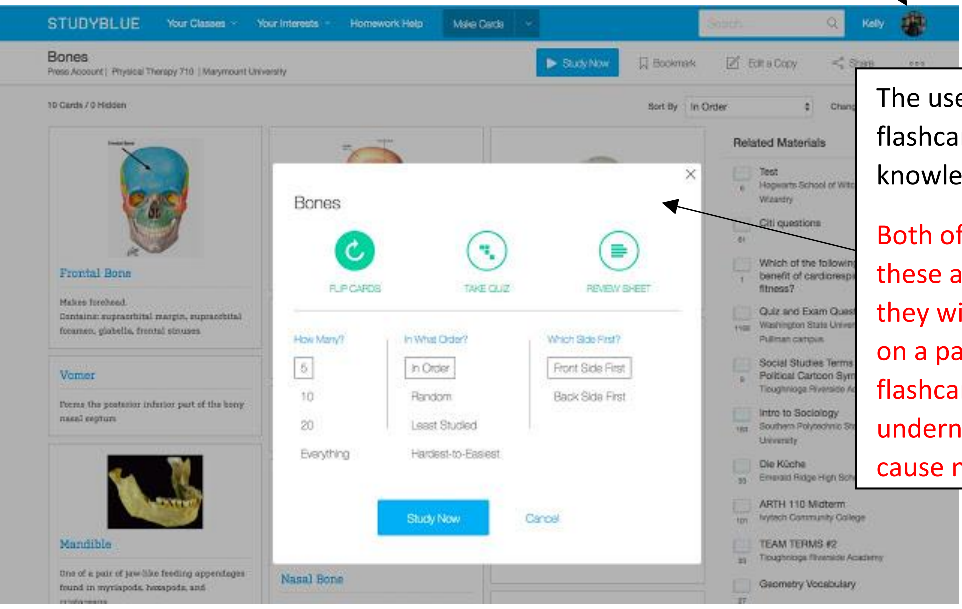
My solution will only require the user to enter their username and password (for log-in purposes) and their email (for email notifications). I will have no need for the user's full name or date of birth, and thus these will not need to be entered.

The log-in page for StudyBlue requires the user's **email** and password.

Alternatively, my solution will require the user's **username** and password as this is quicker and easier for the user to enter

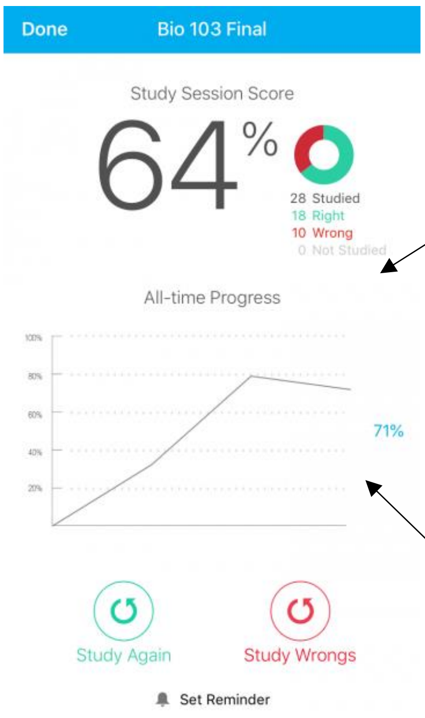
Similarly to Quizlet, StudyBlue allows the user to both browse sets of flashcards made by other students, and make their own sets on whatever subject they like. They can also edit their sets (remove/add flashcards).

As stated earlier, my solution will allow the user to create their own sets of flashcards and edit them whenever they like, but I am not planning to include the ability to share sets between users as I want this to be a very personalised application.



The user has the option to either revise the set as flashcards ('flipcards') or take a quiz to test their knowledge.

Both of these will be features in my solution as I believe these are the 2 most effective methods for revision, and they will be the easiest ways to track the users' progress on a particular topic. The layout will consist of the flashcard itself, then a 'next', 'previous' and 'flip' button underneath – a simple and clean GUI design that will cause no confusion for the user.



There is also the ability for the user to study solely the definitions they got right/wrong, and set reminders.

My solution will show the user which definitions they got right and wrong, and will send the user reminders to revise via email notifications. These will only be able to be sent whilst the program is running, however.

The progress tracker on StudyBlue is relatively similar to Quizlet – for each revision session the user completes, the percentage score, and a graph of their percentage scores in that particular subject, appear on the screen. It also shows exactly how many definitions the user got right/wrong.

The progress tracker in my solution will give an average score and percentage for a given set of flashcards. I will do this by creating two variables for each set that will be updated every time the user takes a quiz on that set.

Advantages	**Disadvantages**
Detailed progress tracker with visual representations	Browser app requires internet access
Can list your interests when creating an account so the app can recommend more relevant sets for you to study.	Slightly more complex layout than Quizlet, but many GUI features are the same.
Unlimited sets/classes, and can share sets.	

Solution 3: Real-life flashcards
My solution is heavily based on the concept of flashcards – one side of the card has a term/question on it, the other side has the definition/answer on it. My solution allows users to do everything they would be able to do with real flashcards, but virtually.
Real-life flashcards are a more feasible solution for those without consistent access to a computer, due to the technological requirements of revision apps.

But why would my stakeholders choose to use a virtual app over actual flashcards? What are the advantages and disadvantages of a virtual app compared to paper flashcards?

Advantages	Disadvantages
All in one place – cannot lose the flashcards	Hardware and software requirements
No cost (do not need to buy resources, such as card, pens etc.)	Power consumption by running a computer system
Quick and automatic test scores and progress checking	Cannot access anywhere – only accessible on a computer.

Main features of my proposed solution

1.) **Log-in/Sign-up screen** – on this screen, the user can either log-in to their account (if they have already created one) or create an account if they haven't already by signing up. They can choose either of these options by pressing either the 'Log-in' button, or the 'Sign-up' button. When the user signs up, they will be asked to enter their desired username, password and email into the textual input boxes, and these will be saved into a .csv document, which can then be referred to and read from in the future when the user logs back in. On the log in screen, the user will have to enter their username and password into the two free input boxes labelled 'username' and 'password' – these will then be checked to see if they're correct by checking the corresponding boxes in the .csv document (I.e. if person 1's username is found in box A1, then box A2 will contain their password and A3 will contain their email. When the username in box A1 is entered into the 'username' input box, the password in A2 is checked, and if it matches what the user has entered in the 'password' input box, they will be allowed in. If not, a message will appear that says 'incorrect password'). I have included this as it adds personalisation to my program and allows the user to have their own account – it is a simple and user-friendly design that will cause no confusion.

2.) **Menu screen** – Once the user has logged in, they will be taken to the menu screen. Here, they have the option to create a new set of revision material, edit an already existing set by adding and removing flashcards, revise a set, take a test on a set of their choice, or review their progress via the progress tracker. If I decide to add the ability for the user to set exam deadlines in order to keep them motivated to revise, this will also be a 'Set Deadline' button on the menu screen. These will all be buttons that can be pressed. There will also be an option to log out and exit the application (which will automatically log you out), both of which will both be buttons which can be pressed. Once again, this is a simple design that is easy to use.

3.) **Flashcard screen** – If the user chooses the option to revise a set, the flashcard screen will appear. In the centre of the screen, there will be the 'flashcard' itself, and then underneath there will be a 'next' and 'previous' button to allow the user to go to the next/previous flashcard. There will also be and a 'flip' button, which will 'flip' the flashcard (i.e. if the user is looking at the term on the flashcard and then presses the flip button, it will show the definition, and vice versa). There will also be a button to delete the flashcard the user is currently looking at, and one to add a new flashcard at the end of the set. If I decide to implement the functionality to allow the user to edit the content on already existing flashcards, there will be a button for this too. Each of these buttons will be a picture of the appropriate symbols (e.g. arrows) to make it quicker and easier for the user to recognise their functionality. Once all of the flashcards in the set have been gone through, it will go back to the start of the set. I will do this to make it much quicker to go from the end of the set of flashcards to the start, and vice versa. The user will be able to exit revising that set at any point via an ‘Back’ button.

4.) **Quiz screen** – This aspect of my solution is designed to allow the user to check their knowledge on a particular subject to check whether they fully understand it or not. The layout will consist of the question in text at the top of the window, and then 4 multiple choice answers underneath. These multiple-choice answers will be in the form of buttons, so the user can press the button that they think has the correct answer on it, and then the screen updates with the next question and 4 possible answers. At the end of the quiz, the final score, percentage and time taken will be displayed, as well as an option to share the result on the applications twitter page. It is during and after the quiz that most of the complex processing will be done – the score and percentage will be calculated and updated on the progress tracker, and the questions will be randomly chosen. I could also decide to add the option for a text-based quiz rather than multiple choice, where the user would have to type in the answers into a free input rather than press a button.

5.) **Progress Tracker** – The user will be able to track their progress here – they will be able to see their best to worst sets (in terms of test percentages), their highest scores and their exam deadlines all in the form of textual outputs on the screen, with different colours to represent good (green) and bad (red) test scores. I could also decide to implement this in the form of a line graph in order to give the user a better visual representation of their progress, showing a general trend of how they are doing and thus allowing the user to see more clearly which areas they need to improve in.

Summary of key/essential features and their justification:

Feature	Justification
User can create an account	So that their individual progress can be saved. A username, password and email will be required to be entered, as these were the fields mentioned by my stakeholders.
User can log into an already existing account using their username and password	So that the user can return to their work when the application is re-run. All revision apps I researched had the create an account/log-in feature, and it is something my stakeholders wanted when I questioned them earlier on.
User can manage their account	This feature gives the user the ability to change their details or even delete their account, allowing for improved security

	as passwords and emails can be changed.
User can create sets of flashcards with a term and a definition	So that the user can make use of the application's other features such as revising sets and tracking progress. All other researched solution also included this ability in order to allow the user to tailor their revision.
User can revise sets of flashcards	This is the main purpose of all revision apps – to allow the user to revise. My stakeholders also mentioned that flashcards were a method of revision they have found effective before.
User can test their knowledge on a particular set of material via a quiz, and see their results.	This is another method of the user consolidating their knowledge that I found was consistent with all revision apps I researched. It was also a feature my stakeholders mentioned they would benefit from.
User can view their progress via a progress tracker within the app.	The existing solutions that I researched all included this feature in one way or another. My progress tracker will include both visual representations of data (e.g. graphs) and raw numbers. This will help the user to effectively track their progress and see where they can improve, which was mentioned to be beneficial to my stakeholders.
User can send email notifications of their progress.	This will serve as a reminder to the user, and is a feature that is quite unique to my solution as none of my researched solutions included it.
User can post their progress to social media, namely Twitter.	This is a way to motivate students to revise more as they will be competing with each other, and this is another feature that is relatively unique to my solution but that is because my researched solutions were designed for a much larger scale rather than a single class/school size.

Limitations
One limitation of my proposed solution is that there may be a maximum number of 5 sets of flashcards per user (this may yet be increased or even entirely removed). This will be to limit the number of sheets in the .csv file and make the program more manageable and reduce the amount of memory needed.

Another limitation of my proposed solution, as I mentioned earlier on in the report, is that email notifications will only be able to be sent whilst the program is running. This is because the way email notifications will be sent out is when the user passes a quiz, or when they press the 'email notification' button in order to send out a reminder via email. This means that when the application is not being ran on a computer system, email notifications cannot be sent out. The email notifications will also only be for gmail accounts as I will be using the Google API.

My proposed solution will also not allow users to share their sets with each other as many other already existing revision apps do. This is because my solution is an individual revision app. However, due to the limit of 30 users, I may decide to make my solution tailored towards being a class application, where the whole class can sign up and create an account. As part of this, I would have to allow the users to share their sets with each other and create a leaderboard feature.

Design Requirements

Requirement	Why it is required
GUI layout that is easy to use and understand	To make it a more user-friendly experience
Clear and obvious instructions and buttons	So that the user knows exactly how to do what they want to do.
Reasonable image and font sizes	So that the text/images are not too small to see, but also not so large that they are taking up the whole display.
Colour coding	Whilst **not** an essential requirement, some colour coding (e.g. green text for a correct answer and red text for an incorrect answer) will improve the experience for the user.

Functionality Requirements

Requirement	Why it is required

Usernames and passwords need to save to an external .csv file	So that the user can successfully log in to their account again when they next come to use the application.
Flashcards created by the user also need to save to an external .csv file	So that the user can revise using the flashcards they have created by using the 'flashcard' functionality.
When certain buttons are pressed, the correct display needs to appear (e.g. if the 'Flashcards' button is pressed, the program needs to take the user to the flashcard screen).	So that my stakeholders can use the solution correctly and effectively, and use the wide range of features available.
Quizzes need to display random questions in a random order	So that they do not become repetitive and to ensure that the user has a full knowledge of all the content they need to revise.
Test score percentages and times need to be consistently recorded	To ensure that the progress tracker works correctly and gives the user a reliable measure of their progress in a certain topic (so they know where they need to improve).
A main menu once the user logs in	The main menu will contain buttons for all of the available features that the user can use – all in one place.

Software Requirements

Requirement	**Why it is required**
An operating system compatible with running Python 3.x.x (Window/MacOS/Linux)	To run the finished program
A gmail account	To be able to send email notifications, as these notifications will be sent using the Google API, so the user would need a gmail account.
Python 3.x.x installed with the PySimpleGUI library	This is required to run the application as this is the framework it has been created with.

Hardware Requirements

Requirement	Why it is required
Mouse	In order to be able to click buttons and navigate around the GUI, and input data to the computer system.
Speakers	This is one way data/information is outputted to the user. It is required in order to be able to hear some of the sound effects I will be adding, such as a 'ding' sound when the user passes a quiz etc… This requirement however is NOT essential.
Keyboard	In order to input important details when signing up or logging in (such as usernames and passwords), as well as for text based quizzes (if I decide to add them) where the user would have to type in their answers. The app requires a keyboard input for the password, username and email features.
Monitor/screen	This is how the application and its data is outputted to the user.

Success Criteria

No.	Criteria	Justification	How it can be proved
1	The stakeholder would like the system to allow them to create an account if they haven't already got one.	This is the first thing the user will have to do when they use the application.	Screen record a stakeholder entering their details into the sign-up page, which then takes them to the menu if their details are all valid/haven't been used before.
2	The stakeholder would like the system to allow them to log in to their account if they have already created one.	This is the first thing the user will have to do when they use the application, and both this and the sign-up feature were requested by	Screen record a stakeholder entering their username and password into the log-in page, which then takes them to their menu if the details they entered are all correct

		my stakeholders and were common features among already existing solutions.	and valid. If not, an error message will appear.
3	The stakeholder would like the system to allow them to create their own sets of flashcards.	So the user can revise whatever content they wish – the main purpose of the app.	Screen record a stakeholder creating a flashcard, then show a screenshot of this flashcard saved into an external .csv file.
4	The stakeholder would like the system to view sets of flashcards they have already created.	So that the user knows what revision material they have/haven't already created.	Screenshot of the screen showing a stakeholder's already existing sets of flashcards.
5	The stakeholder would like the system to allow them to revise sets of flashcards they have created.	This is the main purpose of the revision app – allowing the user to revise. Flashcards were a popular method of revision among my stakeholders.	Screen recording of a stakeholder going through a set of flashcards and using all of the available features (next/previous flashcard, flip flashcard)
6	The stakeholder would like the system to allow them to test their knowledge on a particular set of flashcards.	This is an extremely important way of the user seeing where they need to improve, and was a common feature among my stakeholders requests and already existing revision apps.	Screen record a stakeholder taking part in a quiz and reviewing their score and time taken to complete the quiz.
7	The stakeholder would like the system to keep track of their progress and show it in a clear and concise way.	Once again, this is a way for my stakeholders to see which areas they aren't scoring highly in, and the ability to track progress over a ranging period of	Screenshot of the progress tracker feature showing a range of statistics (discussed earlier) about their progress.

		time was a feature mentioned by all of my stakeholders earlier on in this section.	
8	The stakeholder would like the system to allow them to share their results and progress on social media.	This adds an incentive for the user to keep revising and compete with other students. Twitter was a commonly mentioned social media platform among my stakeholders and is a popular platform among teenage students.	Screenshot of tweet that has been shared by a stakeholder.
9	The stakeholder would like the system to have a clear main menu.	Makes the program easier to navigate which is crucial to providing a user-friendly experience.	Screenshot of the main menu.
10	The stakeholder would like the system to be user-friendly and easy to understand.	The user should not be confused at any point whilst using a revision app – it should be extremely user-friendly. This links to how my stakeholders preferred a revision app that can be used for quick, snappy revision, so my solution needs to be easy to use so the user is not spending too long figuring out how to use the app.	Screen recording of some general use of the application (e.g. logging in, navigating the main menu etc.)

11	The stakeholder would like the system to allow them to edit already existing sets of flashcards.	This allows the user to add to their revision material as they go, helping to keep the application organised and effective.	Screen recording of a stakeholder editing (adding/removing) some flashcards from an already existing set that they have created.
12	The stakeholder would like the system to allow them to set exam dates/deadlines.	This is to help keep the user organised and aware of when they need to have revised material for – these will show up on the user's reminders list on the main menu.	Screen recording of a stakeholder setting an exam date / screenshot of exam dates set by a stakeholder.
13	The stakeholder would like the system to send them email notifications when prompted to.	This serves as a method of reminding my stakeholders that they need to revise on a consistent basis.	Screenshot of email sent to a stakeholder.

Questionnaires
I have created multiple questionnaires about my solution in order to monitor user feedback throughout my project to measure the degree of success.

Questionnaire 1: Sign-up/Log-in
1.) Can you successfully create an account on the app?
2.) Can you successfully log back into this account using the same details?
3.) Was this an easy process to understand and carry out?
4.) Do error messages appear when incorrect details are entered?

Questionnaire 2: Creating/editing/revising a set of flashcards
1.) Can you successfully create a set of flashcards with a given name?
2.) Can you then add flashcards to this set?
3.) Can you remove flashcards?
4.) Can you 'flip' flashcards for revision purposes?
5.) Is the process easy to understand and carry out?
6.) Do you feel the revision is efficient?

Questionnaire 3: Quizzes
1.) Does the program allow you to take a quiz on a set of flashcards you have created?
2.) Was the quiz thorough enough?
3.) Were the questions in a random order?
4.) Did the multiple choice/written quiz work correctly and were you given a valid score, percentage and time taken upon completion?

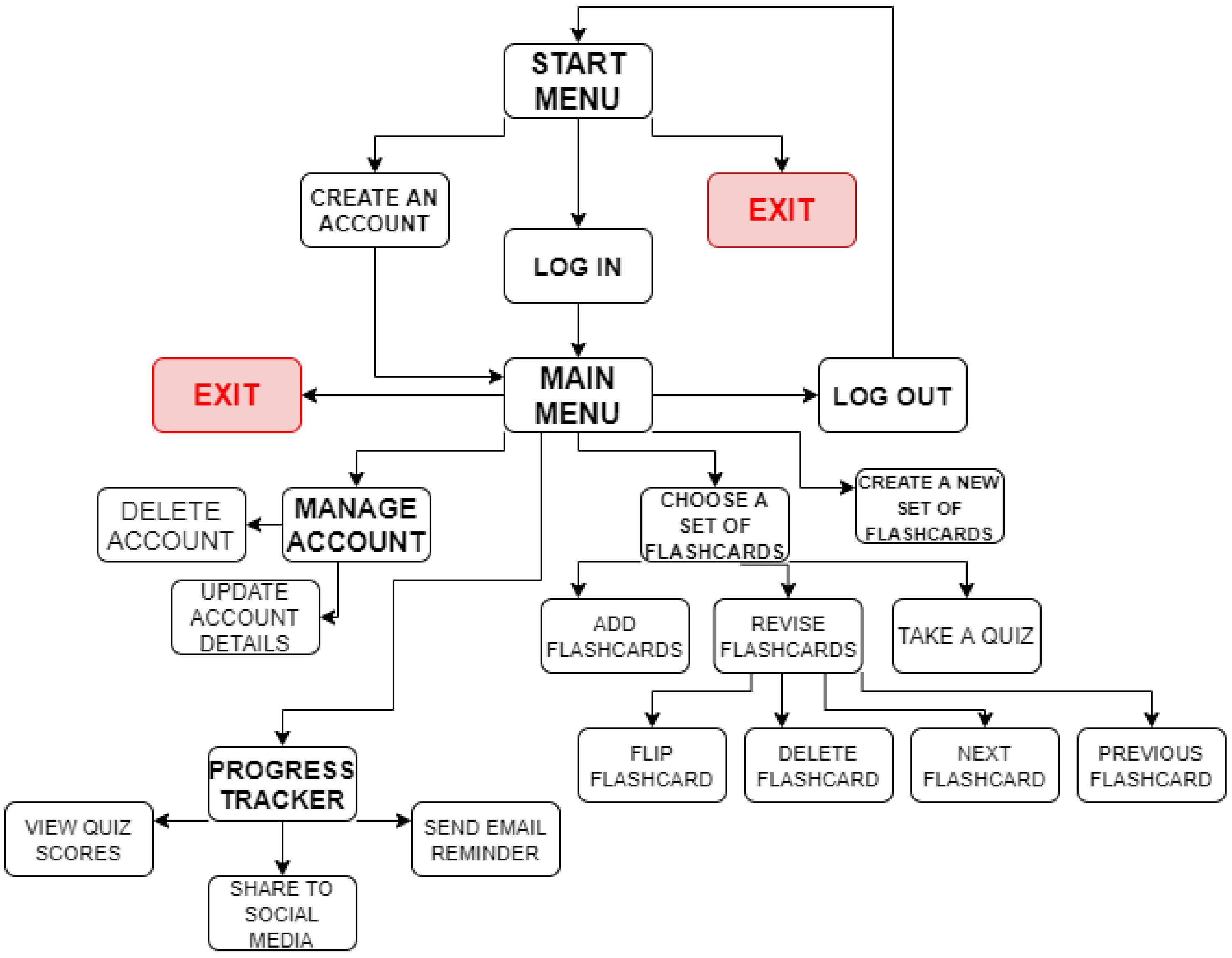
Questionnaire 4: Progress Tracker
1.) Can you successfully access and view the progress tracker feature?
2.) Are the statistics shown correct?
3.) Do you feel this feature is helping you?
4.) Can you successfully set exam dates/deadlines, and do these show on the main menu?
5.) Are email notifications successfully sent and received when you prompt the program to send one?
6.) Can you successfully share your results on the application's Twitter page?

Questionnaire 5: General
1.) Is the app easy to use and navigate?
2.) Is the GUI layout simple and not confusing?
3.) Are there any errors/problems you encountered whilst using the app? If so, what were these?
4.) Did the sound effects work correctly?
5.) Is the colour scheme easy on the eye?

Design Section

In this section I will design my solution by decomposing it into smaller subroutines, planning out each of these subroutines, and describing any variables which may be needed and how my solution may be tested during and after the development phase.
After analysing my solution, I will first break my problem down into smaller subroutines to make my solution easier to create and test later on.

Systems diagram



Systematic breakdown of problem:

The start menu is required to show up first and be the first display the user sees because this is where the user has the option to either create an account or log in to an already existing account. This step is required first because each account is personalised, so in order to make the experience as tailored to the user as possible, they will need to log into their account before they use any other features of the app. Creating an account/logging in before anything else is an authentic feature of almost all revision apps, so following this in my program is key to making my solution as simple and easy to use as possible.
There will also be an option to exit the app from the start menu.

Once logged in to their account, the user will be taken to the main menu. This is the main hub of the application from the user's point of view as it will be where they can access all of the features of the application. Each of the options (exit, manage account, progress tracker etc) will be displayed as clear buttons on a contrasting background, making my solution as clear and user-friendly as possible. I have shown the main menu to be separate from the start menu in my systems diagram (above) because a user can only advance to the main menu once they have logged into their account, and their inputs to the username and password fields need to be validated.

There will be various different features of my program that you can access from the main menu. You will be able to exit the program, which will automatically log the user out, or log out manually, which will take you back to the start menu as shown in my systems diagram. There will also be an option to manage your account, where the user can change account details (such as their email address) or even delete their account. I have broken down 'Manage account' into the two smaller problems 'Delete account' and 'Update account details' because each of these problems will require a different subroutine – updating account details will involve re-writing to the user details file, whereas deleting an account will involve removing all of the information associated with that account as well as any sets of flashcards or progress that user has made.

The progress tracker contains many features itself, each of which will require a separate subroutine. A user's quiz scores will be saved to a file whenever they complete a quiz, and these will be viewable via the progress tracker. There will also be options to post your progress on the applications Twitter page using the Twitter API and send an email reminder of your progress to yourself using the Google API. I have abstracted these three features of the progress tracker separately because each of them have a completely different purpose, and will be programmed using separate subroutines.

There will also be options from the main menu to use existing sets of flashcards or create a new set.

Creating a new set of flashcards will simply involve the user entering the name of the new set, the name of which will be saved to an external .csv file and will be saved to the main menu. They can add flashcards to this new set via returning to the main menu and clicking on the set name.
The menu will also display each of the users sets of flashcards they have already created (I am planning for there to initially be a maximum of 5 flashcard sets per user, although this may be increased or even entirely removed later on). They will be able to click on one of the sets, which will be displayed in the form of a button, and there will be 3 options:

- Revise that set – in the form of traditional flashcards (this will involve reading from the .csv file and displaying the terms/definitions on screen) – meaning the user can 'flip' a flashcard and go to the next or previous flashcard. I have also added the ability for the user to delete the displayed flashcard from this screen. These will all use separate subroutines to carry out their functions and hence I have abstracted them separately.
- Add flashcards – this will involve the user adding flashcards to an existing set. This feature will involve writing to (changing) the .csv file, so I have abstracted it separately. This has been shown separate to revising a set because revising a set will involve reading from the file, so this will require a different subroutine.
- Take a quiz – this allows the user to test their knowledge by answering questions about a set of content. Whilst this involves reading from the .csv file, it is abstracted separately to revising a set because the purpose of the two functions is completely different, and the quiz will involve randomly reading from the file rather than doing it in a set order.

Structure of solution:
Start menu – this will be the first screen the user sees when they start the app. It will have 3 buttons for either logging in, creating an account or exiting the app, as well as a short paragraph of text explaining the functionality of the app and who it is best for. This is a simple, easy to navigate layout that will appeal to my stakeholders, and will be achieved using PySimpleGUI's 'Button' and 'Text' features. The ability to log in or create an account will be linked via the press of a button, which will take the user to a different layout where there will be text fields where the user can either log in or sign up by entering the relevant information.

Create an account – here the user will enter their desired username, password and gmail account into the relevant input boxes, which will be clearly labelled. I chose the username and password method of user verification after consulting my stakeholders in the analysis section, as they found usernames and passwords to be the easiest and most secure method of user verification. The user will be asked to enter their password twice to ensure there are no typos (and if the two entries do not match, the user will be notified in red text at the bottom of the screen). Once all of this information has been entered and is correct, they will be able to press the 'Create' button, which will store their data into a .txt file. Passwords will be hashed before they are stored in order to make my app more secure. I have decided to store the users account information into an external file so that their information is saved even when the app is re-run, and the user can log in again using the account they have previously created. The user will then be automatically navigated to the main menu.

Log-in – if the user has already created an account in the past, they can log in using the same username and password, and then these will be verified to see if they are correct by reading the data saved into the external file when the user made their account. If the username and corresponding password entered into the input boxes do not match any username and password in the .txt file, then an error message in red text saying "Invalid username/password" will appear at the bottom of the window, prompting the user to re-enter their username and password. If the username and password is correct for a certain account, then the user will be directed to the main menu, and the email variable will be set to the users email address.

Exit application – This will be an option from both the start and main menu which allows the user to exit the application. Exiting from the main menu will automatically log the user out. I will implement this by breaking the event loop whenever the exit button is pressed.

Main menu – this will be main 'hub' of the app, and it is where the user will be taken once they have logged into their account. It will be laid out in a simple way with buttons for each of the user's sets, as well as options to create a new set, manage their account, access the progress tracker and exit the application, all of which will also be in the form of buttons, once again via the use of PySimpleGUI's 'Button' feature. This once again links to how my stakeholders wanted a simple, quick and snappy GUI design allowing them to fully focus on their revision rather than working out how to use the app.

Manage account – When this button is pressed, it will take the user to a layout where they can either:

- **Delete their account** – this will delete all of their information stored in the external files, including their username, password, email and any sets of flashcards/progress they have made.
- **Edit account details** – this will allow my stakeholders to change their password, username or email address, meaning that if they change their email at any point, they can continue using the app on the same account so that their progress/sets are not lost. This will be done be writing to the external file which stores all of the users' usernames, passwords and email addresses.

There will also be a ‘BACK’ button on this display, as well as at most points throughout the application, so that the user can go back to the main menu (if they accidently clicked on this button for example) via a press of this button.

Log out – This option will take the user back to the start menu and reset the email variable, ready for another user to use the app.

Create a set of flashcards – Accessible from the main menu, pressing this button will take the user to a screen where they can enter the name of the new set they wish to create, which will be written to a different external file. The user can then return to the main menu and click on the new set to add new flashcards to it. I added the ability for the user to create their own sets rather than revising from pre-made sets as my stakeholders wanted to be able to personalise their revision to their own needs. The data saved to an external file from this process will be required for the user to be able to use the other features of the revision app, because they will not have anything to revise if they do not create a set.

Choose a set of flashcards – each of the users created sets of flashcards (max of 5, possibly more) will be viewable from the main menu by the click of a button. Once the user has chosen a set by clicking on the button with its name on it, they will be able to do three things:

- **Revise the set** – this will be in the form of traditional flashcards; it will work by the program reading from the .csv file, and displaying the term on screen. There will be a button for the user to **‘flip’** the flashcard, which will then display the definition. I will most likely use a variable which changes between ‘term’ and ‘definition’ to implement this. The user can revise their set of flashcards via this method, which is similar to how other revision apps (discussed in analysis section), as well as real-life flashcards, work and this is what my stakeholders wanted. There will also be the option to **delete** the flashcard being viewed from this screen, also via a button. The **next and previous flashcard** to allow the user to navigate through their set will also be displayed here. These features will be implemented by using counter variables to track how far into the set the user is.
- **Edit set** – this will give the user the ability to add flashcards to the set as they feel is necessary. The changes they make will be written to the .csv file and this data will then be used when the user chooses to revise the amended set.
- **Take a quiz** – the user will be able to use this feature to test their knowledge on a particular set of material. The variables ‘score’ and ‘time’ taken to complete will be recorded and this data will be stored in an external file, ready to be used in the progress tracker. This was a commonly wanted feature among my stakeholders and so I am including it and integrating the results into the progress tracker. Sound effects may also be added when the user is taken to the summary screen after the quiz (such as a ‘ding’) to indicate

whether they passed (80%) or failed. This is more of an aesthetic touch rather than a requirement.

Progress Tracker – Data from the quizzes will be required for this feature – it will show a user's quiz scores in the form of raw numbers and also on a graph. This was a feature requested by all of my stakeholders as they wanted the ability to track their scores and see where they need to improve. All of this information will be in one place, making it easy to compare and deduce which subjects/areas the user is strong in, and the areas they need to improve in.

- **Post results to social media** – the app will have a shared Twitter page where users can post their progress and test scores along with other members of the app (designed for class use). This adds a sense of competition to the app which will add to the user's motivation to revise. I have decided to use Twitter as it is the social media platform that a couple of my stakeholders said they use regularly. I will be using the Twitter API, and tweets will follow the rough template "[username] scored [score] on a quiz about [set name]! Well done!". Data from the quizzes/progress tracker will be read from the external file for this feature.
- **Send email reminder** – this feature requires the users email address, which they would have entered when signing up to the app, and will send an email reminder via the press of a button. It is designed to ensure my stakeholders do not forget about their progress, and it encourages them to stay on top of their revision. Whilst this is not a scheduled reminder, it will still be in the users inbox when they next check their emails, which should still have the same effect. I will be using the Google API to achieve this, meaning unfortunately only gmail accounts will be able to benefit from this feature.

Systems Algorithms

Start Menu

```
layout = STARTMENU
if button == 'sign_up' THEN
    layout = SIGNUP
if button == 'log_in' THEN
    layout = LOGIN
if button == 'Exit' THEN
    break event loop
```

This feature changes the layout shown to the user via the press of a button, allowing the user to easily log in and sign up.

Create an account

```
valid_username = True
import hashlib
if button == ‘create_account’ THEN
    username = str(USERINPUT)
    password1 = str(USERINPUT)
    password_validation = str(USERINPUT)
    email = str(USERINPUT)
    OPEN FILE ‘userinfo.txt’, READ AND WRITE
    for i in range 1 to (num_of_accounts*4) THEN
        if f.read(line i) == username THEN
            UPDATE ELEMENT ‘error_message’ TO “Username already taken”
            valid_username = FALSE

    if valid_username == True THEN
        if password1 == password_validation THEN
            if len(email) > 11 AND email[len(email)-11 , len(email)-1] == ‘@gmail.com’ THEN
                password = hashlib.sha256(password1)
                num_of_sets = 0
                user_id += 1
                WRITE username TO ‘userinfo.txt’ on line[(num_of_accounts*4)+1]
                WRITE password TO ‘userinfo.txt’ on line[(num_of_accounts*4)+2]
                WRITE email TO ‘userinfo.txt’ on line[(num_of_accounts*4)+3]
                WRITE user_id TO ‘userinfo.txt’ on line[(num_of_accounts*4)+4]
                CLOSE FILE ‘userinfo.txt’
                CREATE FILE ‘{user_id}.csv’
                OPEN FILE ‘{user_id}.csv’, WRITE
                WRITE num_of_sets TO ‘{user_id}.csv’ in D1
                num_of_accounts += 1
                layout = MAINMENU
                break
            else THEN
                UPDATE ELEMENT ‘signup_error_message’ TO “Invalid email”

        else THEN
            UPDATE ELEMENT ‘signup_error_message’ TO “Incorrect password entry”
    PROCEDURE set_names()
    CLOSE FILE ‘{user_id}.csv’
```

This algorithm allows the user to textually input their desired username, password and email address into 3 input boxes, and when they confirm their details via the press of a button, their account is either created if their inputs are valid (their details are written to and stored in userinfo.txt), and if they aren’t, an error message is shown at the bottom of the display informing the user of the problem, which they can then fix and resubmit.

<u>Log in</u>

```
if button == ‘log_in’ THEN
    username = str(USERINPUT)
    password = str(USERINPUT)
    OPEN FILE ‘userinfo.txt’, READ
    for i in range 1 to (num_of_accounts*4) THEN
        if f.read(line i) == username THEN
            if f.read(line i +1) == hashlib.sha256(password) THEN
```

```
                    UPDATE ELEMENT 'login_error_message' TO "Log-in Successful"
                    email = f.read(line i +2)
                    user_id = f.read(line i +3)
                    CLOSE FILE 'userinfo.txt', READ
                    OPEN FILE '{user_id}.csv', READ
                    num_of_sets = D1 in '{user_id}.csv'
                    layout = MAINMENU
                    break
                else THEN
                    UPDATE ELEMENT 'login_error_message' TO "Incorrect Password"
        PROCEDURE set_names(user_id)
        CLOSE FILE '{user_id}.csv'
```

This feature allows the user to log in to an already existing account via typing their username and password into the textual input boxes and validating their details via the press of a button. The details are searched for in userinfo.txt and if they match, the user Is taken onto their account, and if they don't the user is notified via an error message at the bottom of the display.

PROCEDURE set_names()

```
define PROCEDURE set_names(user_id)
        OPEN FILE '{user_id}.csv'
        UPDATE ELEMENT 'set1' TO "Set 1 – {f.read[E1]}"
        UPDATE ELEMENT 'set2' TO "Set 2 – {f.read[E2]}"
        UPDATE ELEMENT 'set3' TO "Set 3 – {f.read[E3]}"
        UPDATE ELEMENT 'set4' TO "Set 4 – {f.read[E4]}"
        UPDATE ELEMENT 'set5' TO "Set 5 – {f.read[E5]}"
        CLOSE FILE '{user_id}.csv'
```

This procedure simply updates the text on the set buttons on the users main menu to the names of the users sets, which are read from their personal .csv file.

Main Menu

```
if button == 'log_out' THEN
        PROCEDURE LOGOUT()
if button == 'manage_account' THEN
        layout = ACCOUNT
if button == 'progress_tracker' THEN
        layout = PROGRESS
        … shown later on
if button == 'new_set' THEN
        layout = SETNAME
if button == 'my_sets' THEN
        layout = SETS
if button == 'Exit' THEN
```

```
    break event loop
```

The main menu is the 'hub' of the application, as it is where the user can access all of the main features of the app via the press of a button. This algorithm simply updates the display whenever a button is pressed to the relevant layout, or shuts the app down by breaking the event loop if the 'Exit' button is pressed.

Log Out

```
define PRODECURE LOGOUT()
    UPDATE ELEMENT 'signup_error_message' TO ""
    UPDATE ELEMENT 'login_error_message' TO ""
    layout = STARTMENU
```

This algorithms shows how when the 'log_out' button is pressed, a procedure is ran which takes the user back to the start menu and resets the error messages ready for when the next user uses the application.

Manage Account

```
if button == 'update_username' THEN
    new_username = str(USERINPUT)
    OPEN FILE 'userinfo.txt', READ
    valid_username = True
    for i in range 1 to (num_of_accounts*4) THEN
        if f.read(line i) == new_username THEN
            UPDATE ELEMENT 'error_message' TO "Username already taken"
            valid_username = False
            break
    if valid_username == True
        CREATE FILE 'new.txt', WRITE (created temporarily for file transfer, then renamed)
        for i in range 1 to (num_of_accounts*4) THEN
            if line(i) IN 'userinfo.txt' == username THEN
                line(i) in 'new.txt' = new_username
            else THEN
                line(i) IN 'new.txt' = line(i) IN 'userinfo.txt'
        DELETE FILE 'userinfo.txt'
        RENAME FILE 'new.txt' to 'userinfo.txt'
        CLOSE FILE 'userinfo.txt'

if button == 'update_password' THEN
    new_password = hashlib.sha256(str(USERINPUT))
    OPEN FILE 'userinfo.txt', READ
    CREATE FILE 'new.txt', WRITE (created temporarily for file transfer, then renamed)
    for i in range 1 to (num_of_accounts*4) THEN
        if line(i) IN 'userinfo.txt' == username THEN
            line(i) IN 'new.txt' = line(i) IN 'userinfo.txt'
            line(i+1) in 'new.txt' = new_password
            i+=2
        else THEN
```

```
                line(i) IN ‘new.txt’ = line(i) IN ‘userinfo.txt’
        DELETE FILE ‘userinfo.txt’
        RENAME FILE ‘new.txt’ to ‘userinfo.txt’
        CLOSE FILE ‘userinfo.txt’

if button == ‘update_email’ THEN
        new_email = str(USERINPUT)
        OPEN FILE ‘userinfo.txt’, READ
        CREATE FILE ‘new.txt’, WRITE (created temporarily for file transfer, then renamed)
        if len(new_email) > 11 AND new_email[len(email)-11 , len(email)-1] == ‘@gmail.com’ THEN
                for i in range 1 to (num_of_accounts*4) THEN
                        if line(i) IN ‘userinfo.txt’ == username THEN
                                line(i) IN ‘new.txt’ = line(i) IN ‘userinfo.txt’
                                line(i+1) in ‘new.txt’ = line(i+1) IN ‘userinfo.txt’
                                line(i+2) in ‘new.txt’ = new_email
                                i+=3
                        else THEN
                                line(i) IN ‘new.txt’ = line(i) IN ‘userinfo.txt’
                DELETE FILE ‘userinfo.txt’
                RENAME FILE ‘new.txt’ to ‘userinfo.txt’
                CLOSE FILE ‘userinfo.txt’
        else THEN
                UPDATE ELEMENT ‘update_account_error’ TO “Invalid email”
```

This algorithm shows the different processes ran if the user decides to update either their username, password or email. The new username, password or email is validated to ensure that it is correct/does not already exist, and if it is valid, userinfo.txt is rewritten to update the new piece of information. If the data entered is invalid, then an error message is displayed to the user at the bottom of the display.

Delete Account

```
new_line = 1
define PROCEDURE delete_account(username)
        OPEN FILE ‘userinfo.txt’, READ
        CREATE FILE ‘new.txt’, WRITE (created temporarily for file transfer, then renamed)
        for i in range 1 to (num_of_accounts*4) THEN
                if line(i) IN ‘userinfo.txt’ == username THEN
                        i+=4
                else THEN
                        line(new_line) IN ‘new.txt’ = line(i) IN ‘userinfo.txt’
                        new_line += 1
        DELETE FILE ‘{user_id}.csv’
        DELETE FILE ‘userinfo.txt’
        RENAME FILE ‘new.txt’ to ‘userinfo.txt’
        CLOSE FILE ‘userinfo.txt’
```

This algorithm re-writes userinfo.txt to include everything in the original file apart from the information associated with the deleted account, and deletes the users personal .csv file. By *doing this, no data has to be stored from deleted accounts (thus reducing the storage space needed by the app).*

Create a new set of flashcards

```
if button == 'new_set' THEN
    layout = SETNAME

if button == 'confirm_name' THEN
    if num_of_sets == 5 THEN
        UPDATE ELEMENT 'create_error_message' TO "Max number of sets reached"
    else THEN
        OPEN FILE '{user_id}.csv', WRITE
        set_name = str(USERINPUT)
        num_of_sets += 1
        E[num_of_sets] in '{user_id}.csv' = set_name
        D1 in '{user_id}.csv' = num_of_sets
        CLOSE FILE '{user_id}.csv'
        PROCEDURE set_names(user_id)
        layout = FLASHCARDSET
```

This algorithm allows the user to enter the name of a new set they wish to create, which is then written to the users .csv file as long as they have not already reached their 5 set limit. The buttons on the 'My Sets' page are then updated to show the name of this new set.

Choose a set of flashcards

```
if button == 'set1' THEN
    set = 1
    layout = SETMENU
if button == 'set2' THEN
    set = 2
    layout = SETMENU
if button == 'set3' THEN
    set = 3
    layout = SETMENU
if button == 'set4' THEN
    set = 4
    layout = SETMENU
if button == 'set5' THEN
    set = 5
    layout = SETMENU
```

This algorithm changes the layout shown to the user via the press of a button, and updating the 'set' variable to the relevant set so that the program knows which set to search for in the .csv file.

Add flashcards

```
if button = 'new_flashcard' THEN
    layout = NEWFLASHCARD

if button = 'add' THEN
    term = str(USERINPUT)
    definition = str(USERINPUT)
    num_of_flashcards += 1
    OPEN([{user_id}.csv] to append) = user_file
    write = csv.writer[user_file]
    write.addnewrow(term, definition, set)
    CLOSE FILE '{user_id}.csv'
    CLEAR INPUTS 'term' AND 'definition'
```

This algorithm takes 2 user inputs as strings and stores them in the users .csv file along with the relevant set number. This is so these flashcards can be read from the file and displayed on the screen for other features of the app.

Revise flashcards

```
if button == 'revise' THEN
    layout = FLASHCARD
    flashcard_number = 1
    OPEN FILE '{user_id}.csv', READ
    PROCEDURE next_flashcard(flashcard_number)
    CLOSE FILE '{user_id}.csv'

define PROCEDURE next_flashcard(flashcard_number)
    for flashcard_number in range 1 to (num_of_flashcards) THEN
        if position C[flashcard_number] in '{user_id}.csv' == set THEN
            term = A[flashcard_number] in '{user_id}.csv'
            definition = B[flashcard_number] in '{user_id}.csv'
            UPDATE ELEMENT 'flashcard_text' TO term
            flashcard_number += 1
            break
        else THEN
            flashcard_number += 1

if button == 'flip' THEN
    UPDATE ELEMENT 'flashcard_text' TO definition

if button == 'next' THEN
    OPEN FILE '{user_id}.csv', READ
    PROCEDURE next_flashcard(flashcard_number)
    CLOSE FILE '{user_id}.csv'
```

```
if button == 'previous' THEN
    flashcard_number -= 2
    OPEN FILE '{user_id}.csv', READ
    for flashcard_number in range 1 to (num_of_flashcards) THEN
        if position C[flashcard_number] in '{user_id}.csv' == set THEN
            term = A[flashcard_number] in '{user_id}.csv'
            definition = B[flashcard_number] in '{user_id}.csv'
            UPDATE ELEMENT 'flashcard_text' TO term
            flashcard_number -= 1
            break
        else THEN
            flashcard_number -= 1
    CLOSE FILE '{user_id}.csv'
```

This algorithm sequentially searches through the users .csv file to find flashcards with the relevant set number associated to them, and when one is found, the search is paused and the flashcard is displayed on screen. The term and definition are stored as variables so that when the flashcard is 'flipped' the definition is shown instead of the term (and vice versa). When the user decides to move on to the next flashcard, the search is resumed and this whole process goes on until the algorithm has searched all of the users created flashcards in the .csv file. The 'previous' button works the same way except it searches up the file rather than down it.

Delete flashcards

```
if button == 'delete' THEN
    OPEN FILE '{user_id}.csv', READ + WRITE
    position A[flashcard_number] = " "
    position B[flashcard_number] = " "
    position C[flashcard_number] = " "
    PROCEDURE next_flashcard(flashcard_number)
    CLOSE FILE '{user_id}.csv'
```

This algorithm simply deletes a flashcard (term, definition, and its set number) from the users .csv file, and then updates to the next flashcard in the same way as stated in the previous algorithm.

Quiz

```
if button == 'quiz' THEN
    layout = QUIZ
    start_time recorded
    num_of_questions = 0
    flashcard_number = 1
    PROCEDURE update_question()

define PROCEDURE update_question()
    random_answers = []
    random_answer_slot = [0,1,2,3]
    OPEN FILE '{user_id}.csv', READ
    for i in range 1 to num_of_flashcards THEN
        if position C[flashcard_number] in '{user_id}.csv' == set THEN
```

```
                    term = A[flashcard_number] in '{user_id}.csv'
                    definition = B[flashcard_number] in '{user_id}.csv'
                    APPEND flashcard_number to random_answers
                    UPDATE ELEMENT 'question' TO definition
                    random_slot = random.randint(0,len(random_answer_slot)-1)
                    correct_answer = random_answer_slot[random_slot]
                    UPDATE ELEMENT 'answer{correct_answer}' TO term
                    REMOVE random_answer_slot[correct_answer]
                    while len(random_answers) < 4 THEN
                        random_ans = random.randint(1,num_of_flashcards)
                        if random_ans NOT IN random_answers THEN
                            APPEND random_ans TO random_answers
                            random_slot = random_answer_slot[random.randint(0,len(random_answer_slot)-1)]
                            UPDATE ELEMENT 'answer{random_slot}' TO B[random_ans} in '{user_id}.csv'
                            REMOVE random_answer_slot[random_slot]
                    flashcard_number += 1
                    num_of_questions += 1
                    END PROCEDURE
                else THEN
                    flashcard_number += 1
        end_time recorded
        time_taken = start_time – end_time
        percentage = (score/num_of_questions) x100
        num_of_quizzes += 1
        WRITE num_of_quizzes TO D1 in '{user_id}.csv'
        WRITE time_taken TO F[num_of_quizzes] in '{user_id}.csv'
        WRITE percentage TO G[num_of_quizzes] in '{user_id}.csv'
        WRITE set TO H[num_of_quizzes] in '{user_id}.csv'
        UPDATE ELEMENT 'SCORE' TO score
        UPDATE ELEMENT 'TIME' TO time_taken
        layout = QUIZRESULTS
        CLOSE FILE '{user_id}.csv'
        if percentage > 80 THEN
            play 'ding.mp4'

if button == 'answer0' THEN
        chosen_answer = 0
        if chosen_answer == correct_answer THEN
            score += 1
        PROCEDURE update_question()

if button == 'answer1' THEN
        chosen_answer = 1
        if chosen_answer == correct_answer THEN
            score += 1
        PROCEDURE update_question()

if button == 'answer2' THEN
        chosen_answer = 2
        if chosen_answer == correct_answer THEN
            score += 1
        PROCEDURE update_question()

if button == 'answer3' THEN
        chosen_answer = 3
        if chosen_answer == correct_answer THEN
            score += 1
        PROCEDURE update_question()
```

This algorithm works in a similar way to the revise feature in the sense that it sequentially searches through the users .csv file to find the next term and definition for the chosen set, and then it displays the term/question at the top of the display, and places the correct definition/answer into one of the 4 multiple choice buttons. The algorithm then fills the other 3 buttons with 3 other randomly chosen definitions/answers from the .csv file. If the correct answer is chosen (via a button press) by the user, the score variable increments by one, and whenever any answer is chosen, the display is updated to show the next question and 4 possible answers. This process repeats until the algorithm has gone through all of the *flashcards in the users .csv file, at which point a summary screen is shown with the users final score, percentage and time taken.*

Progress Tracker

```
import matplotlib.pyplot as plt
if button == ‘progress_tracker’ THEN
        layout = PROGRESS
        OPEN FILE ‘{user_id}.csv’, READ
        UPDATE ELEMENT ‘result1’ TO “Set [H[num_of_quizzes]] – [G[num_of_quizzes]] in [F[num_of_quizzes]]”
        UPDATE ELEMENT ‘result2’ TO “Set [H[num_of_quizzes -1]] – [G[num_of_quizzes -1]] in [F[num_of_quizzes -1]]”
        UPDATE ELEMENT ‘result3’ TO “Set [H[num_of_quizzes -2]] – [G[num_of_quizzes -2]] in [F[num_of_quizzes -2]]”
        UPDATE ELEMENT ‘result4’ TO “Set [H[num_of_quizzes -3]] – [G[num_of_quizzes -3]] in [F[num_of_quizzes -3]]”
        UPDATE ELEMENT ‘result5’ TO “Set [H[num_of_quizzes -4]] – [G[num_of_quizzes -4]] in [F[num_of_quizzes -4]]”
        x = [1,2,3,4,5]
        y = [G[num_of_quizzes], G[num_of_quizzes -1], G[num_of_quizzes -2], G[num_of_quizzes -3], G[num_of_quizzes -4]]
        plt.plot(x, y)
        plt.xlabel(‘Attempt (most recent to least recent)’)
        plt.ylabel(‘Percentage’)
        plt.title(‘Percentage Graph’)
        UPDATE ELEMENT ‘graph’ TO plt.show()
        CLOSE FILE ‘{user_id}.csv’

import tweepy
if button == ‘twitter’ THEN
        OPEN FILE ‘{user_id}.csv’, READ
        Authenticate consumer_key and consumer_secret_key
        Authenticate access_token and secret_access_token
        Create API Object
        API.update_status(show 5 most recent percentages in same method as above)
        CLOSE FILE ‘{user_id}.csv’

if button == ‘email_reminder’ THEN
        try:
                sender = revisionapp@gmail.com
                recipient = email
                subject = “REMINDER TO CONTINUE REVISION!”
                message = “Keep up the good work!”
                SEND subject WITH message FROM sender TO recipient
        except:
                UPDATE ELEMENT ‘progress_tracker_error_message’ TO “An error occurred”
```

This algorithm takes the user to the progress tracker screen. When the button is pressed, the features of the progress tracker are updated and displayed. 5 raw number results will be shown on the left along with their respective set number, and on the right there will be a graph created and displayed of percentage against attempt. There will also be a button which sends a tweet of the users 5 most recent scores to the applications Twitter page, as well as a button to send an email reminder to the users email address via the Google API.

Back

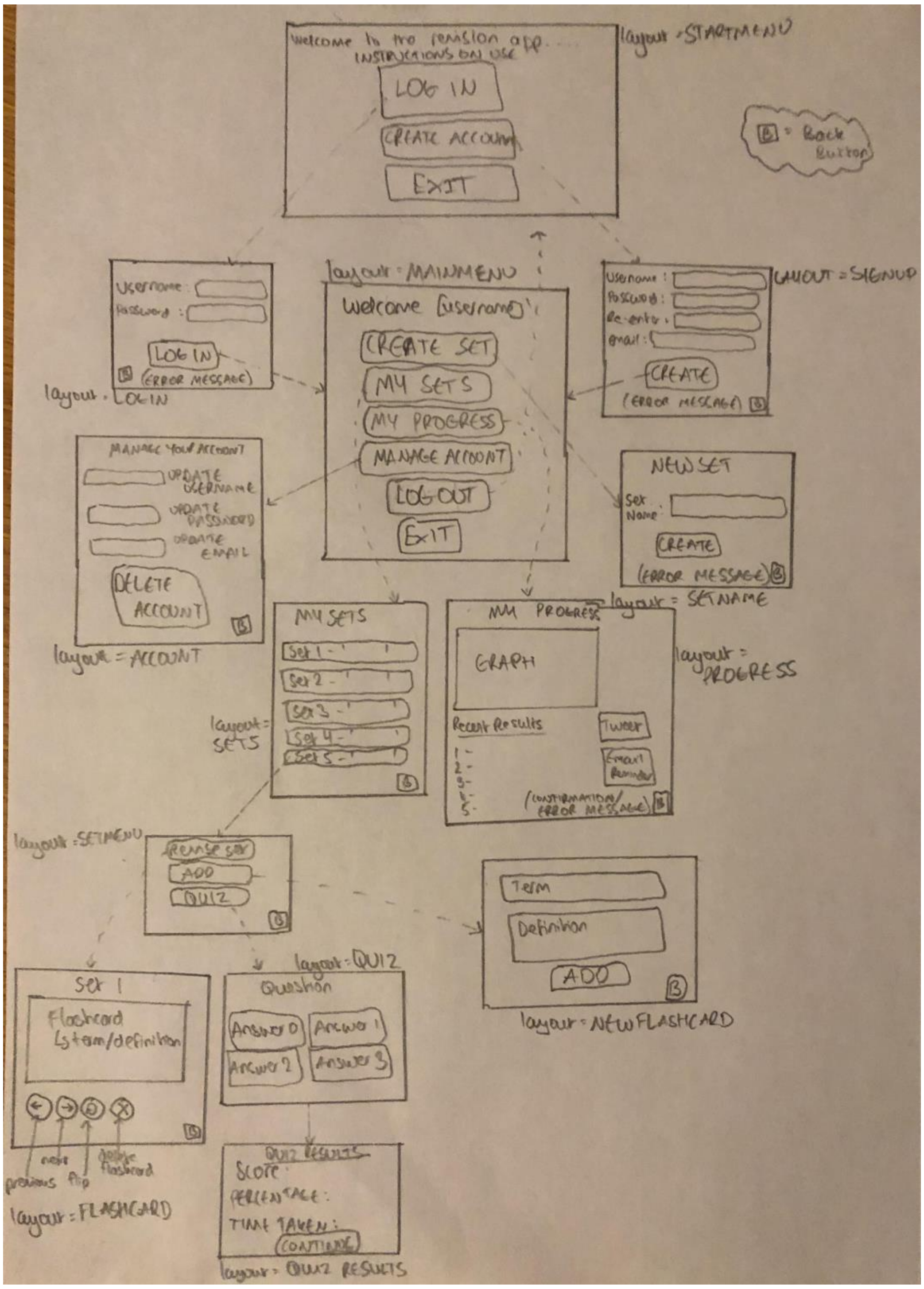
Throughout my application, there will be a back button in the bottom corner of the screen with will simply update the layout to the layout that was previously visible to the user.

Algorithm Summary

All of these algorithms link together and cover all areas and features of my program, allowing the user to use every feature as easily and effectively as possible, and navigate

between areas of my solution as logically and fluently as possible in a way that will allow the user to gain a fast understanding of how to use the app.
Therefore, all the algorithms form a complete solution because they correspond to and link together in the same way as the systems diagram.

Usability features



Layout Diagram

I have designed my application to be as simple to understand and use as possible. I made this decision with my stakeholders in mind – they asked for a simple and quick-to-use app that would allow them to revise effectively using a wide range of features in as little time as possible. To allow this, I have reduced the number of layouts I am using to approximately one layout per feature to reduce the time the user has to spend navigating in between features and around the app. This also lowers the storage space needed by the program to as small an amount as possible, as storage space will need to be saved for the .txt and .csv files that will store all of the users data and information.

The only sources of user input in my application will be via buttons and textual inputs, linking to how the main hardware requirements for my solution are a mouse and keyboard. This also means that the user can run the application on almost any device, as all devices nowadays allow the user to press buttons (either via a mouse click or touchscreen) and input text (either via keyboard hardware or a digital keyboard). The log-in/sign-up features will require textual inputs via a keyboard so that the user can enter their username, password etc, and textual inputs will also be required when the user creates flashcards and updates account details. However, apart from that, most of the user inputs are buttons (as seen primarily in the start menu and main menu), as this is a quick way of navigating around the app. I also opted for multiple choice answers involving buttons rather than textual inputs for the quiz feature, as I felt that again pressing a button to choose your answer is a lot quicker than having to type an answer in to an input box whilst being no less effective in terms of revision.

The 'Exit' buttons will be red in colour as this makes them stand out and is a common convention among most apps nowadays, not just revision apps. The error messages will also be in red text for this reason. Other text around the app, such as the instructions on the start menu, will be in bold, and will be as large as possible in the window so that the user does not have to struggle to read the text. The text itself will be as brief and short as possible, once again to reduce the amount of time the user would have to spend reading it.

Key Variables

Variable Name	**Data Type**	**Purpose**
username	String	To store the current users plaintext username. This may be changed if the user decides to change/update their username via the 'Manage Account' feature.
password	String	To store the users **hashed** password (increased security). This may be changed if the user decides to change/update their password via the 'Manage Account' feature.
email	String	To store the users email address. This may be changed if the user decides to change/update their email via the 'Manage Account' feature, and it will also be used as the address that email reminders are sent to.
user_id	Integer	This is the users unique user ID, and is used to identify their own personal .csv file.
num_of_sets	Integer	Stores the number of sets a user has created, and this value is stored in their own .csv file and updated whenever the variable changes value. This

		variable is used to ensure the user does not exceed their 5 set limit.
num_of_accounts	Integer	Stores the number of accounts on the application, and it is used to stop the search for a username/password/email in userinfo.txt becoming infinite (creates a limit in the for loop)
valid_username	Boolean	Used to confirm whether a username is valid when entered before being written to the .csv file (i.e. that it has not already been used by another user) whenever a user creates a new account or updates their account details.
new_username	String	Stores the new username a user enters into the relevant input box when updating their account details, and the contents of this variable are then written to the .csv file if it is valid.
new_password	String	Stores the new **hashed** password a user enters into the relevant input box when updating their account details, and the contents of this variable are then written to the .csv file if it is valid.
new_email	String	Stores the new email a user enters into the relevant input box when updating their account details, and the contents of this variable are then written to the .csv file if it is valid.
set	Integer	Stores the integer value of the set that the user is currently revising from/editing/adding to/taking a quiz from. This is so that each term and definition the user adds to a set is stored in the .csv file along with the set they are created for. This is so that when the program is reading back from the file, it can validate which flashcards should be shown to the user for a particular set by sequentially searching for that set number in the database, and when a row with that set number is found, the corresponding term and definition are allocated to the term and definition variables (below).
term	String	Stores the term of flashcard to be written to the .csv file or stores the term which has been read from the .csv file which will be displayed to the user.
definition	String	Stores the definition of flashcard to be written to the .csv file or stores the definition which has been read from the .csv file which will be displayed to the user.
flashcard_number	Integer	Stores the row number from the .csv file of the term/definition which is being searched to check whether it is in a particular set.

num_of_questions	Integer	This stores the total number of questions the user has answered in a quiz.
score	Integer	Stores the number of questions a user has gotten correct in a quiz. This can then be used along with the num_of_questions variable to calculate the percentage.
random_answers	Integer array	Stores the row numbers of the correct answer to a question in a quiz along with the other 3 randomly chosen answers. This is to ensure the same answer does not appear twice in a multiple choice question, as if the row number of a randomly chosen answer is already in the array, it is rejected and another random number is generated.
random_answers_slot	Integer array	Stores 4 integers – 0,1,2,3 – and each randomly chosen answer to a question is randomly assigned one of these numbers (which is then removed from the array), which determines which of the 4 multiple choice boxes the answer is placed in. This ensures that correct answers are not always in the same box, and helps make sure that the quiz is never repetitive and so properly tests the users knowledge.
random_slot	Integer	Generates a random integer between 1 the length of the random_answers_slot array, which is used as the index position of an integer in the list which is then used to determine which multiple choice box an answer will be in.
correct_answer	Integer	Holds the integer of the answer box which holds the correct answer, so that if this answer is chosen, the users score is incremented by 1.
random_ans	Integer	Holds the row number of a randomly chosen definition from the users .csv file. The definition on this row is then used as one of the four multiple choice question answers.
start_time	Time	Records the time a quiz was started at.
end_time	Time	Records the time a quiz was completed.
time_taken	Time	Time taken to complete a quiz (end_time – start_time).
percentage	Integer	Holds the percentage a user achieves in a quiz.
num_of_quizzes	Integer	The number of quizzes a user has completed is stored in the users .csv file, and this variable is used to update this value – if the number of quizzes a user has completed changes, the original value is read from the .csv file and stored as this variable, the variable is incremented by 1, and then the value the variable holds is then written back to the .csv file. This is needed for the progress tracker.

x	Integer array	Used to store the values that will go on the x axis (quiz attempt number) of the progress tracker graph.
y	Integer array	Used to store the values that will go on the y axis (percentage) of the progress tracker graph.

Additional Files

File Name	**Use**	**File Size**
userinfo.txt	Stores all of the users' information for their accounts (usernames, hashed passwords, emails, IDs etc)	PER USER assuming: • max 20 character long username – 20 bytes • password - fixed 64 bit SHA 256 value • email – max 30 bytes • user ID – max 2 bytes - TOTAL = 116bytes - 30 USERS (class size) = around 3500bytes = **3.5KB**
A .csv file for each user named '{user_id}.csv' e.g. user with ID 1 would have a .csv file named '1.csv'	Stores all of a users individual flashcards, sets, quiz scores and results, number of quizzes/flashcards etc	PER USER FILE: • max 5 sets, assuming each is 20 characters long – 100 bytes • number_of_flashcards variable – 2 bytes max • 3 columns for terms, definitions and corresponding set numbers, assuming 50 flashcards, and each term/definition being an average of 20 characters = 3000 bytes • 3 columns for test percentages, times and sets, assuming 10 quiz results are stored = 60 bytes TOTAL = **3162 bytes = 3.2KB**
ding.mp3	Sound effect for when a user passes a quiz.	**47KB** (stated in file manager)

Iterative development

This section will include test tables for each algorithm previously written to ensure my solution is robust and effective.

Start Menu – 1st Iteration

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
1.1.1	Selection of option via 'Sign_up' button	Mouse click	Nothing will happen as the functionality of the button has not been implemented yet, and the SIGNUP layout has not been defined.	Checking whether the buttons appear on the screen as intended and are pressable.	
1.1.2	Selection of option via 'log_in' button	Mouse click	Nothing will happen as the functionality of the button has not been implemented yet, and the LOGIN layout has not been defined.	Checking whether the buttons appear on the screen as intended and are pressable.	
1.1.3	Selection of option via 'Exit' button	Mouse click	Nothing will happen as the functionality of the button has not been implemented yet.	Checking whether the buttons appear on the screen as intended and are pressable.	

Start Menu – 2nd iteration

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
1.2.1	Selection of option via 'Sign_up' button	Mouse click	Layout will be updated to 'SIGNUP'	To check whether the sign-up screen is accessible via a button press from the start menu.	
1.2.2	Selection of option via 'log_in' button	Mouse click	Layout will be updated to 'LOGIN'	To check whether the log-in screen is accessible via a button press from the start menu.	
1.2.3	Selection of option via 'Exit' button	Mouse click	App will close as event loop is broken.	To check whether the Exit button functions as intended from the start menu.	

Sign up – 1st iteration

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
2.1.1	Entering username, password and email into relevant fields	String 'username123' String 'password123' String 'bradleymak2003 @gmail.com' **VALID DATA**	App will allow user to enter these strings into the relevant input boxes.	Checking whether the user can enter a string into the textual input boxes that should appear on screen.	

			The password should be dotted.		
2.1.2	Selection of button to confirm account details	Mouse click	Nothing, as functionality of button has not been implemented yet.	To check whether the button on screen appears in the correct place and allows user interaction.	

Sign up – 2nd iteration

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
2.2.1	Entering username, password (twice) and email into relevant fields	String 'username123' String 'password123' String 'password123' String 'bradleymak2003 @gmail.com' **VALID DATA**	App will allow user to enter these strings into the relevant input boxes. The password should be dotted.	Checking whether the user can enter a string into the textual input boxes that should appear on screen.	
2.2.2	Selection of button to confirm account details	Mouse click	Entry fields will clear and strings will be written to 'userinfo.txt', but app will remain on the same screen.	To check whether details entered into the input boxes by the user are stored to an external file via a button press so they can be referenced at any later date.	

Sign up – 3rd iteration

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
2.3.1	Entering username, password and email into relevant fields	String 'username123' String 'password123' String 'password123' String 'bradleymak2003 @gmail.com' **VALID DATA**	App will allow user to enter these strings into the relevant input boxes. The password should be dotted.	Checking whether the user can enter a string into the textual input boxes that should appear on screen.	
2.3.2	Selection of button to confirm account details	Mouse click	Entry fields will clear and strings will be written to 'userinfo.txt', and layout will be updated to MAIN MENU.	To check whether the confirm account button takes the user to the main menu once the details they have entered have been validated and stored.	

Sign up – 4th iteration

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
2.4.1	Entering duplicate username and invalid email	String ‘username123’ String ‘password123’ String ‘password123’ String ‘bradleymak2003@gml .com’ **INVALID DATA**	App will allow user to enter these strings into the relevant input boxes. The password should be dotted.	Checking whether the user can enter a string into the textual input boxes that should appear on screen.	
2.4.2	Selection of button to confirm account details	Mouse click	Error message will appear at bottom of screen stating invalid username at first (as this username was used in a previous test), so change username to ‘111’ and try again, now error message should say invalid email and layout will remain the same.	To check whether the validation algorithm detects an invalid email and displays the issue to the user (and does not proceed to the main menu).	

Sign up – 5th iteration

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
2.5.1	Entering invalid email	String ‘123’ String ‘password123’ String ‘password’ String ‘bradleymak2003 @gmail.com’ **INVALID DATA**	App will allow user to enter these strings into the relevant input boxes. The password should be dotted.	Checking whether the user can enter a string into the textual input boxes that should appear on screen.	
2.5.2	Selection of button to confirm account details	Mouse click	Error message will appear at bottom of screen stating that the user mistyped their password (i.e. the 2 password input boxes do not match).	To check whether the validation algorithm detects an invalid password re-entry and displays the issue to the user (and does not proceed to the main menu).	

Sign up – 6th iteration

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result

2.5.1	Missing field	String '12345' String 'password123' String 'password123' String ' ' **INVALID DATA**	App will allow user to enter these strings into the relevant input boxes. The password should be dotted.	Checking whether the user can enter a string into the textual input boxes that should appear on screen.	
2.5.2	Selection of button to confirm account details	Mouse click	Error message will appear at bottom of screen stating that all fields need to be entered (as email is missing)	To check whether the validation algorithm detects missing inputs, as all inputs are required.	

Log in – 1st iteration

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
3.1.1	Entering username and password into relevant fields	String 'username123' String 'password123' **VALID DATA**	App will allow user to enter these strings into the relevant input boxes	Checking whether the user can enter a string into the textual input boxes that should appear on screen.	
3.1.2	Selection of button to confirm account details	Mouse click	Nothing, as functionality of button has not been implemented yet.	To check whether the confirm button on screen appears in the correct place and allows user interaction.	

Log in – 2nd iteration

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
3.2.1	Entering username and password into relevant fields	String 'username123' String 'password123' **VALID DATA**	App will allow user to enter these strings into the relevant input boxes	Checking whether the user can enter a string into the textual input boxes that should appear on screen.	
3.2.2	Selection of button to confirm account details	Mouse click	Layout will be updated to MAINMENU and user will be	To check whether the button takes the user to the main menu of the app if their log-in details	

			taken to the main menu.	are valid and correct.	

Log in – 3rd iteration

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
3.3.1	Entering incorrect password	String 'username123' String 'password1' **INVALID DATA**	App will allow user to enter these strings into the relevant input boxes	Checking whether the user can enter a string into the textual input boxes that should appear on screen.	
3.3.2	Selection of button to confirm account details	Mouse click	Error message will appear at bottom of screen stating invalid password, and layout will remain the same.	To check whether the confirm account button takes the user to the main menu if the details they enter are incorrect.	
3.3.3	Entering username that doesn't exist	String 'username1' String 'password123' **INVALID DATA**	App will allow user to enter these strings into the relevant input boxes	Checking whether the user can enter a string into the textual input boxes that should appear on screen.	
3.3.4	Selection of button to confirm account details	Mouse click	Error message will appear at bottom of screen stating that the username doesn't exist.	To check whether the confirm account button takes the user to the main menu if the details they enter are incorrect.	

Main Menu – 1st iteration

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
4.1.1	Selection of option via 'new_set' button	Mouse click	Nothing will happen as the functionality of the button has not been implemented yet, and the SETNAME layout has not been defined.	Checking whether the button appears in the correct format on screen and can be interacted with by the user.	

4.1.2	Selection of option via ‘my_sets’ button	Mouse click	Nothing will happen as the functionality of the button has not been implemented yet, and the SETS layout has not been defined.	Checking whether the button appears in the correct format on screen and can be interacted with by the user.	
4.1.3	Selection of option via ‘progress_tracker’ button	Mouse click	Nothing will happen as the functionality of the button has not been implemented yet, and the PROGRESS layout has not been defined.	Checking whether the button appears in the correct format on screen and can be interacted with by the user.	
4.1.4	Selection of option via ‘manage_account’ button	Mouse click	Nothing will happen as the functionality of the button has not been implemented yet, and the ACCOUNT layout has not been defined.	Checking whether the button appears in the correct format on screen and can be interacted with by the user.	
4.1.5	Selection of option via ‘log_out’ button	Mouse click	Nothing will happen as the functionality of the button has not been implemented yet.	Checking whether the button appears in the correct format on screen and can be interacted with by the user.	
4.1.6	Selection of option via ‘Exit’ button	Mouse click	Nothing will happen as the functionality of the button has not been implemented yet.	Checking whether the button appears in the correct format on screen and can be interacted with by the user.	

Main Menu – 2nd iteration

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
4.2.1	Selection of option via ‘new_set’ button	Mouse click	Layout will be updated to SETNAME, so the user will be taken to a different screen.	Checking whether the button functions properly (i.e. updates the layout to the corresponding option)	
4.2.2	Selection of option via ‘my_sets’ button	Mouse click	Layout will be updated to SETS, so the user will be taken to a different screen.	Checking whether the button functions properly (i.e. updates the layout to the corresponding option)	
4.2.3	Selection of option via	Mouse click	Layout will be updated to	Checking whether the button	

	‘progress_tracker’ button		PROGRESS, so the user will be taken to a different screen.	functions properly (i.e. updates the layout to the corresponding option)	
4.2.4	Selection of option via ‘manage_account’ button	Mouse click	Layout will be updated to ACCOUNT, so the user will be taken to a different screen.	Checking whether the button functions properly (i.e. updates the layout to the corresponding option)	
4.2.5	Selection of option via ‘log_out’ button	Mouse click	User will be logged out and taken back to the start menu (layout is updated to STARTMENU)	Checking whether the button functions properly (i.e. updates the layout to the corresponding option)	
4.2.6	Selection of option via ‘Exit’ button	Mouse click	App will close as event loop is broken. User will be automatically logged out.	Checking whether the button functions properly (i.e. updates the layout to the corresponding option)	

Creating a new set of flashcards – 1st iteration

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
5.1.1	Entering name of set into textual input boxes	String ‘Data Structures’ **VALID DATA**	App will allow user to enter these strings into the relevant input boxes	Checking whether the user can enter a string into the textual input boxes that should appear on screen (there is no invalid string for a set name – it can be anything).	
5.1.2	Selection of button to create set with the given name	Mouse click	Nothing will happen as the functionality of the button has not been implemented yet.	To check whether the confirm button on screen appears in the correct place and allows user interaction.	

Creating a new set of flashcards – 2nd iteration

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
5.2.1	Entering name of set into textual input boxes	String ‘Data Structures’ **VALID DATA**	App will allow user to enter these strings into the relevant input boxes	Checking whether the user can enter a string into the textual input boxes that should appear on screen	

				(there is no invalid string for a set name – it can be anything).	
5.2.2	Selection of button to create set with the given name	Mouse click	Name of set will be written to the users personal .csv file, and the relevant button on the choose set layout will be updated to show the new set name.	To check whether the button correctly creates a set for the user with the given name, stores the name in the users .csv file, updates the button text on the 'MYSETS' layout, and updates the relevant variables (i.e. num_of_sets).	

Adding flashcards – 1st iteration

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
6.1.1	Entering term and definition into textual input boxes	String 'Static' String 'Size cannot change during runtime' **VALID DATA**	App will allow user to enter these strings into the relevant input boxes	To check the user can enter characters into the two textual input boxes.	
6.1.2	Selection of button to add flashcard to set	Mouse click	Nothing will happen as the functionality of the button has not been implemented yet.	To check the 'add' button appears correctly formatted on the display and can be interacted with by the user.	

Adding flashcards – 2nd iteration

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
6.2.1	Entering term and definition into textual input boxes	String 'Static' String 'Size cannot change during runtime' **VALID DATA**	App will allow user to enter these strings into the relevant input boxes	To check the user can enter characters into the two textual input boxes.	
6.2.2	Selection of button to add flashcard to set	Mouse click	Term and definition will be written to the relative positions in the users personal .csv file, and the term and definition textual input boxes will be cleared.	To check the 'add' button writes the term and definition to the correct positions in the users .csv file when pressed.	

Revise flashcards – 1st iteration

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
7.1.1	Flip flashcard via button press	Mouse click	Nothing will happen as the functionality of the button has not been implemented yet.	To check whether the button appears in the correct position on screen and can be interacted with by the user.	
7.1.2	Viewing the next flashcard via a button press	Mouse click	Nothing will happen as the functionality of the button has not been implemented yet.	To check whether the button appears in the correct position on screen and can be interacted with by the user.	
7.1.3	Viewing the previous flashcard via a button press	Mouse click	Nothing will happen as the functionality of the button has not been implemented yet.	To check whether the button appears in the correct position on screen and can be interacted with by the user.	
7.1.4	Deleting flashcard via a button press	Mouse click	Nothing will happen as the functionality of the button has not been implemented yet.	To check whether the button appears in the correct position on screen and can be interacted with by the user.	

Revise flashcards – 2nd iteration

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
7.2.1	Flip flashcard via button press	Mouse click	The flashcard will be flipped (e.g. if term was on the screen, definition will now be on the screen and vice versa)	To check whether the correct corresponding definition to the term (or vice versa) is displayed on screen when the button is pressed.	
7.2.2	Viewing the next flashcard via a button press	Mouse click	The next flashcard read from the users .csv file will appear on the screen.	To check whether a new flashcard is displayed on screen from the correct set.	
7.2.3	Viewing the previous flashcard via a button press	Mouse click	The previous flashcard read from the users .csv file will appear on the screen.	To check whether the previously displayed flashcard is displayed on screen when the button is pressed.	
7.2.4	Deleting flashcard via	Mouse click	The flashcard will be deleted from the users	To check whether pressing the button	

	a button press		.csv file, and the screen will turn blank as the procedure to search for the next flashcard has not been ran yet.	to remove a flashcard removes it from the users .csv file.	

Revise flashcards – 3rd iteration

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
7.3.1	Flip flashcard via button press	Mouse click	The flashcard will be flipped (e.g. if term was on the screen, definition will now be on the screen and vice versa)	Checking this feature hasn't been affected by the change to deleting flashcards (test 7.3.4).	
7.3.2	Viewing the next flashcard via a button press	Mouse click	The next flashcard read from the users .csv file will appear on the screen.	Checking this feature hasn't been affected by the change to deleting flashcards (test 7.3.4).	
7.3.3	Viewing the previous flashcard via a button press	Mouse click	The previous flashcard read from the users .csv file will appear on the screen.	Checking this feature hasn't been affected by the change to deleting flashcards (test 7.3.4).	
7.3.4	Deleting flashcard via a button press	Mouse click	The flashcard will be deleted from the users .csv file and the next flashcard procedure is then ran to make the next term appear on the screen.	To check whether the next flashcard automatically appears on screen when the user deletes a flashcard.	

Quiz – 1st iteration

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
8.1.1	Question and 4 multiple choice answers should appear on screen	Clicking on the 'Quiz' button from the set menu.	A space for the question should appear on screen, with 4 empty multiple choice buttons.	To check whether the format of the quiz feature is correct before implementing the Q and A system.	
8.1.2	Choosing answer via button press.	Mouse click	Nothing will happen as the functionality of the button has not been implemented yet.	To check whether the buttons appear in the correct places on screen, are a good size, and can be interacted with by the user.	

Quiz – 2nd iteration

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result

8.2.1	Question and 4 multiple choice answers should appear on screen	Clicking on the 'Quiz' button from the set menu.	A question should appear on the screen, with 4 multiple choice answers in the form of buttons.	To check whether the algorithm is successfully reading the question and corresponding 4 answers from the correct set from the users .csv file.	
8.2.2	Choosing answer via button press.	Mouse click	Nothing will happen as the functionality of the button has not been implemented yet.	To check whether the buttons appear in the correct places on screen, are a good size, and can be interacted with by the user.	

Quiz – 3rd iteration

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
8.3.1	Question and 4 multiple choice answers should appear on screen	Clicking on the 'Quiz' button from the set menu.	A question should appear on the screen, with 4 multiple choice answers in the form of buttons. These answers should be randomly placed in the 4 boxes.	To check whether the 4 multiple choice answers being displayed always contain 1 correct answers and 3 other **random** answers from the correct set.	
8.3.2	Choosing answer via button press.	Mouse click	Nothing will happen as the functionality of the button has not been implemented yet.	To check whether the buttons appear in the correct places on screen, are a good size, and can be interacted with by the user.	

Quiz – 4th iteration

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
8.4.1	Question and 4 multiple choice answers should appear on screen	Clicking on the 'Quiz' button from the set menu.	A question should appear on the screen, with 4 multiple choice answers in the form of buttons. These answers should be randomly placed in the 4 boxes.	To check whether the 4 multiple choice answers being displayed always contain 1 correct answers and 3 other **random** answers from the correct set.	

8.4.2	Choosing answer via button press.	Mouse click	A new question should appear with 4 randomly placed multiple-choice answers. This process will repeat until the quiz is complete, when a summary screen should appear with no statistics on it as the score, percentage and time_taken variables have not been implemented yet.	To check that when the user chooses an answer, the next question is displayed again with random multiple choice answers. This feature is key to allowing the quiz to progress.	
8.4.3	Completing Quiz via multiple button presses	Mouse click	A summary screen should appear with the correct score, percentage and time taken. These should also be written to the users csv file.	To check that after the user has gone through all the questions, the correct statistics are displayed on screen and written in the correct positions in the users .csv file.	

Progress Tracker – 1st iteration

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
9.1.1	Raw numbers representation of users data	Clicking the 'Progress Tracker' button from the main menu.	The users scores on quizzes should appear on the screen, with the corresponding set name.	This is to check that the algorithm is correctly reading the users statistics from their .csv file and displaying them in the correct position on the display.	
9.1.2	Visual representation of users data (e.g. graph)	Clicking the 'Progress Tracker' button from the main menu.	The graph should appear but with no data plotted.	This is to check the graph is positioned correctly on the display, with the correct axis labels.	
9.1.3	Choosing a function via a button press (post to Twitter)	Mouse click	Nothing should happen when the button is pressed as the functionality of the button (the Twitter API) has not been implemented yet.	To check whether the button is formatted correctly on the display and whether the user can interact with it.	
9.1.4	Choosing a function via a button press (email reminder)	Mouse click	Nothing should happen when the button is pressed as the functionality of the button (the Google API) has not	To check whether the button is formatted correctly on the display and	

			been implemented yet.	whether the user can interact with it.	

Progress Tracker – 2nd iteration

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
9.2.1	Raw numbers representation of users data	Clicking the 'Progress Tracker' button from the main menu.	The users scores on quizzes should appear on the screen, with the corresponding set name.	This is to check that the algorithm is correctly reading the users statistics from their .csv file and displaying them in the correct position on the display.	
9.2.2	Visual representation of users data (e.g. graph)	Clicking the 'Progress Tracker' button from the main menu.	The graph should appear on the left with the correct data plotted.	This is to check that the correct data has been plotted on the graph, and that the data is clear to see and make deductions about the users progress from.	
9.2.3	Choosing a function via a button press (post to Twitter)	Mouse click	Nothing should happen when the button is pressed as the functionality of the button (the Twitter API) has not been implemented yet.	To check whether the button is formatted correctly on the display and whether the user can interact with it.	
9.2.4	Choosing a function via a button press (email reminder)	Mouse click	Nothing should happen when the button is pressed as the functionality of the button (the Google API) has not been implemented yet.	To check whether the button is formatted correctly on the display and whether the user can interact with it.	

Progress Tracker – 3rd iteration

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
9.3.1	Raw numbers representation of users data	Clicking the 'Progress Tracker' button from the main menu.	The users scores on quizzes should appear on the screen, with the corresponding set name.	This is to check that the algorithm is correctly reading the users statistics from their .csv file and displaying them in the correct position on the display.	

9.3.2	Visual representation of users data (e.g. graph)	Clicking the 'Progress Tracker' button from the main menu.	The graph should appear on the left with the correct data plotted.	This is to check that the correct data has been plotted on the graph, and that the data is clear to see and make deductions about the users progress from.	
9.3.3	Choosing a function via a button press (post to Twitter)	Mouse click	A tweet should be sent out containing the users percentage, time taken and set name in a logical sentence.	This is to check that the algorithm is tweeting the correct information when the button is pressed.	
9.3.4	Choosing a function via a button press (email reminder)	Mouse click	Nothing should happen when the button is pressed as the functionality of the button (the Google API) has not been implemented yet.	To check whether the button is formatted correctly on the display and whether the user can interact with it.	

Progress Tracker – 4th iteration

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
9.4.1	Raw numbers representation of users data	Clicking the 'Progress Tracker' button from the main menu.	The users scores on quizzes should appear on the screen, with the corresponding set name.	This is to check that the algorithm is correctly reading the users statistics from their .csv file and displaying them in the correct position on the display.	
9.4.2	Visual representation of users data (e.g. graph)	Clicking the 'Progress Tracker' button from the main menu.	The graph should appear on the left with the correct data plotted.	This is to check that the correct data has been plotted on the graph, and that the data is clear to see and make deductions about the users progress from.	
9.4.3	Choosing a function via a button press (post to Twitter)	Mouse click	A tweet should be sent out containing the users percentage, time taken and set name in a logical sentence.	This is to check that the algorithm is tweeting the correct information when the button is pressed.	

9.4.4	Choosing a function via a button press (email reminder)	Mouse click	An email should be sent to the users email address with the correct subject and message.	This is to check that an email is successfully sent to the users email address (which is already stored by the 'email' variable in the program), containing the relevant information.	

Post-development Phase
Here I will go through my success criteria and outline how I will test the criteria to check whether they have been met and to test the robustness of my solution.

The stakeholder would like the system to allow them to create an account if they haven't already got one.

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
10.1.1	Attempting to submit valid account details	String 'username123' String 'password123' String 'password123' String 'bradleymak2003 @gmail.com' **VALID DATA**	The users account will be created, their details will be saved to their .csv file, and they will be taken to the main menu.	Ensures the user can successfully create an account using valid account details	
10.1.2	Attempting to submit invalid account details	String 'username123' String 'password123' String 'password123' String '123456789' **INVALID DATA**	An error message will appear at the bottom of the screen notifying the user of the problem (invalid email in this case).	Checks whether the system detects when an invalid email has been entered (and does not crash).	
10.1.3	Attempting to submit incorrect account details	String 'username123' String 'password123' String 'password' String 'bradleymak2003 @gmail.com' **INVALID DATA**	An error message will appear at the bottom of the screen notifying the user that the 2 passwords they entered do not match (i.e. they mistyped their password).	Checks whether the system detects when the two password fields do not match.	

The stakeholder would like the system to allow them to log in to their account if they have already created one.

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
11.1.1	Entering correct username and password	String 'username123' String 'password123' **VALID DATA**	App will verify their account details and take the user to the main menu.	Ensures that the user can log back into the system at a different time using their already existing log-in details.	
11.1.2	Entering incorrect username and password.	String 'username123' String 'password1' **INVALID DATA**	An error message will appear at the bottom of the screen alerting the user to the problem (most likely an incorrect password)	Ensures the app is secure by checking whether the system detects when the user has entered a password that is not associated with the entered username.	

The stakeholder would like the system to have a clear main menu.
The stakeholder would like the system to be user-friendly and easy to understand.

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
12.1.1	Selection of option via 'Sign_up' button	Mouse click	Layout will be updated to 'SIGNUP'	Ensures the user can choose to sign up easily via a button press.	
12.1.2	Selection of option via 'log_in' button	Mouse click	Layout will be updated to 'LOGIN'	Ensures the user can choose to log in easily via a button press.	
12.1.3	Selection of option via 'new_set' button	Mouse click	Layout will be updated to SETNAME, so the user will be taken to a different screen.	Ensures the user can choose to create a new set easily via a button press.	
12.1.4	Selection of option via 'my_sets' button	Mouse click	Layout will be updated to SETS, so the user will be taken to a different screen.	Ensures the user can choose to view their sets easily via a button press.	
12.1.5	Selection of option via 'progress_tracker' button	Mouse click	Layout will be updated to PROGRESS, so the user will be taken to a different screen.	Ensures the user can choose to view their progress easily via a button press.	
12.1.6	Selection of option via 'manage_account' button	Mouse click	Layout will be updated to ACCOUNT, so the user will be taken to a different screen.	Ensures the user can choose to manage their account easily via a button press.	
12.1.7	Selection of option via 'log_out' button	Mouse click	User will be logged out and taken back to the start menu	Ensures the user can choose to log	

			(layout is updated to STARTMENU)	out easily via a button press.	
12.1.8	Selection of option via 'Exit' button	Mouse click	App will close as event loop is broken.	Ensures the user can choose to exit the app easily via a button press.	

The stakeholder would like the system to allow them to create their own sets of flashcards.

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
13.1.1	Entering name of set into textual input boxes	String 'Data Structures' **VALID DATA**	Name of set will be written to the users personal .csv file, and the relevant button on the choose set layout will be updated to show the new set name.	Ensures the user can create a new set of flashcards under whatever name they want, and that this name is written to the users .csv file and the buttons on the 'MYSETS' menu are updated to include this new set.	

The stakeholder would like the system to allow them to view sets of flashcards they have already created.

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
14.1.1	Viewing names of sets of flashcards the user has already created	Clicking the 'My Sets' button from the main menu.	The names of the users sets should be displayed on the buttons on the MYSETS display.	To ensure that the set names displayed are correct and up-to-date.	

The stakeholder would like the system to allow them to revise sets of flashcards they have created.

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
15.1.1	Flip flashcard via button press	Mouse click	The flashcard will be flipped (e.g. if term was on the screen, definition will now be on the screen and vice versa)	Ensures that the user can 'flip' flashcards (i.e. alternate between the term and definition) via a button press.	
15.1.2	Viewing the next flashcard via a button press	Mouse click	The next flashcard read from the users .csv file will appear on the screen.	Ensures that the next flashcard displayed is from the correct set and is not repeated.	
15.1.3	Viewing the previous flashcard via	Mouse click	The previous flashcard read from the users	Ensures that when tis button is pressed, the previous flashcard is	

	a button press		.csv file will appear on the screen.	successfully displayed to the user.	

The stakeholder would like the system to allow them to edit already existing sets of flashcards

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
16.1.1	Submitting a term and definition to be created as a flashcard via 2 textual inputs and a button press.	String 'Static' String 'Size cannot change during runtime' **VALID DATA**	Term and definition will be written to the relative positions in the users personal .csv file, and the term and definition textual input boxes will be cleared.	Ensures that any term/definition can be entered and written to the .csv file without an error occurring.	
16.1.2	Submitting a term and definition to be created as a flashcard via 2 textual inputs and a button press.	String '12*4' String '48' **VALID DATA**	Term and definition will be written to the relative positions in the users personal .csv file, and the term and definition textual input boxes will be cleared.	Ensures that any term/definition can be entered and written to the .csv file without an error occurring.	
16.1.3	Deleting flashcard via a button press	Mouse click	The flashcard will be deleted from the users .csv file and the next flashcard procedure is then ran to make the next term appear on the screen.	Ensures that the user can successfully delete a flashcard from a set (by checking whether it has been removed from the users .csv file).	

The stakeholder would like the system to allow them to test their knowledge on a particular set of flashcards.

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
17.1.1	Question and 4 multiple choice answers should appear on screen. User should be able to choose an answer.	Clicking the 'Quiz' button from the set menu. Mouse click.	A question should appear on the screen, with 4 multiple choice answers in the form of buttons. When an answer is chosen, a new question should appear with 4 randomly placed multiple-choice answers. This process	To ensure that the quiz feature works correctly regardless of which set the user chooses, and that the questions and answers are random and not repeated.	

			will repeat until the quiz is complete, when a summary screen should appear with no statistics on it as the score, percentage and time_taken variables have not been implemented yet.	To ensure that the quiz ends after all of the questions have been answered.	
17.1.2	Quiz Summary	Visual output when quiz is complete via multiple button presses.	A summary screen should appear with the correct score, percentage and time taken. These should also be written to the users csv file.	To ensure that the correct statistics are displayed on screen when the quiz ends.	

The stakeholder would like the system to keep track of their progress and show it in a clear and concise way.
The stakeholder would like the system to allow them to share their results and progress on social media.
The stakeholder would like the system to send them email notifications when prompted to.

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
18.1.1	Raw numbers representation of users data	Clicking the 'Progress Tracker' button from the main menu.	The users scores on quizzes should appear on the screen, with the corresponding set name.	To ensure that the statistics displayed on screen are correct.	
18.1.2	Visual representation of users data (e.g. graph)	Clicking the 'Progress Tracker' button from the main menu.	The graph should appear on the left with the correct data plotted.	To ensure that the graph is plotted correctly and is easy for the user to see and deduce their progress from.	
18.1.3	Choosing a function via a button press (post to Twitter)	Mouse click	A tweet should be sent out containing the users percentage, time taken and set name in a logical sentence.	To ensure that the user can successfully tweet their progress onto the applications Twitter page via a button press.	

18.1.4	Choosing a function via a button press (email reminder)	Mouse click	An email should be sent to the users email address with the correct subject and message.	To ensure that the user can successfully send themselves an email reminder to their email address stored by the app.	

Development Section

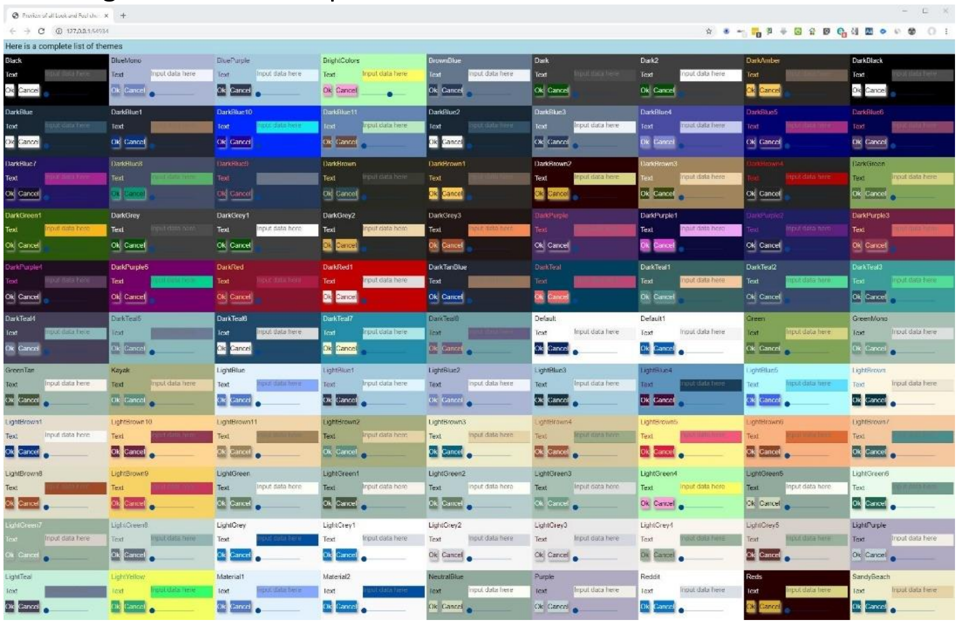
This section will cover the development of my solution including testing throughout, any problems encountered and how they were solved, and screenshots/videos of my solution in action. I will also be asking my stakeholders for feedback and evaluating my success criteria throughout.

Initialisation

I have first started by importing the PySimpleGUI library in Python and defining the colour theme for my app:

```python
import PySimpleGUI as sg
#imports PySimpleGUI
sg.theme('DarkBlue')
#sets a colour theme for the program
```

I have chosen this colour theme after looking online at all of the available colour schemes in PySimpleGUI, and I think it allows for a good contrast between the background and the buttons allowing for the app to look sharper. I chose PySimpleGUI as it allows for simple and easily changeable layout designs and allows for a clear structure in my solution via event checking in the event loop.


Here is a complete list of themes
Black BlueMono BluePurple BrightColors BrownBlue Dark Dark2 DarkAmber DarkBlack
DarkBlue DarkBlue1 DarkBlue10 DarkBlue11 DarkBlue2 DarkBlue3 DarkBlue4 DarkBlue5 DarkBlue6
DarkBlue7 DarkBlue8 DarkBlue9 DarkBrown DarkBrown1 DarkBrown2 DarkBrown3 DarkBrown4 DarkGreen
DarkGreen1 DarkGrey DarkGrey1 DarkGrey2 DarkGrey3 DarkPurple DarkPurple1 DarkPurple2 DarkPurple3
DarkPurple4 DarkPurple5 DarkRed DarkRed1 DarkTanBlue DarkTeal DarkTeal1 DarkTeal2 DarkTeal3
DarkTeal4 DarkTeal5 DarkTeal6 DarkTeal7 DarkTeal8 Default Default1 Green GreenMono
GreenTan Kayak LightBlue LightBlue1 LightBlue2 LightBlue3 LightBlue4 LightBlue5 LightBrown
LightBrown1 LightBrown10 LightBrown11 LightBrown2 LightBrown3 LightBrown4 LightBrown5 LightBrown6 LightBrown7
LightBrown8 LightBrown9 LightGreen LightGreen1 LightGreen2 LightGreen3 LightGreen4 LightGreen5 LightGreen6
LightGreen7 LightGreen8 LightGrey LightGrey1 LightGrey2 LightGrey3 LightGrey4 LightGrey5 LightPurple
LightTeal LightYellow Material1 Material2 NeutralBlue Purple Reddit Reds SandyBeach
Text
Input data here
Ok
Cancel

Start Menu

I have next created the start menu layout called 'StartScreen' which should consist of 3 centralised buttons for each of the functions available at this point.

```python
StartScreen = [[sg.Text('Welcome to the revision app! Please log in, or create a new account to continue.', justification='center', size=(35, 3))],
        # Creates a centred title for the layout which will appear at the top of the screen
        [sg.Button('Log In', key='LOGIN', border_width=5, size=(13, 3), justification='center')],
        [sg.Button('Create an account', key='SIGNUP', border_width=5, size=(13, 3), justification='center')],
        [sg.Button('EXIT', key='EXIT', border_width=5, size=(13, 3), button_color=('white', 'red'), justification='center')]]
#         Creates 3 centralised buttons on different 'layers' of the screen which have text on them stating their purpose. The Exit button is red to signify its purpose.
```

I also defined the first column (I will be using columns in my solution to allow the program to 'switch' between layout by making the relevant columns visible/invisible) and a window for the GUI to be displayed on.

```
window = sg.Window('Revision App', layout)
#Creates a window with a given name for the layout to be displayed on whilst the program is running.
```

```
layout = [[sg.Column(StartScreen, key='-COL1-', visible=True)]]
#Defines the initial layout of the program as StartScreen.
```

I next created the event loop which allows the program to run so I can check the layout as part of my first iteration of testing:

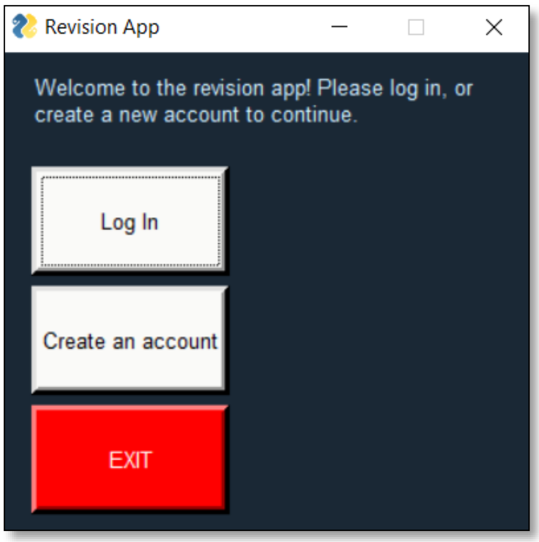
```
while True:  # event loop to allow the program to run
    event, values = window.read()
    print(event, values)
#   Read and print the event that happened and the values dictionary. This allows me to see if the buttons have been defined correctly.
    if event == None:
        break
#   If the window is closed, the event loop breaks and the program stops running.
```

However, when I run this code I receive an error shown here:

```
Traceback (most recent call last):
  File "C:/Users/bradl/PycharmProjects/Projects/NEA - Revision App.py", line 8, in <module>
    [sg.Button('Log In', key='LOGIN', border_width=5, size=(13, 3), justification='center')],
TypeError: __init__() got an unexpected keyword argument 'justification'
```

This is due to the fact that there is no 'center' justification for buttons, so I tried putting the justification in the column element instead of defining it for each individual button/text element, and whilst I now got no error, the buttons and text were not actually centralised.

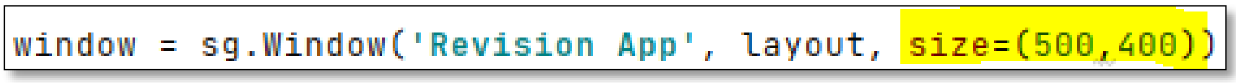
```
layout = [[sg.Column(StartScreen, key='-COL1-', visible=True, justification='center')]]
```



Welcome to the revision app! Please log in, or create a new account to continue.

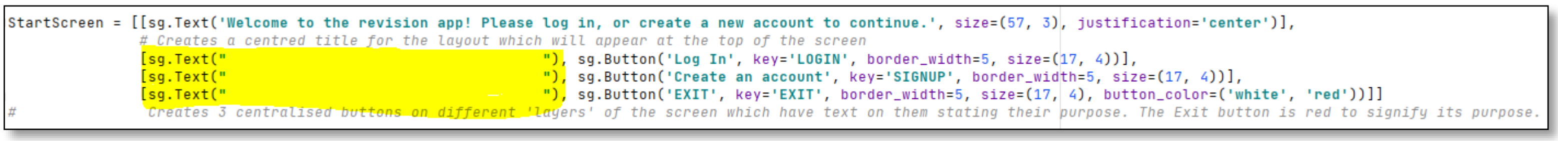
As you can see from the screenshot on the left, the buttons all look correct (i.e. correct size, colour and text) but are not appearing in the correct positions on the display (central).

To solve this problem I have started by giving the window a fixed size, meaning that it will appear the same on all devices:



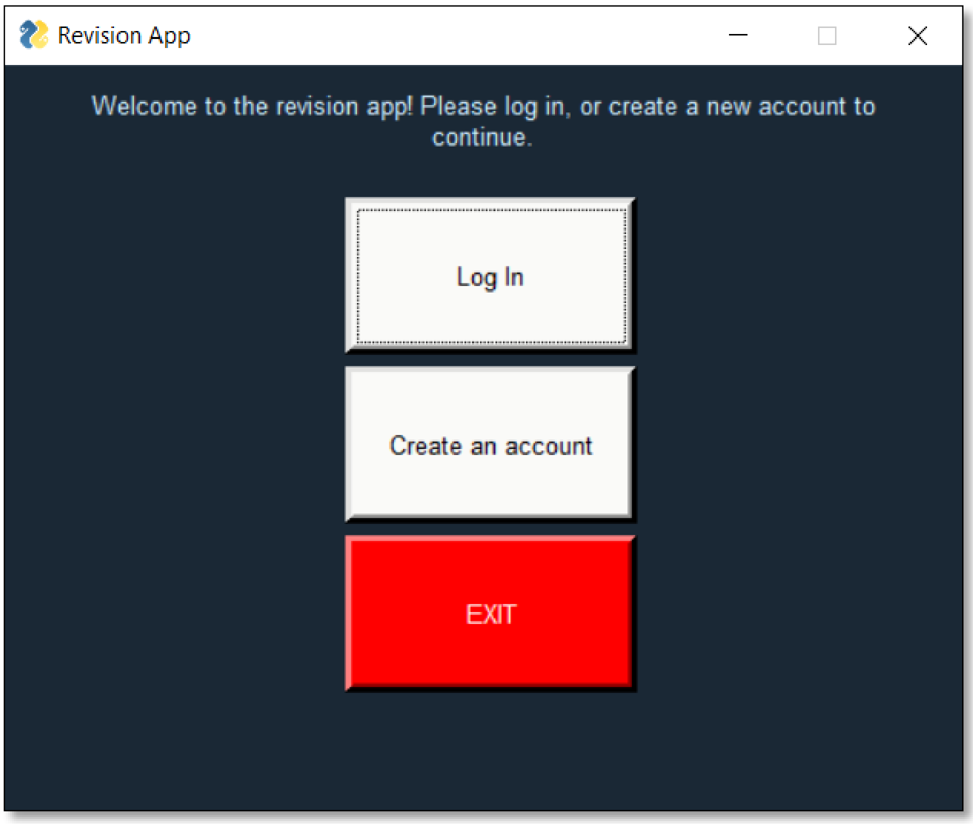
```
window = sg.Window('Revision App', layout, size=(500,400))
```

I have next used empty text elements to centralise the buttons on the display, which because of the fixed display size, will mean that the buttons always appear central on all devices, allowing for convenient, on the go usage of the app on any device which was a feature requested by my stakeholders. The 'center' justification worked for the text element.



```
StartScreen = [[sg.Text('Welcome to the revision app! Please log in, or create a new account to continue.', size=(57, 3), justification='center')],
               # Creates a centred title for the layout which will appear at the top of the screen
               [sg.Text("                                      "), sg.Button('Log In', key='LOGIN', border_width=5, size=(17, 4))],
               [sg.Text("                                      "), sg.Button('Create an account', key='SIGNUP', border_width=5, size=(17, 4))],
               [sg.Text("                                      "), sg.Button('EXIT', key='EXIT', border_width=5, size=(17, 4), button_color=('white', 'red'))]]
#               Creates 3 centralised buttons on different 'layers' of the screen which have text on them stating their purpose. The Exit button is red to signify its purpose.
```

The result is a layout with the buttons all correctly formatted and positioned in order to make the display as user friendly and easy to use as possible in line with my success criteria I set out in the analysis section.



Revision App
Welcome to the revision app! Please log in, or create a new account to continue.
Log In
Create an account
EXIT

I can now carry out the first iteration of my testing for the start menu (clip shown in testing evidence powerpoint):

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
1.1.1	Selection of option via 'Sign_up' button	Mouse click	Nothing will happen as the functionality of the button has not been implemented yet, and the SIGNUP layout has not been defined.	Checking whether the buttons appear on the screen as intended and are pressable.	**Buttons appears correctly on the interface but is not functioning as of yet.**
1.1.2	Selection of option via	Mouse click	Nothing will happen as the functionality of the button has not been	Checking whether the buttons appear	**Buttons appears correctly on the**

	'log_in' button		implemented yet, and the LOGIN layout has not been defined.	on the screen as intended and are pressable.	**interface but is not functioning as of yet.**
1.1.3	Selection of option via 'Exit' button	Mouse click	Nothing will happen as the functionality of the button has not been implemented yet.	Checking whether the buttons appear on the screen as intended and are pressable.	**Buttons appears correctly on the interface but is not functioning as of yet.**

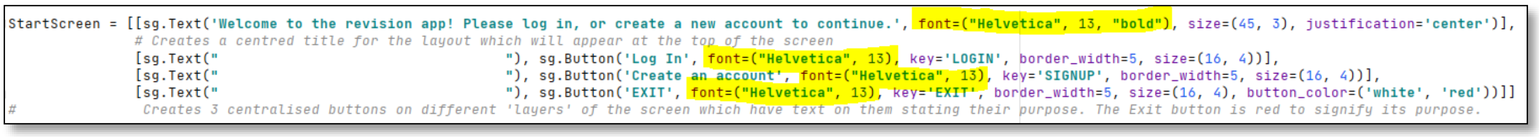
Start Screen – Review 1

Success Criteria Review

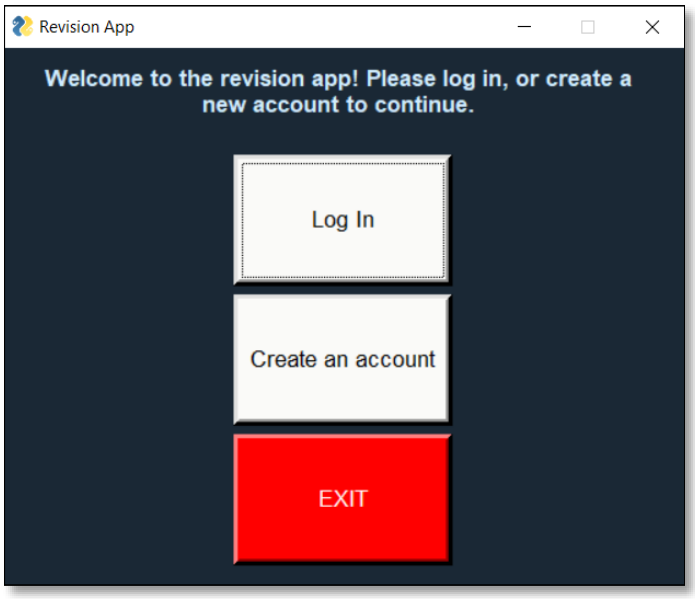
The development so far shows I am focusing on the success criteria of offering a user friendly design (success criteria 10 in analysis section) and allowing my app to be used and appear the same on a wide range of devices, allowing for convenience as requested by my stakeholders.

Stakeholder feedback

I asked one of my stakeholders, Vivek, for his opinions on the layout and whether he thought it was easy to use and understand. He said "The layout is good and the functionality is clear but it would be nice if the text was a bit bigger/bolder."
After hearing this feedback, I have increased the font size of both the text and button elements (along with some adjustment of the size of the elements in order to keep them centralised) and made the text at the top of the screen bold in order to make it stand out and easier to read.

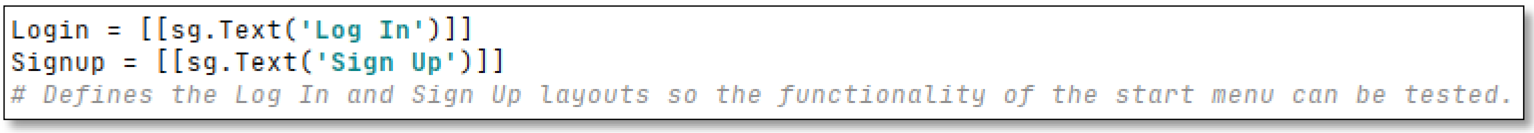


```
StartScreen = [[sg.Text('Welcome to the revision app! Please log in, or create a new account to continue.', font=("Helvetica", 13, "bold"), size=(45, 3), justification='center')],
               # Creates a centred title for the layout which will appear at the top of the screen
               [sg.Text("                                            "), sg.Button('Log In', font=("Helvetica", 13), key='LOGIN', border_width=5, size=(16, 4))],
               [sg.Text("                                            "), sg.Button('Create an account', font=("Helvetica", 13), key='SIGNUP', border_width=5, size=(16, 4))],
               [sg.Text("                                            "), sg.Button('EXIT', font=("Helvetica", 13), key='EXIT', border_width=5, size=(16, 4), button_color=('white', 'red'))]]
#               Creates 3 centralised buttons on different 'layers' of the screen which have text on them stating their purpose. The Exit button is red to signify its purpose.
```


Revision App
Welcome to the revision app! Please log in, or create a new account to continue.
Log In
Create an account
EXIT

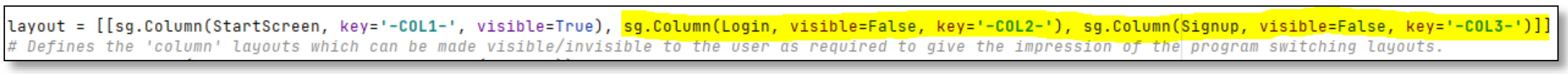
This is how the layout now looks.

I have quickly defined the sign up and log in layouts to allow me to test the functionality of my buttons. These are not the final layouts, but they are sufficient to allow me to carry out iteration 2 of the start menu testing.:



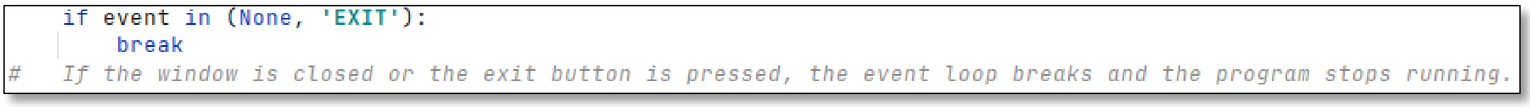
```
Login = [[sg.Text('Log In')]]
Signup = [[sg.Text('Sign Up')]]
# Defines the Log In and Sign Up layouts so the functionality of the start menu can be tested.
```

I have also added the two new layouts as columns which, as stated earlier, can be made visible/invisible as required to give the impression of the program switching interfaces.



```
layout = [[sg.Column(StartScreen, key='-COL1-', visible=True), sg.Column(Login, visible=False, key='-COL2-'), sg.Column(Signup, visible=False, key='-COL3-')]]
# Defines the 'column' layouts which can be made visible/invisible to the user as required to give the impression of the program switching layouts.
```

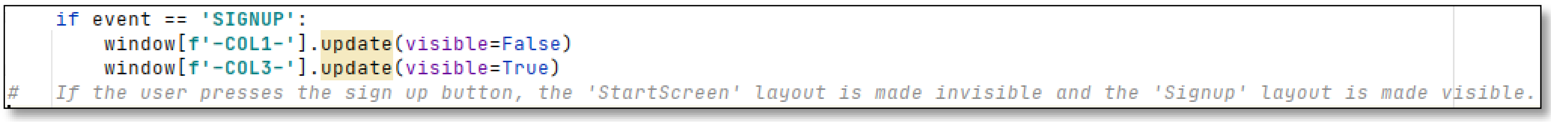
I have started by implementing the functionality of the 'Exit' button by adding the following piece of code, which tells the program that if the button with the ID 'Exit' is pressed (or if the window is closed) the event loop should break and thus the app will close.



```
    if event in (None, 'EXIT'):
        break
#   If the window is closed or the exit button is pressed, the event loop breaks and the program stops running.
```

I then tested the functionality of the button (test 1.2.3 in testing evidence powerpoint) and it worked as intended with no issues.

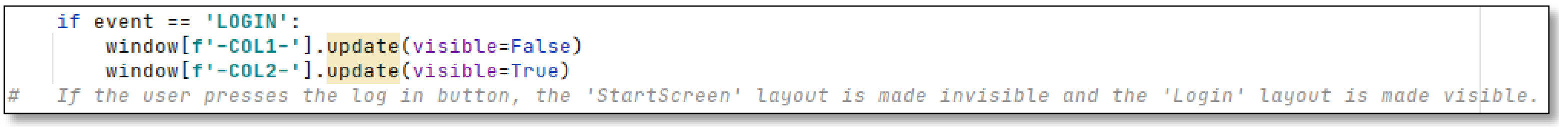
Next, I implemented the functionality of the signup button. To do this, when the button is pressed, the Start Screen layout is made invisible and simultaneously the Sign Up layout is made visible to the user. This gives the impression of the program switching layouts when in reality all of the layouts are there, some are just invisible and so only the visible layout is shown on the screen to the user.



```
    if event == 'SIGNUP':
        window[f'-COL1-'].update(visible=False)
        window[f'-COL3-'].update(visible=True)
#   If the user presses the sign up button, the 'StartScreen' layout is made invisible and the 'Signup' layout is made visible.
```

After coding this, I tested the functionality of the button (test 1.2.1 in testing evidence powerpoint) and the test was successful with no issues – the correct layout was displayed upon pressing the button.

Finally, I implemented the functionality of the login button. This works in an identical way to the signup button, except the layout which is made visible is the LOGIN layout rather than the SIGNUP layout.



```
    if event == 'LOGIN':
        window[f'-COL1-'].update(visible=False)
        window[f'-COL2-'].update(visible=True)
#   If the user presses the log in button, the 'StartScreen' layout is made invisible and the 'Login' layout is made visible.
```

Again, I tested the functionality of the login button (test 1.2.2 in testing evidence powerpoint) and the test was successful with no issues – the correct layout was displayed on screen to the user.

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result

1.2.1	Selection of option via 'Sign_up' button	Mouse click	Layout will be updated to 'SIGNUP'	To check whether the sign-up screen is accessible via a button press from the start menu.	**The correct layout ('SIGNUP') was displayed on screen.**
1.2.2	Selection of option via 'log_in' button	Mouse click	Layout will be updated to 'LOGIN'	To check whether the log-in screen is accessible via a button press from the start menu.	**The correct layout ('LOGIN') was displayed on screen.**
1.2.3	Selection of option via 'Exit' button	Mouse click	App will close as event loop is broken.	To check whether the Exit button functions as intended from the start menu.	**App closes as event loop is broken.**

Start Screen – Review 2

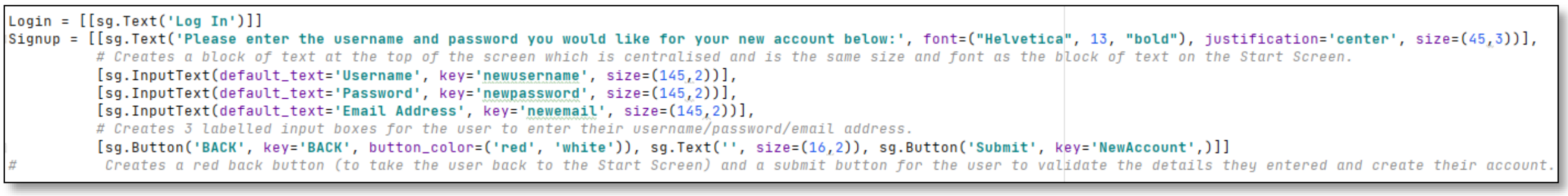
Success Criteria

I am now partway through success criteria 1 and 2 about allowing the user to log in/sign up. Creating the functional buttons which allow the user to access the page to log in/sign up is part of this success criteria and thus whilst success criteria 1 and 2 are not fully complete yet, I am now partway there.

Sign up – Stage 1

I have next decided to develop the sign-up feature of my app, as this feature will be required before the log in feature can be tested. It is also the first feature a user will come across when they start using the app (as they will first need to create an account).

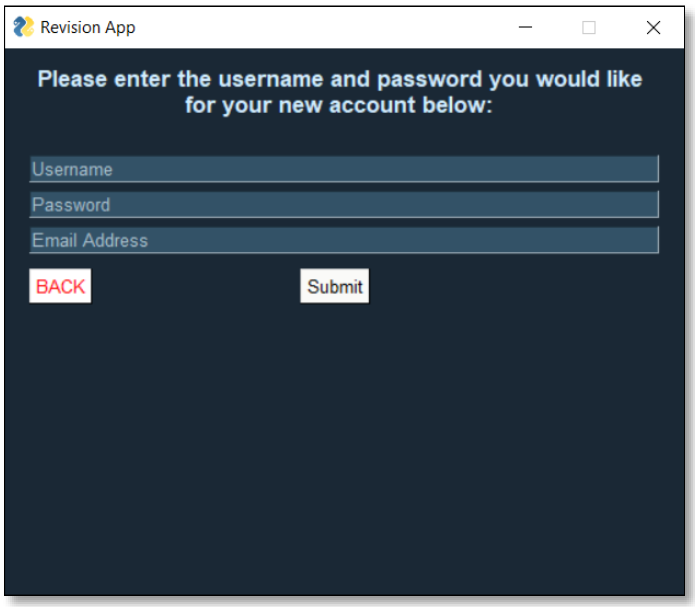
Firstly, I need to create the proper SIGNUP layout rather than the test layout I used for the previous iteration of tests. This is the code I have written for the SIGNUP layout after some minor changes to the size of elements (via trial and error) to ensure they were properly positioned:



```
Login = [[sg.Text('Log In')]]
Signup = [[sg.Text('Please enter the username and password you would like for your new account below:', font=("Helvetica", 13, "bold"), justification='center', size=(45,3))],
          # Creates a block of text at the top of the screen which is centralised and is the same size and font as the block of text on the Start Screen.
          [sg.InputText(default_text='Username', key='newusername', size=(145,2))],
          [sg.InputText(default_text='Password', key='newpassword', size=(145,2))],
          [sg.InputText(default_text='Email Address', key='newemail', size=(145,2))],
          # Creates 3 labelled input boxes for the user to enter their username/password/email address.
          [sg.Button('BACK', key='BACK', button_color=('red', 'white')), sg.Text('', size=(16,2)), sg.Button('Submit', key='NewAccount',)]]
#          Creates a red back button (to take the user back to the Start Screen) and a submit button for the user to validate the details they entered and create their account.
```

I ran the app and when I clicked on the Signup button, this was the display which appeared:

After my previous conversation with one of my stakeholders, Vivek, I decided to make the block of text at the top of the screen large and bold to make it easy to read.
The input boxes are sufficiently large and the submit/back buttons are properly positioned and coloured as I intended.

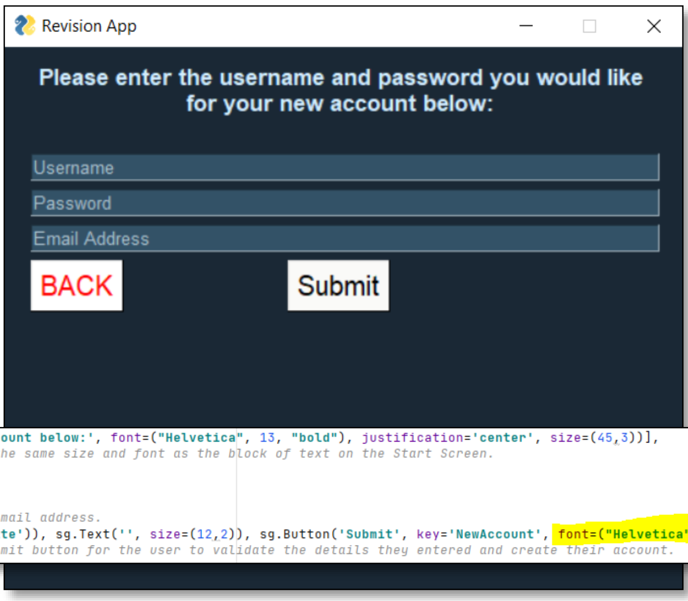

Revision App
Please enter the username and password you would like for your new account below:
Username
Password
Email Address
BACK
Submit

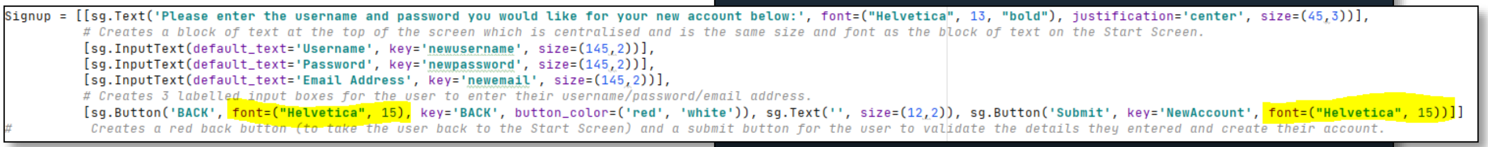
Sign Up – Review 1

Stakeholder feedback

I sent this prototype to one of my stakeholders, Marcus, to try out for his feedback on the design of the SIGNUP layout. He said "I like the text at the top, I'd prefer the input boxes to appear empty so I don't have to clear the text from them before I type something and have the purpose of each input box as text to the left of it. It'd also be better if the buttons were a bit bigger. Other than that I like it".

I followed up on Marcus' request for the input boxes to be made larger, and I increased their size so that the layout now appears like this:

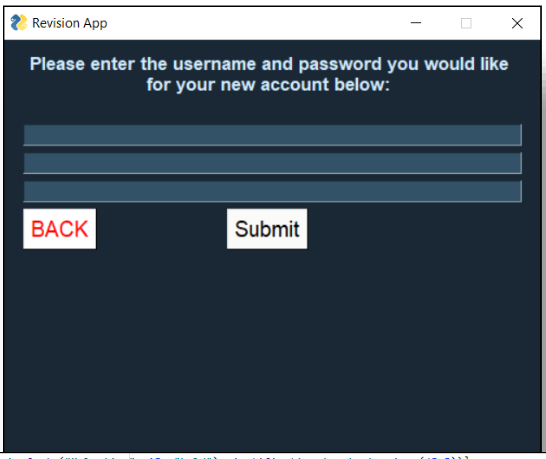

Revision App
Please enter the username and password you would like for your new account below:
Username
Password
Email Address
BACK
Submit



```
Signup = [[sg.Text('Please enter the username and password you would like for your new account below:', font=("Helvetica", 13, "bold"), justification='center', size=(45,3))],
          # Creates a block of text at the top of the screen which is centralised and is the same size and font as the block of text on the Start Screen.
          [sg.InputText(default_text='Username', key='newusername', size=(145,2))],
          [sg.InputText(default_text='Password', key='newpassword', size=(145,2))],
          [sg.InputText(default_text='Email Address', key='newemail', size=(145,2))],
          # Creates 3 labelled input boxes for the user to enter their username/password/email address.
          [sg.Button('BACK', font=("Helvetica", 15), key='BACK', button_color=('red', 'white')), sg.Text('', size=(12,2)), sg.Button('Submit', key='NewAccount', font=("Helvetica", 15))]]
#          Creates a red back button (to take the user back to the Start Screen) and a submit button for the user to validate the details they entered and create their account.
```

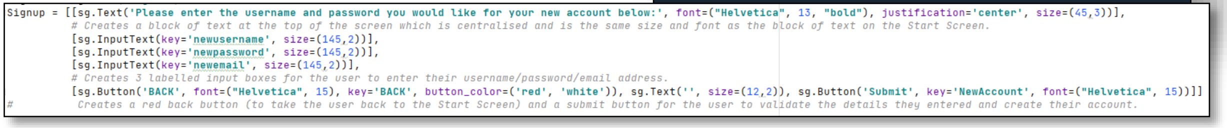
Marcus also requested that the input boxes had the text outside of the box rather than inside so that he didn't have to clear the text every time he wanted to enter his details. I contacted Sam and Vivek about this issue too as I was interested to hear their opinions on it. Sam wasn't overly bothered but Vivek also said he'd prefer the text outside the input box rather than inside for the same reason (to allow for quicker and easier account creation). Therefore, I have decided that I am going to change this prototype in line with the feedback from my stakeholders.

Firstly, I have removed the default text from each input text element, so they now just appear blank when the program is ran as shown here:



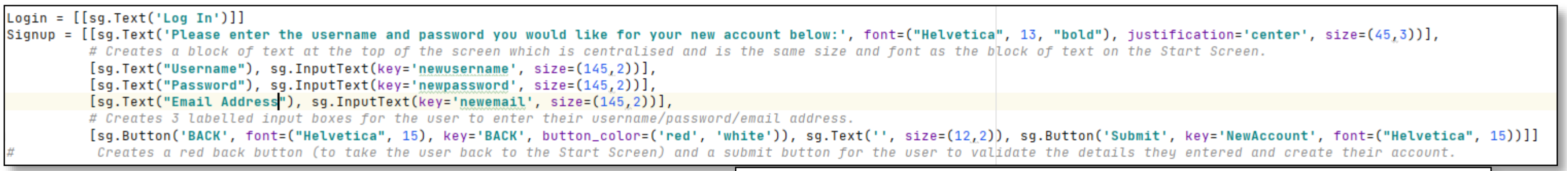
Revision App

Please enter the username and password you would like for your new account below:



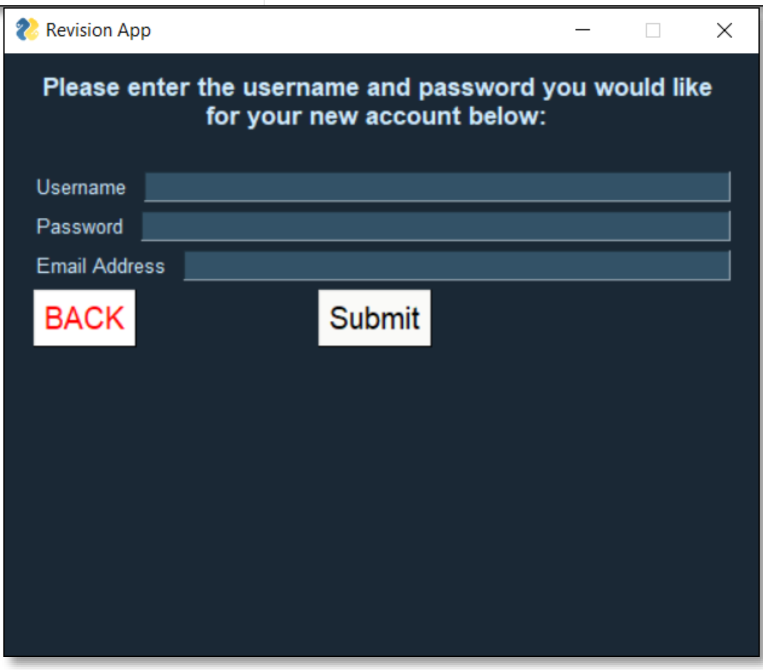
```
Signup = [[sg.Text('Please enter the username and password you would like for your new account below:', font=("Helvetica", 13, "bold"), justification='center', size=(45,3))],
          # Creates a block of text at the top of the screen which is centralised and is the same size and font as the block of text on the Start Screen.
          [sg.InputText(key='newusername', size=(145,2))],
          [sg.InputText(key='newpassword', size=(145,2))],
          [sg.InputText(key='newemail', size=(145,2))],
          # Creates 3 labelled input boxes for the user to enter their username/password/email address.
          [sg.Button('BACK', font=("Helvetica", 15), key='BACK', button_color=('red', 'white')), sg.Text('', size=(12,2)), sg.Button('Submit', key='NewAccount', font=("Helvetica", 15))]]
#          Creates a red back button (to take the user back to the Start Screen) and a submit button for the user to validate the details they entered and create their account.
```

I have next added a text element before each input box, each displaying the required field to be entered in each box:



```
Login = [[sg.Text('Log In')]]
Signup = [[sg.Text('Please enter the username and password you would like for your new account below:', font=("Helvetica", 13, "bold"), justification='center', size=(45,3))],
          # Creates a block of text at the top of the screen which is centralised and is the same size and font as the block of text on the Start Screen.
          [sg.Text("Username"), sg.InputText(key='newusername', size=(145,2))],
          [sg.Text("Password"), sg.InputText(key='newpassword', size=(145,2))],
          [sg.Text("Email Address"), sg.InputText(key='newemail', size=(145,2))],
          # Creates 3 labelled input boxes for the user to enter their username/password/email address.
          [sg.Button('BACK', font=("Helvetica", 15), key='BACK', button_color=('red', 'white')), sg.Text('', size=(12,2)), sg.Button('Submit', key='NewAccount', font=("Helvetica", 15))]]
#          Creates a red back button (to take the user back to the Start Screen) and a submit button for the user to validate the details they entered and create their account.
```

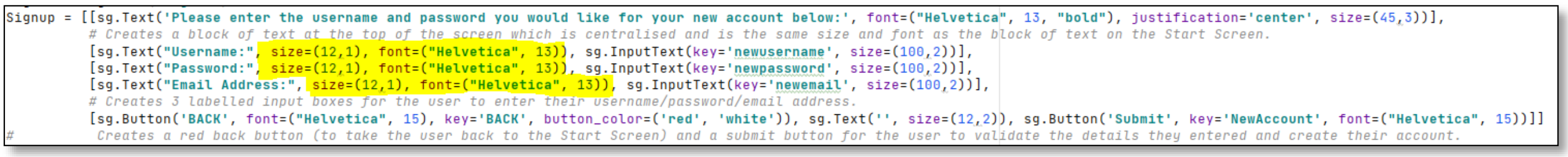
The result was this interface when the program was ran:



Revision App

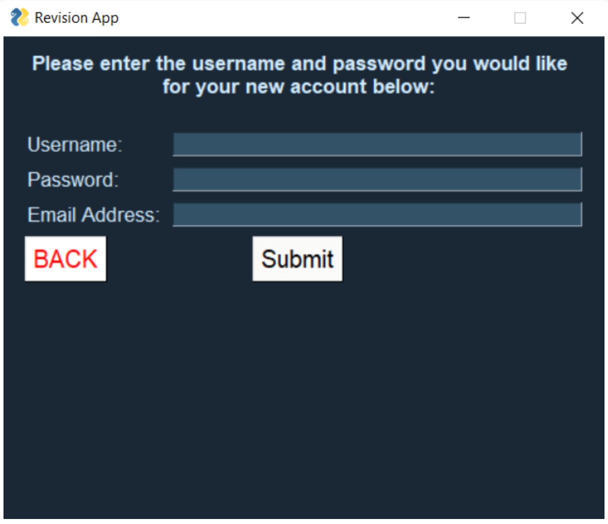
Please enter the username and password you would like for your new account below:

I decided to increase the text size in order to make it easier to read (in line with an earlier request from my stakeholders) and aligned the input boxes by setting each text element to a given size:



```
Signup = [[sg.Text('Please enter the username and password you would like for your new account below:', font=("Helvetica", 13, "bold"), justification='center', size=(45,3))],
          # Creates a block of text at the top of the screen which is centralised and is the same size and font as the block of text on the Start Screen.
          [sg.Text("Username:", size=(12,1), font=("Helvetica", 13)), sg.InputText(key='newusername', size=(100,2))],
          [sg.Text("Password:", size=(12,1), font=("Helvetica", 13)), sg.InputText(key='newpassword', size=(100,2))],
          [sg.Text("Email Address:", size=(12,1), font=("Helvetica", 13)), sg.InputText(key='newemail', size=(100,2))],
          # Creates 3 labelled input boxes for the user to enter their username/password/email address.
          [sg.Button('BACK', font=("Helvetica", 15), key='BACK', button_color=('red', 'white')), sg.Text('', size=(12,2)), sg.Button('Submit', key='NewAccount', font=("Helvetica", 15))]]
#          Creates a red back button (to take the user back to the Start Screen) and a submit button for the user to validate the details they entered and create their account.
```

The result is an interface which looks clean, simple and effective and has my stakeholders requests in mind:


Revision App
Please enter the username and password you would like for your new account below:
Username:
Password:
Email Address:
BACK
Submit

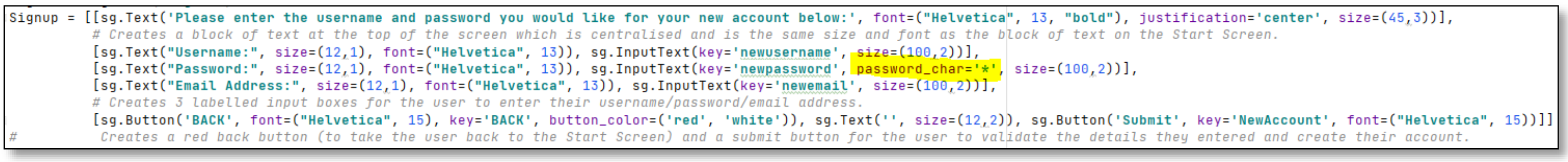
Now that the design of the SIGNUP interface has been finalised in line with my stakeholders requests, I can begin the first iteration of testing of this prototype.

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
2.1.1	Entering username, password and email into relevant fields	String 'username123' String 'password123' String 'bradleymak2003 @gmail.com' **VALID DATA**	App will allow user to enter these strings into the relevant input boxes. The password should be dotted.	Checking whether the user can enter a string into the textual input boxes that should appear on screen.	**Program allows user to enter any data type into the input boxes via the keyboard. However, the password was not dotted.**
2.1.2	Selection of button to confirm account details	Mouse click	Nothing, as functionality of button has not been implemented yet.	To check whether the button on screen appears in the correct place and allows user interaction.	**Nothing happens when the submit (or back) button are pressed as the functionality of these buttons has not been implemented yet.**

Testing evidence for tests 2.1.1 and 2.1.2 can be found in the testing evidence powerpoint.

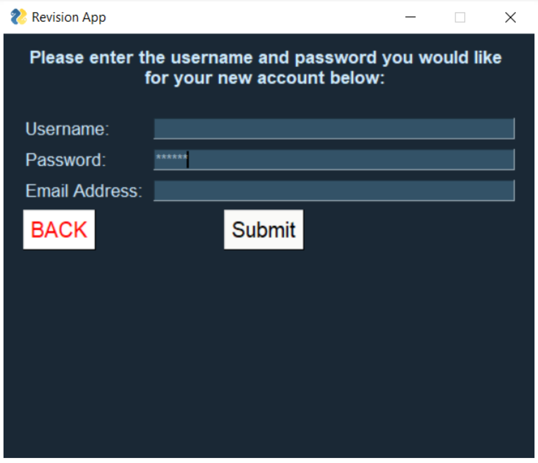
However, whilst testing this prototype, I noticed that the password is not dotted whilst it is being entered. This puts the user at risk of shouldering (where someone steals their password by looking at it whilst it is being entered) as the password is not dotted (hidden) as it is being entered.

I did a bit of research about how to do this as is it something I have not done before, and I found the following website: https://python-forum.io/Thread-PyGUI-Hi-All-how-to-hide-mask-user-input-in-PySimpleGUI. It turns out it is quite simple and just requires the password character to be defined within the element (highlighted):



```
Signup = [[sg.Text('Please enter the username and password you would like for your new account below:', font=("Helvetica", 13, "bold"), justification='center', size=(45,3))],
          # Creates a block of text at the top of the screen which is centralised and is the same size and font as the block of text on the Start Screen.
          [sg.Text("Username:", size=(12,1), font=("Helvetica", 13)), sg.InputText(key='newusername', size=(100,2))],
          [sg.Text("Password:", size=(12,1), font=("Helvetica", 13)), sg.InputText(key='newpassword', password_char='*', size=(100,2))],
          [sg.Text("Email Address:", size=(12,1), font=("Helvetica", 13)), sg.InputText(key='newemail', size=(100,2))],
          # Creates 3 labelled input boxes for the user to enter their username/password/email address.
          [sg.Button('BACK', font=("Helvetica", 15), key='BACK', button_color=('red', 'white')), sg.Text('', size=(12,2)), sg.Button('Submit', key='NewAccount', font=("Helvetica", 15))]]
#           Creates a red back button (to take the user back to the Start Screen) and a submit button for the user to validate the details they entered and create their account.
```

Now when a password is entered, the password appears as asterisks rather than the actual characters, making my program much more secure:


Revision App
Please enter the username and password you would like for your new account below:
Username:
Password:
******
Email Address:
BACK
Submit

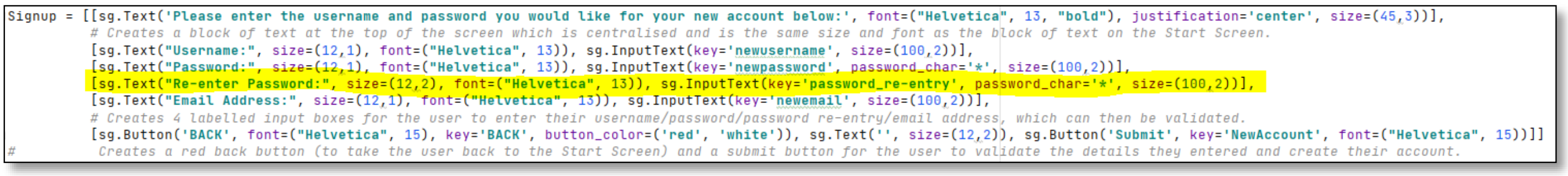
Testing evidence of this can be found in the testing evidence powerpoint (test 2.1.1 re-run).

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
2.1.1	Entering username, password and email into relevant fields	String 'username123' String 'password123' String 'bradleymak2003 @gmail.com' **VALID DATA**	App will allow user to enter these strings into the relevant input boxes. The password should be dotted.	Checking whether the user can enter a string into the textual input boxes that should appear on screen.	**Program allows user to enter any data type into the input boxes via the keyboard. The password is now hidden as it is entered.**
2.1.2	Selection of button to confirm account details	Mouse click	Nothing, as functionality of button has not been implemented yet.	To check whether the button on screen appears in the correct place and allows user interaction.	**Nothing happens when the submit (or back) button are pressed as the functionality of these buttons has not been implemented yet.**

<u>Success Criteria Review</u>
I am working towards success criteria 1, but I have not fully met it yet. The functionality of the submit button needs to be coded before this criteria has been fully met.

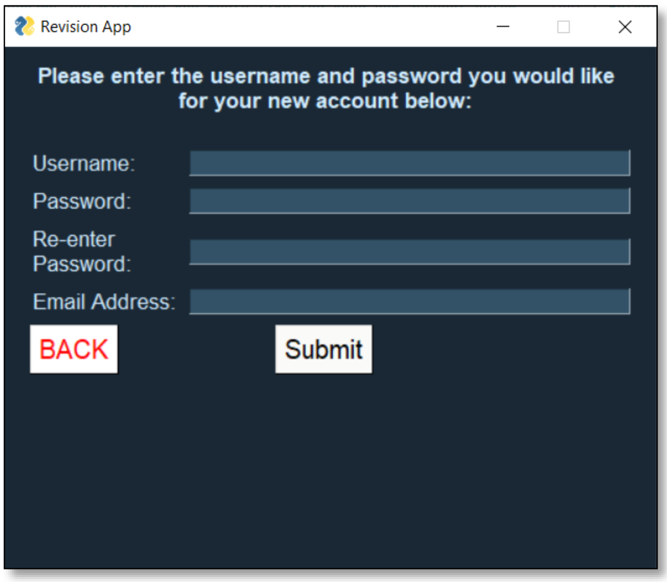
I have next added another textual user input field for password re-entry. The program will check whether the 2 passwords the user has entered are the same and if they are (and there

are no other invalid details entered) their account will be successfully created. This will be implemented later.
This is the code I added (highlighted) to create the extra textual input box for the user to re-enter their desired password:



```
Signup = [[sg.Text('Please enter the username and password you would like for your new account below:', font=("Helvetica", 13, "bold"), justification='center', size=(45,3))],
          # Creates a block of text at the top of the screen which is centralised and is the same size and font as the block of text on the Start Screen.
          [sg.Text("Username:", size=(12,1), font=("Helvetica", 13)), sg.InputText(key='newusername', size=(100,2))],
          [sg.Text("Password:", size=(12,1), font=("Helvetica", 13)), sg.InputText(key='newpassword', password_char='*', size=(100,2))],
          [sg.Text("Re-enter Password:", size=(12,2), font=("Helvetica", 13)), sg.InputText(key='password_re-entry', password_char='*', size=(100,2))],
          [sg.Text("Email Address:", size=(12,1), font=("Helvetica", 13)), sg.InputText(key='newemail', size=(100,2))],
          # Creates 4 labelled input boxes for the user to enter their username/password/password re-entry/email address, which can then be validated.
          [sg.Button('BACK', font=("Helvetica", 15), key='BACK', button_color=('red', 'white')), sg.Text('', size=(12,2)), sg.Button('Submit', key='NewAccount', font=("Helvetica", 15))]]
#          Creates a red back button (to take the user back to the Start Screen) and a submit button for the user to validate the details they entered and create their account.
```

The result was the SIGNUP interface now appearing like this:

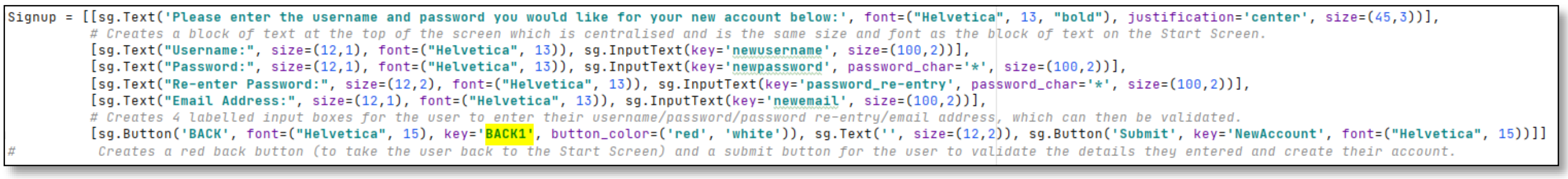

Please enter the username and password you would like for your new account below:

Sign Up – Stage 2

I have next created the file 'userinfo.txt' and placed it in the correct directory so it can be referred to in my program.

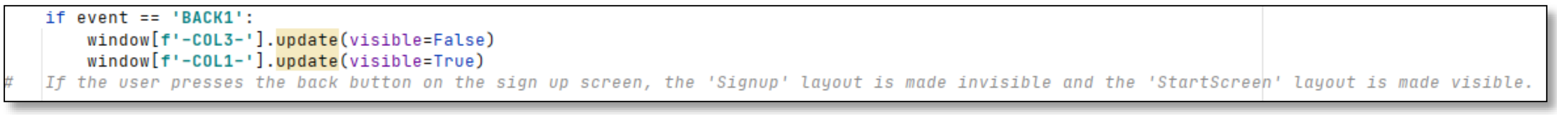
userinfo.txt

I am going to start by creating the functionality of the back button. I have first changed the key of the back button on the SIGNUP layout as there will be many back buttons throughout my program and each will need their own unique key as they will all update the layout in different ways.



```
Signup = [[sg.Text('Please enter the username and password you would like for your new account below:', font=("Helvetica", 13, "bold"), justification='center', size=(45,3))],
          # Creates a block of text at the top of the screen which is centralised and is the same size and font as the block of text on the Start Screen.
          [sg.Text("Username:", size=(12,1), font=("Helvetica", 13)), sg.InputText(key='newusername', size=(100,2))],
          [sg.Text("Password:", size=(12,1), font=("Helvetica", 13)), sg.InputText(key='newpassword', password_char='*', size=(100,2))],
          [sg.Text("Re-enter Password:", size=(12,2), font=("Helvetica", 13)), sg.InputText(key='password_re-entry', password_char='*', size=(100,2))],
          [sg.Text("Email Address:", size=(12,1), font=("Helvetica", 13)), sg.InputText(key='newemail', size=(100,2))],
          # Creates 4 labelled input boxes for the user to enter their username/password/password re-entry/email address, which can then be validated.
          [sg.Button('BACK', font=("Helvetica", 15), key='BACK1', button_color=('red', 'white')), sg.Text('', size=(12,2)), sg.Button('Submit', key='NewAccount', font=("Helvetica", 15))]]
#          Creates a red back button (to take the user back to the Start Screen) and a submit button for the user to validate the details they entered and create their account.
```

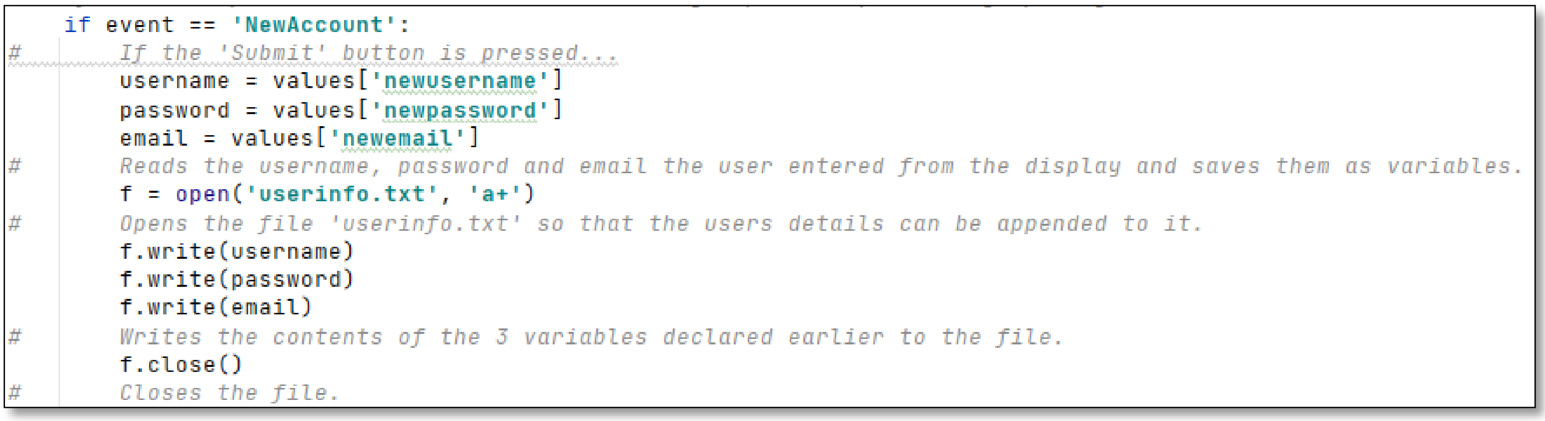
I then wrote the following piece of code in the event loop which updates the layout when the back button is pressed (by making the 'Signup' layout visible and the 'StartScreen' layout visible):



```
    if event == 'BACK1':
        window[f'-COL3-'].update(visible=False)
        window[f'-COL1-'].update(visible=True)
#   If the user presses the back button on the sign up screen, the 'Signup' layout is made invisible and the 'StartScreen' layout is made visible.
```

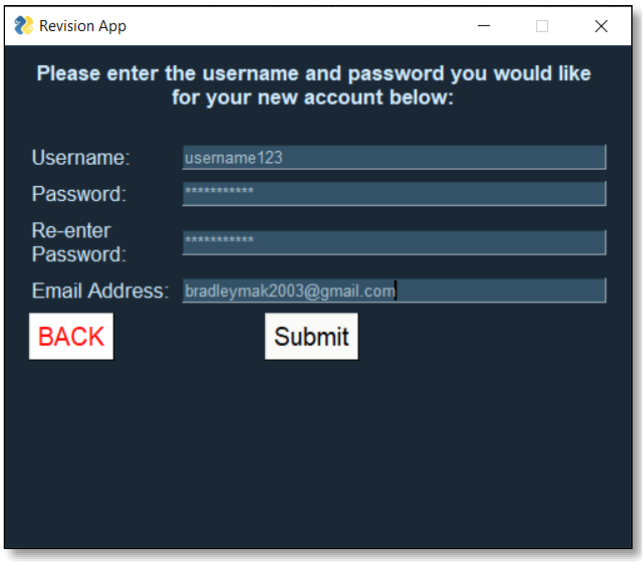
The back button now works as intended. This is not included as part of my iterative testing however as it is a simple feature that will be repeated multiple times throughout the program so I didn't think there was much point testing it all the time.

I will next implement the functionality of the submit button.
I have started by writing the following code which takes the users inputs and writes them to the userinfo.txt file. The program does not yet validate the inputs or check the password re-entry.



```
    if event == 'NewAccount':
#       If the 'Submit' button is pressed...
        username = values['newusername']
        password = values['newpassword']
        email = values['newemail']
#       Reads the username, password and email the user entered from the display and saves them as variables.
        f = open('userinfo.txt', 'a+')
#       Opens the file 'userinfo.txt' so that the users details can be appended to it.
        f.write(username)
        f.write(password)
        f.write(email)
#       Writes the contents of the 3 variables declared earlier to the file.
        f.close()
#       Closes the file.
```

The file is opened to append (rather than write) so that nothing is overwritten and errors are avoided.
I tested my program using the inputs stated in test 2.2.1:

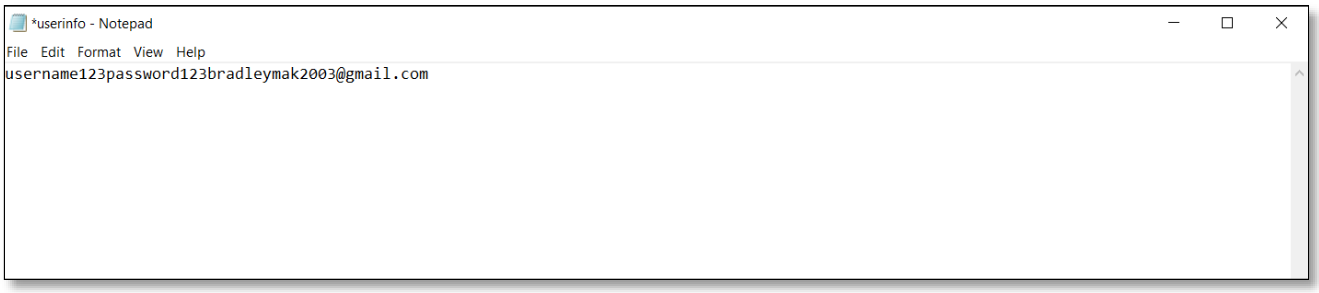


Revision App

Please enter the username and password you would like for your new account below:

Username: username123
Password: ***********
Re-enter Password: ***********
Email Address: bradleymak2003@gmail.com

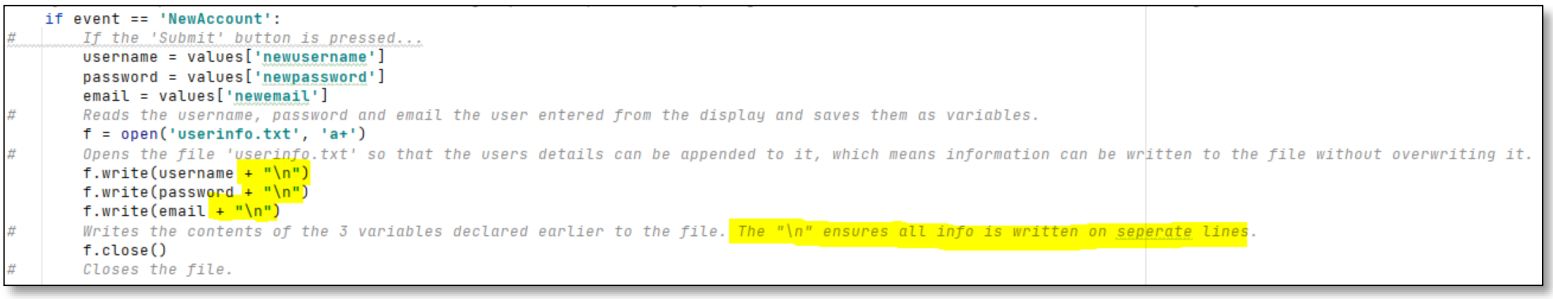
And the following was written to 'userinfo.txt':



*userinfo - Notepad

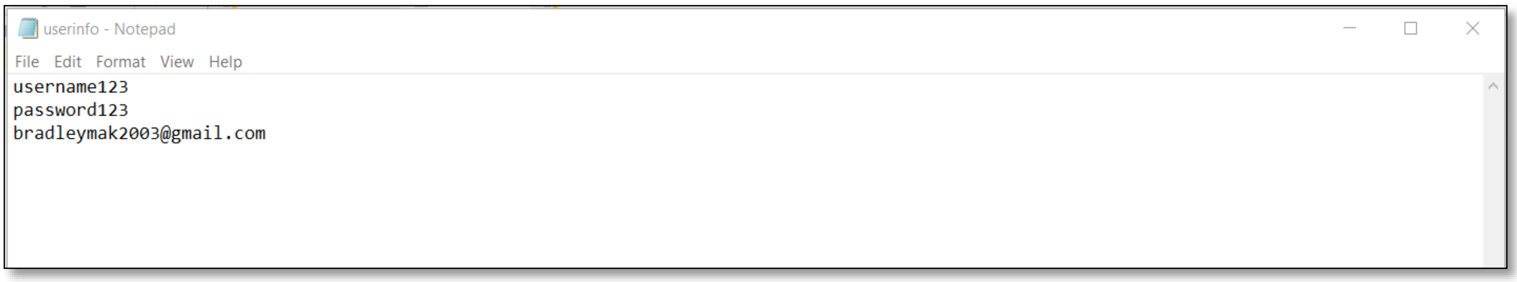
```
username123password123bradleymak2003@gmail.com
```

The issue here is that all of the inputs are being written one after another on the same line, which is an issue as my program needs to be able to read information from the file line by line.
To attempt to fix the issue, I used string manipulation to add a line break to every user input before it is written to the file.



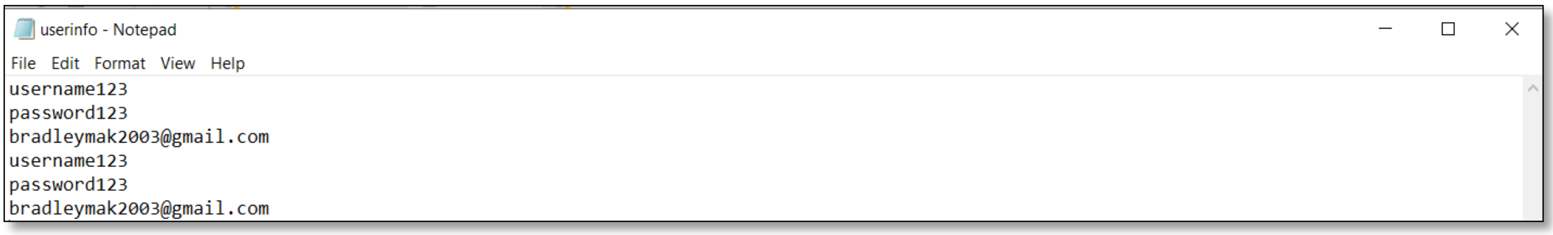
```
    if event == 'NewAccount':
#       If the 'Submit' button is pressed...
        username = values['newusername']
        password = values['newpassword']
        email = values['newemail']
#       Reads the username, password and email the user entered from the display and saves them as variables.
        f = open('userinfo.txt', 'a+')
#       Opens the file 'userinfo.txt' so that the users details can be appended to it, which means information can be written to the file without overwriting it.
        f.write(username + "\n")
        f.write(password + "\n")
        f.write(email + "\n")
#       Writes the contents of the 3 variables declared earlier to the file. The "\n" ensures all info is written on seperate lines.
        f.close()
#       Closes the file.
```

I manually cleared the text file and then re-ran the program with the same inputs and the result was this:



```
username123
password123
bradleymak2003@gmail.com
```

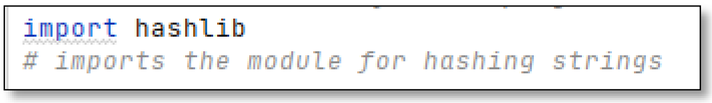
To test that it worked after more than one submission, I pressed the submit button again and the information entered was again written correctly to the file on separate lines as intended:



```
username123
password123
bradleymak2003@gmail.com
username123
password123
bradleymak2003@gmail.com
```

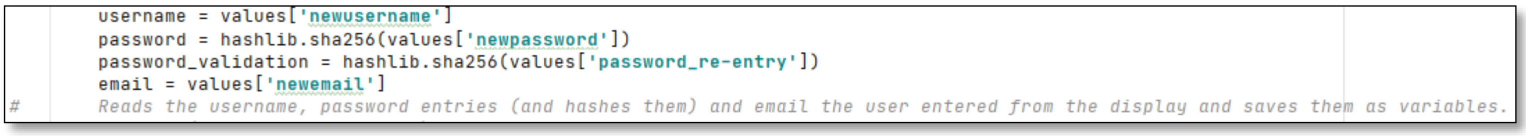
I had originally planned for each user to have their own unique user ID (which would just be an integer), however since every user has a unique username (which will be checked when they create their account), there is not really much need for the user ID. Therefore, I am not going to include it as it will just take more processing time to produce a unique user ID for each user and will take up more storage space. I am just going to use the users username as their unique identifier instead. This will not change the view of the app from the users end, it just makes the implementation simpler and more space and time efficient (hence it is an example of abstraction being used in my development).
In the userinfo.txt file, users' information will now be stored in blocks of 3 rather than 4 like originally planned (as I have reduced the amount of information that needs to be stored for each user).

Now that the program is correctly storing all of the entered information, the password needs to be hashed before it is stored in order to again increase the security of my app and avoid any serious data breaches.

First, I imported hashlib: 

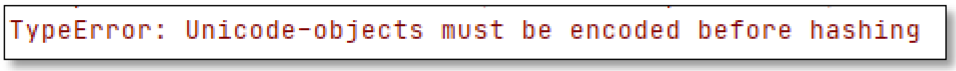
```
import hashlib
# imports the module for hashing strings
```

I then hashed the two



```
        username = values['newusername']
        password = hashlib.sha256(values['newpassword'])
        password_validation = hashlib.sha256(values['password_re-entry'])
        email = values['newemail']
#       Reads the username, password entries (and hashes them) and email the user entered from the display and saves them as variables.
```

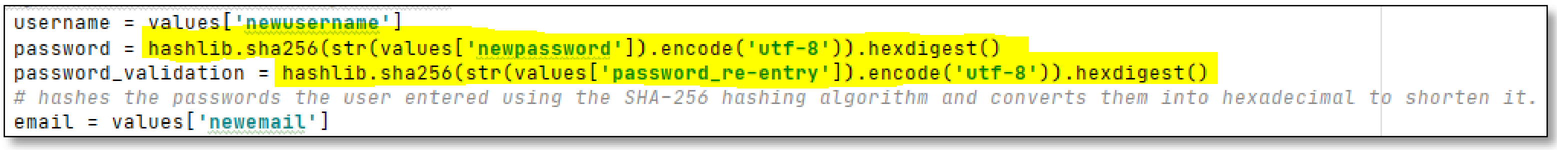
password entries using the SHA-256 hashing algorithm:
Then, when I ran the program with the inputs in test 2.2.1, I received the following error:



```
TypeError: Unicode-objects must be encoded before hashing
```

I had a look online as this is something I do not have experience coding before, and I found the following website: https://stackoverflow.com/questions/7585307/how-to-correct-typeerror-unicode-objects-must-be-encoded-before-hashing.
It explained that it has to be encoded into the 8 bit Unicode format before being hashed because the hashing algorithm uses Unicode to produce the hash value (as mathematical operations can be performed on Unicode).

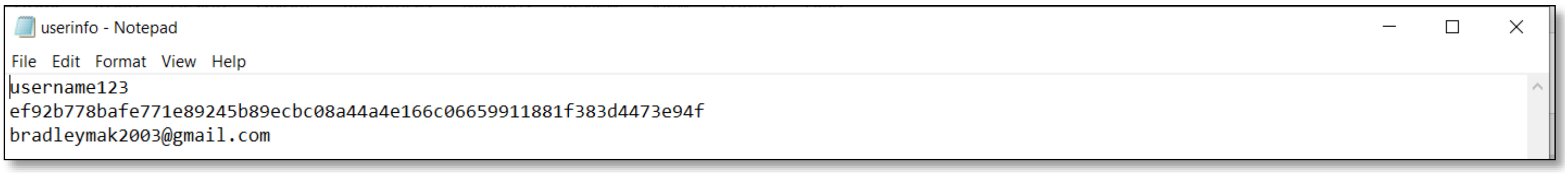
Using this advice, I updated my code to now look like this:



```
username = values['newusername']
password = hashlib.sha256(str(values['newpassword']).encode('utf-8')).hexdigest()
password_validation = hashlib.sha256(str(values['password_re-entry']).encode('utf-8')).hexdigest()
# hashes the passwords the user entered using the SHA-256 hashing algorithm and converts them into hexadecimal to shorten it.
email = values['newemail']
```

The passwords the user entered have both been ran through the SHA-256 hashing algorithm to produce a fixed length hash value specific to that password. It is only the hash value which is stored in the file rather than the actual password which makes my app much more secure because even if there was a data breach, the hashed values cannot be reversed back into their original form using the hashing algorithm. The passwords have then been converted into hexadecimal to shorten them and thus reduce the amount of storage space needed.

I then ran the program again with the inputs in test 2.2.1, and the result was the following being written to the file 'userinfo.txt':

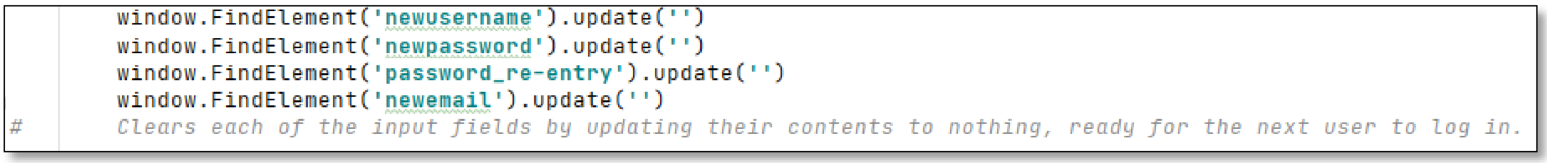


```
username123
ef92b778bafe771e89245b89ecbc08a44a4e166c06659911881f383d4473e94f
bradleymak2003@gmail.com
```

Therefore, the user inputs are now successfully being written to 'userinfo.txt' in the correct form.

Once the user has entered their details, the input fields need to be cleared ready for the next user (as it would be insecure to leave the input fields filled with the previous users log in details).

To do this, I have used each of the input fields IDs to locate them and update their contents to contain nothing (so that they appear blank to the next user that wishes to log in on the same device):



```
        window.FindElement('newusername').update('')
        window.FindElement('newpassword').update('')
        window.FindElement('password_re-entry').update('')
        window.FindElement('newemail').update('')
#       Clears each of the input fields by updating their contents to nothing, ready for the next user to log in.
```

I will now test this feature to check whether it is working (testing evidence for tests 2.2.1 and 2.2.2 can be found in the testing evidence PowerPoint).

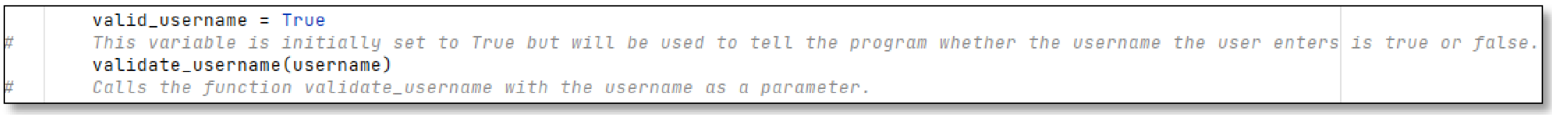
Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
2.2.1	Entering username, password (twice) and email into relevant fields	String ‘username123’ String ‘password123’ String ‘password123’ String ‘bradleymak2003 @gmail.com’ **VALID DATA**	App will allow user to enter these strings into the relevant input boxes. The password should be dotted.	Checking whether the user can enter a string into the textual input boxes that should appear on screen.	**The program allowed me to enter these details into the available input boxes.**
2.2.2	Selection of button to confirm account details	Mouse click	Entry fields will clear and strings will be written to ‘userinfo.txt’, but app will remain on the same screen.	To check whether details entered into the input boxes by the user are stored to an external file via a button press so they can be referenced at any later date.	**The information I entered was successfully written to the file , the passwords were hashed and the input boxes were emptied.**

Sign up – Stage 3

At this stage, I am going to set up the validation of the user inputs.

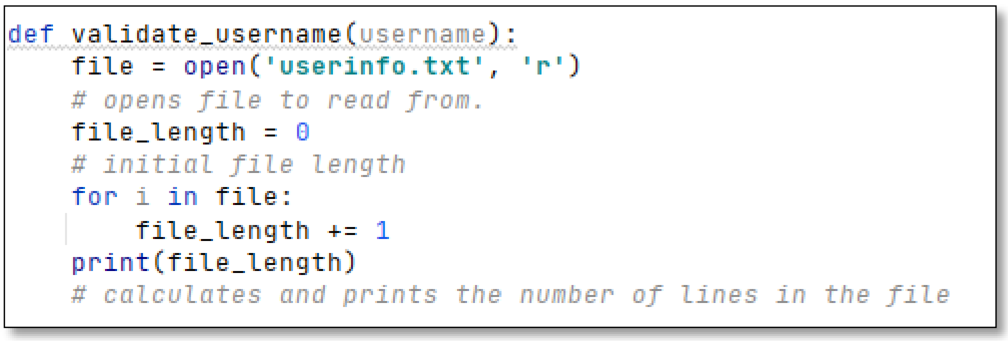
To start, I am going to check whether the username the user enters has been used before (as it has to be unique as it is the users unique identifier) – **username verification**. I have created a boolean variable called valid_username which starts as True. This variable will be updated to false if the username the user enters is already present in ‘userinfo.txt’.

I next need to sequentially search through the file to check whether the username the user entered has been used before. To do this, instead of using the num_of_accounts variable I proposed earlier in the design section, I am going to find the length of ‘userinfo.txt’ and then use that value in a for loop. This is because keeping the num_of_accounts variable up to date would require it to be saved/re-written to a file every time an account was made/deleted, which wouldn’t be time efficient and would be more difficult to implement. I will create a validate_username subroutine which will take the users username as a parameter, and will check whether that username is present in ‘userinfo.txt’. If it is, valid_username will be updated to be false and an error message will appear.



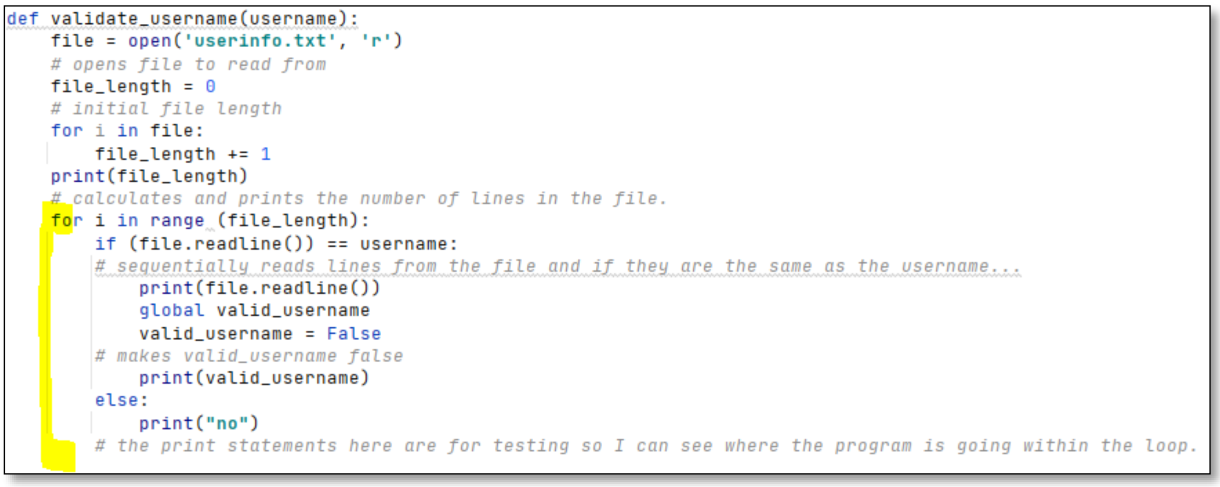
```
    valid_username = True
#   This variable is initially set to True but will be used to tell the program whether the username the user enters is true or false.
    validate_username(username)
#   Calls the function validate_username with the username as a parameter.
```

Initially, I started by calculating the number of lines in the file using the following piece of code:



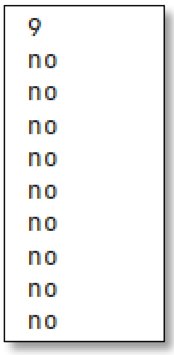
```
def validate_username(username):
    file = open('userinfo.txt', 'r')
    # opens file to read from.
    file_length = 0
    # initial file length
    for i in file:
        file_length += 1
    print(file_length)
    # calculates and prints the number of lines in the file
```

I then added the following code to attempt to test the subroutine to see if it worked correctly:

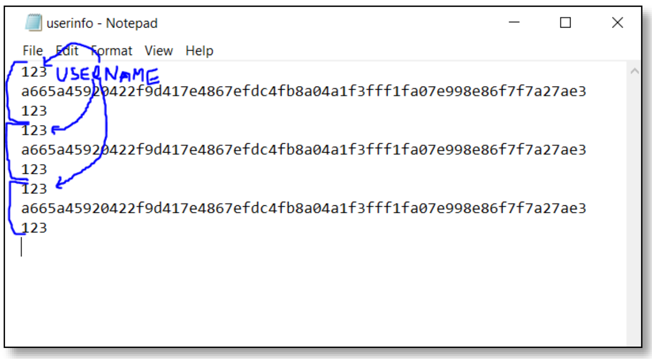


```
def validate_username(username):
    file = open('userinfo.txt', 'r')
    # opens file to read from
    file_length = 0
    # initial file length
    for i in file:
        file_length += 1
    print(file_length)
    # calculates and prints the number of lines in the file.
    for i in range (file_length):
        if (file.readline()) == username:
        # sequentially reads lines from the file and if they are the same as the username...
            print(file.readline())
            global valid_username
            valid_username = False
        # makes valid_username false
            print(valid_username)
        else:
            print("no")
        # the print statements here are for testing so I can see where the program is going within the loop.
```

When I ran this prototype multiple times using '123' for every input field, this was what was being printed by the print statements, as well as the contents of the file at this point:



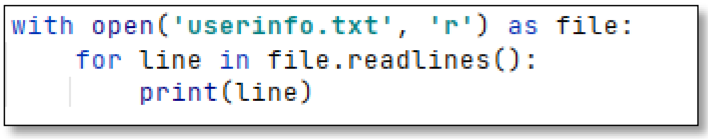
```
9
no
no
no
no
no
no
no
no
no
```



```
123
a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3
123
123
a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3
123
123
a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3
123
```

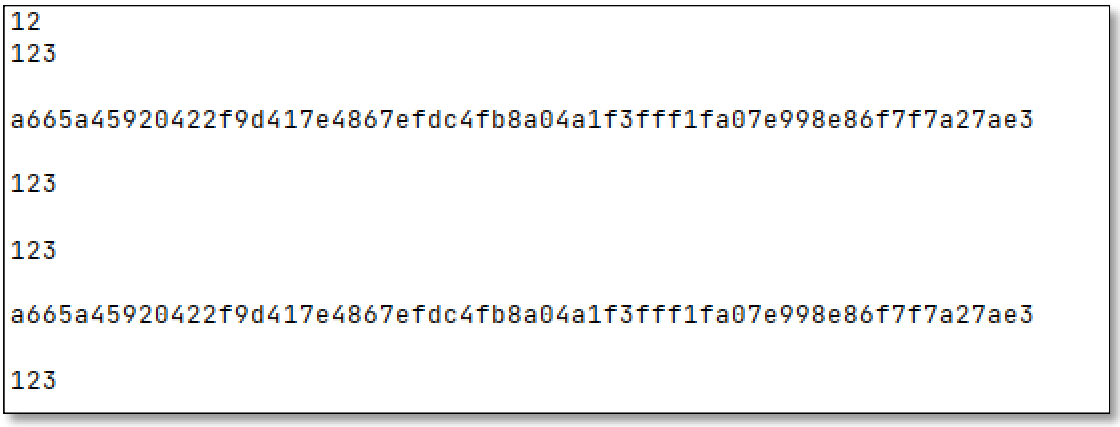
As you can see, the '9' output is correct for the length of the file, but the program is outputting 'no' despite the fact that the username entered is clearly already present in the file (and thus the output should be 'False'). So I assumed that the program is not reading the lines from the file as I intended (as the lines from the file are not being printed by the print statement).

I had a look at the forum https://stackoverflow.com/questions/28873349/python-readlines-not-returning-anything, which helped me with my next steps.
I wrote the following piece of code which outputs all of the lines of data in the file:



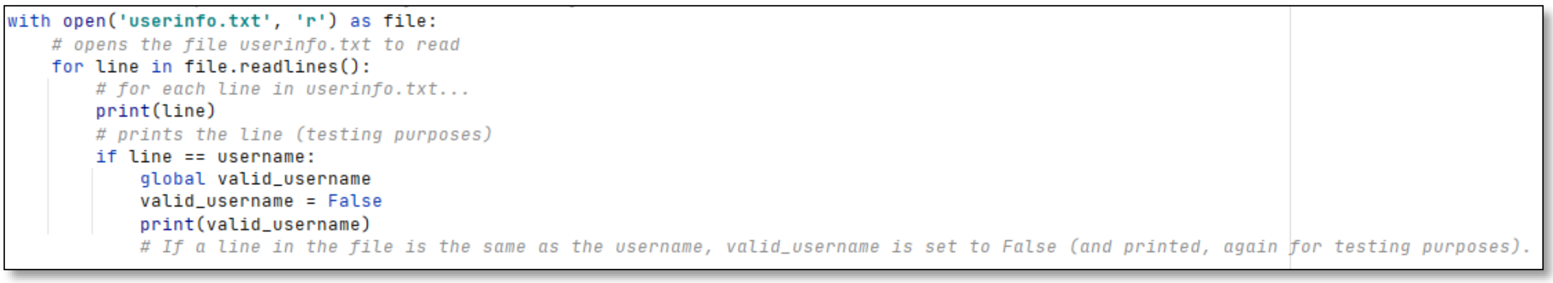
```
with open('userinfo.txt', 'r') as file:
    for line in file.readlines():
        print(line)
```

When I ran the code, all of the lines of data in the file were outputted:



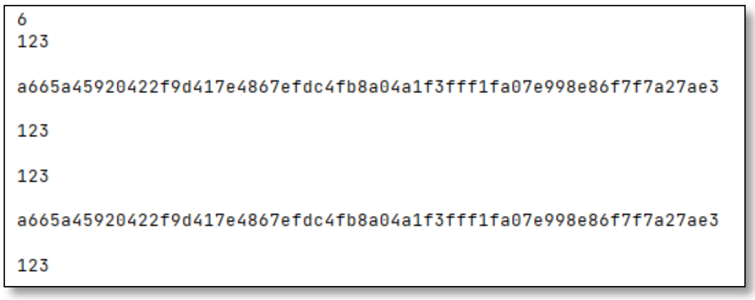
```
12
123

a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3

123

123

a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3

123
```

So, now that I have found a way to extract each individual line from the file, I can use the same logic as I did in the previous prototype to compare each of the extracted lines to the username:



```
with open('userinfo.txt', 'r') as file:
    # opens the file userinfo.txt to read
    for line in file.readlines():
        # for each line in userinfo.txt...
        print(line)
        # prints the line (testing purposes)
        if line == username:
            global valid_username
            valid_username = False
            print(valid_username)
            # If a line in the file is the same as the username, valid_username is set to False (and printed, again for testing purposes).
```

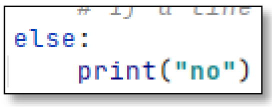
I then manually removed the data entered during the test of the previous prototype from the file to allow for a fair test.



```
6
123

a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3

123

123

a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3

123
```

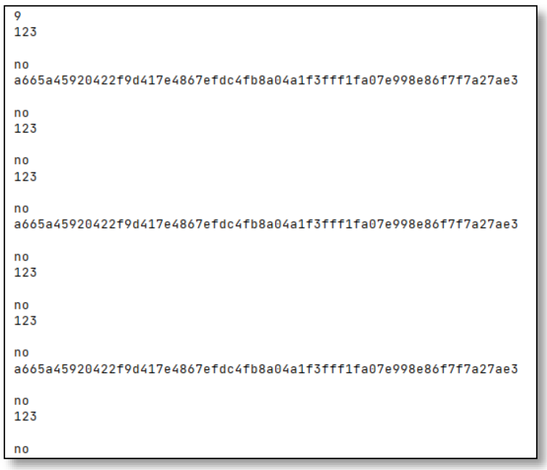
When I run this prototype (again using ‘123’ as the input for every field), this is what gets outputted:
This is the same output as the previous prototype, hence the program is still not detecting that there is data in the file that is the same as the username (when there clearly is).

To further test this prototype, I add an else statement to check whether the program is actually even running the if statement in the first place:

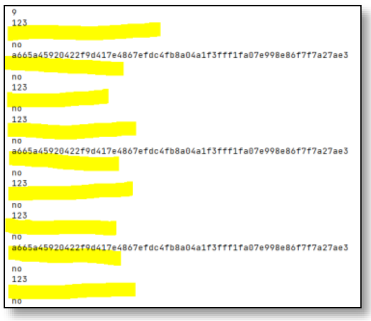


```
else:
    print("no")
```

When I run the prototype again this time, I get the following output:


```
9
123

no
a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3

no
123

no
123

no
a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3

no
123

no
123

no
a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3

no
123

no
```

As you can see, ‘no’ is being outputted after every iteration, so the program must be entering the loop and the comparison is just not working correctly.

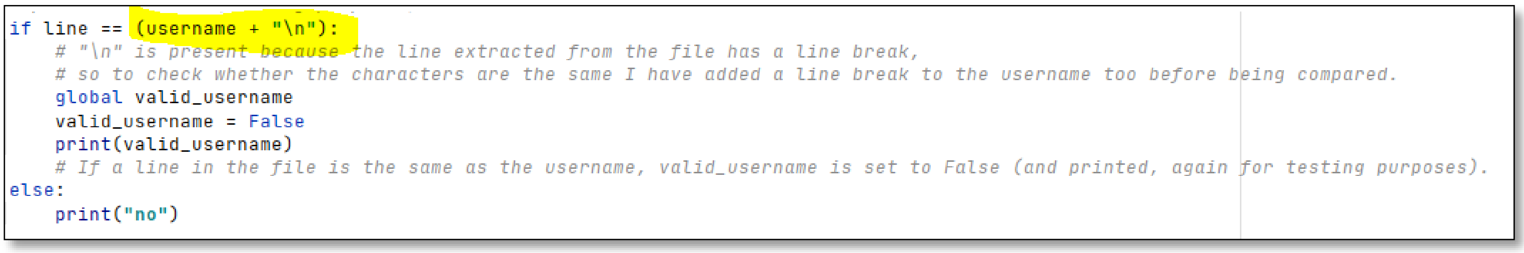


```
9
123

no
a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3

no
123

no
123

no
a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3

no
123

no
123

no
a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3

no
123

no
```

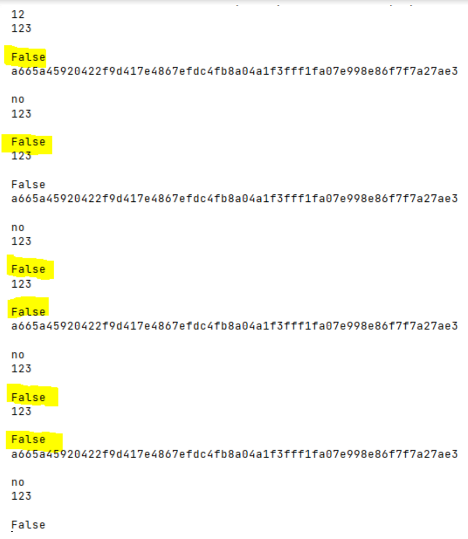
After thinking about this for some time, I realised that there is a line break in the outputs after every line extracted from the file (highlighted in yellow):

This must be why the comparison is not working – the username does not have the line break whereas the line extracted from the file does, so they will never be considered ‘equal’ even if they appear it.

To fix this, I have added a line break to the username before the comparison, which should ensure that if the characters in the line extracted and the username are the same, they are considered the same by the program because they will both have the line break:


```
if line == (username + "\n"):
    # "\n" is present because the line extracted from the file has a line break,
    # so to check whether the characters are the same I have added a line break to the username too before being compared.
    global valid_username
    valid_username = False
    print(valid_username)
    # If a line in the file is the same as the username, valid_username is set to False (and printed, again for testing purposes).
else:
    print("no")
```

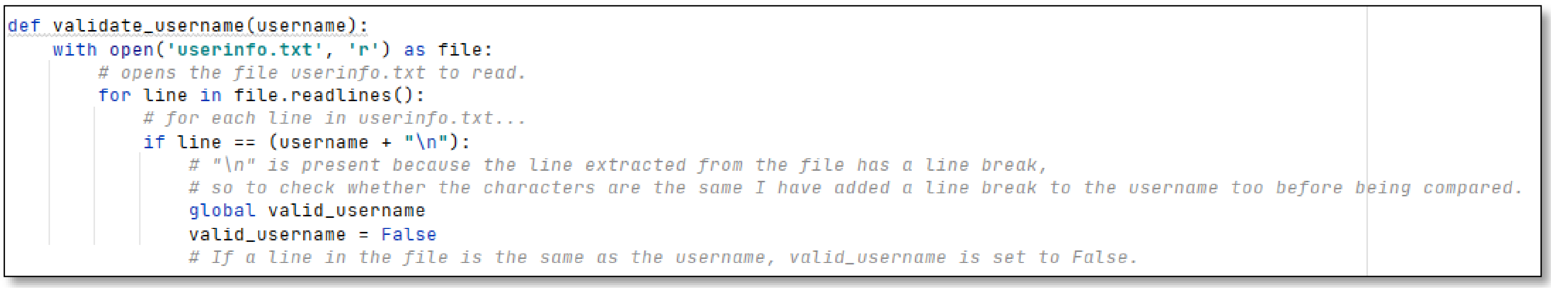
When I ran the new prototype, I received the following output:


```
12
123

False
a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3

no
123

False
123

False
a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3

no
123

False
123

False
a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3

no
123

False
123

False
a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3

no
123

False
```

As you can see, ‘False’ (the new Boolean value of valid_username) is being outputted because there is a string/line of data in the file which matches the username (on multiple occasions in this case).

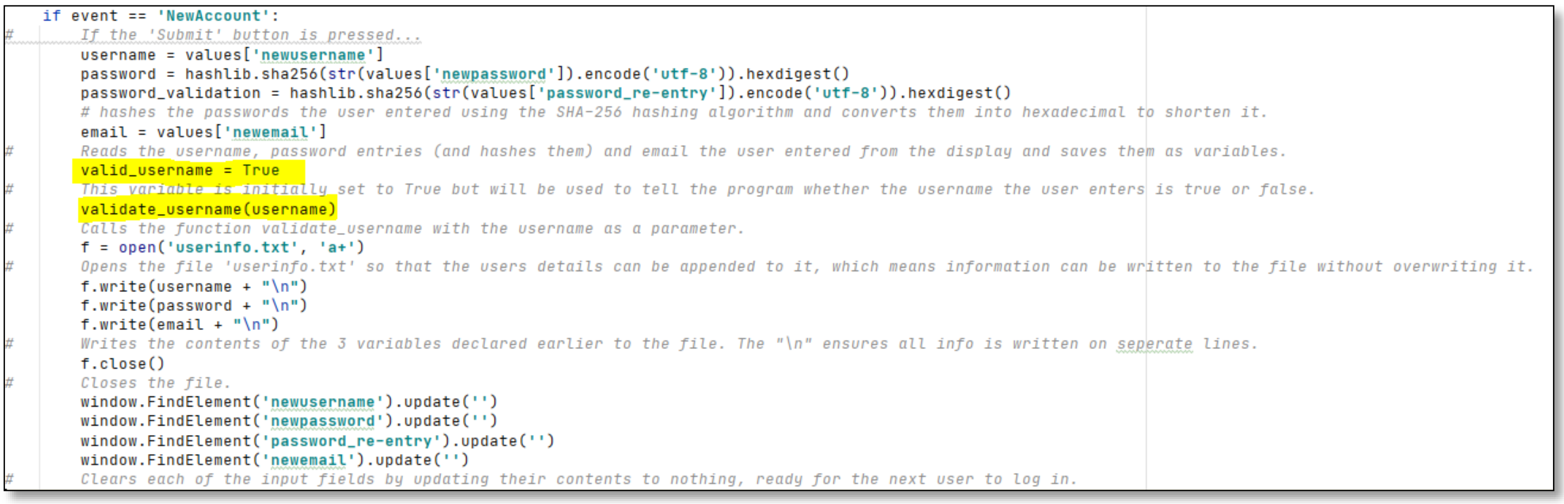
This now means that the user is successfully detecting whether there is a line of data in the file which matches the username (hence suggesting the username is a duplicate, and should not be accepted).

I also now have no need to determine the number of lines in the file as this is not required in the new prototype, so I can remove that piece of code. I can also remove the print statements I used for testing. Therefore, the validate_username subroutine now looks like this:



```python
def validate_username(username):
    with open('userinfo.txt', 'r') as file:
        # opens the file userinfo.txt to read.
        for line in file.readlines():
            # for each line in userinfo.txt...
            if line == (username + "\n"):
                # "\n" is present because the line extracted from the file has a line break,
                # so to check whether the characters are the same I have added a line break to the username too before being compared.
                global valid_username
                valid_username = False
                # If a line in the file is the same as the username, valid_username is set to False.
```

I also moved the calling of the subroutine to BEFORE the point at which the information was written to the file (as it needs to be validated before being added):

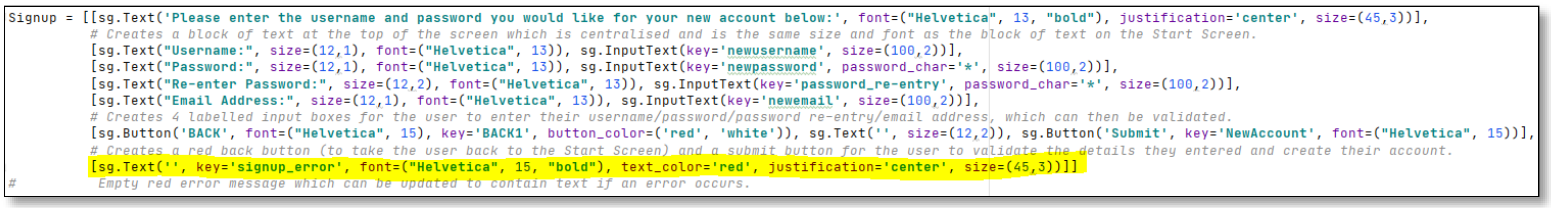


```python
    if event == 'NewAccount':
#       If the 'Submit' button is pressed...
        username = values['newusername']
        password = hashlib.sha256(str(values['newpassword']).encode('utf-8')).hexdigest()
        password_validation = hashlib.sha256(str(values['password_re-entry']).encode('utf-8')).hexdigest()
        # hashes the passwords the user entered using the SHA-256 hashing algorithm and converts them into hexadecimal to shorten it.
        email = values['newemail']
#       Reads the username, password entries (and hashes them) and email the user entered from the display and saves them as variables.
        valid_username = True
#       This variable is initially set to True but will be used to tell the program whether the username the user enters is true or false.
        validate_username(username)
#       Calls the function validate_username with the username as a parameter.
        f = open('userinfo.txt', 'a+')
#       Opens the file 'userinfo.txt' so that the users details can be appended to it, which means information can be written to the file without overwriting it.
        f.write(username + "\n")
        f.write(password + "\n")
        f.write(email + "\n")
#       Writes the contents of the 3 variables declared earlier to the file. The "\n" ensures all info is written on seperate lines.
        f.close()
#       Closes the file.
        window.FindElement('newusername').update('')
        window.FindElement('newpassword').update('')
        window.FindElement('password_re-entry').update('')
        window.FindElement('newemail').update('')
#       Clears each of the input fields by updating their contents to nothing, ready for the next user to log in.
```

I decided to double check whether this was truly working, so I manually cleared 'userinfo.txt' (so that it contained nothing and thus on the first input, it cannot possibly be a duplicate) and added a print(valid_username) statement just underneath the calling of the subroutine so that I could see the final value of valid_username.
I inputted information 3 times:
First input – '123' for all fields
Second input – '987' for all fields
Third input – '123' for all fields
This will allow me to check whether the subroutine is working correctly. This test can be found in the testing evidence powerpoint under 'Sign Up – Stage 3 Test'. The outputs I received were the following:
First input – True
Second input – True
Third input – False

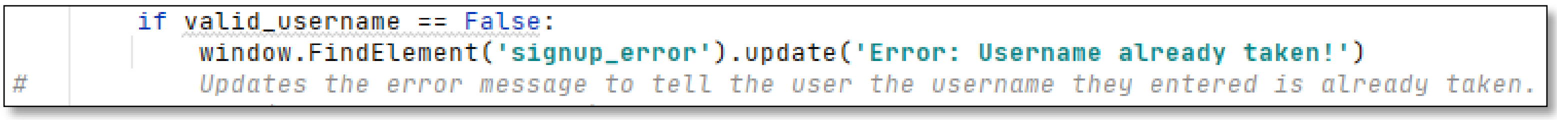
This is absolutely correct as initially, '123' and '987' are unique usernames and so should be accepted, but on the third input, the username '123' has already been taken and so valid_username should have been set to False which it has been.

Now that I have completed the username verification subroutine, I am going to create an error message which should appear if a username is already taken.
I have started by creating a text element on the Signup display which is initially empty but can be updated to contain the relevant text and will appear centralised in bold, red text to signify an error.



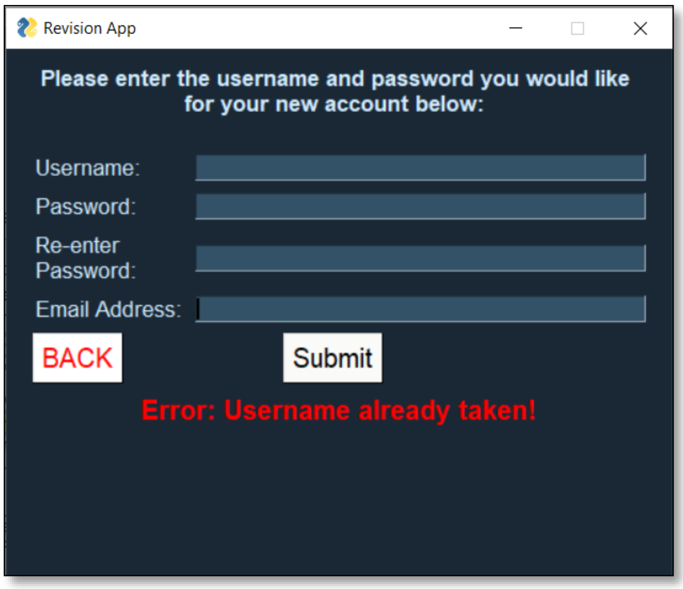
```
Signup = [[sg.Text('Please enter the username and password you would like for your new account below:', font=("Helvetica", 13, "bold"), justification='center', size=(45,3))],
          # Creates a block of text at the top of the screen which is centralised and is the same size and font as the block of text on the Start Screen.
          [sg.Text("Username:", size=(12,1), font=("Helvetica", 13)), sg.InputText(key='newusername', size=(100,2))],
          [sg.Text("Password:", size=(12,1), font=("Helvetica", 13)), sg.InputText(key='newpassword', password_char='*', size=(100,2))],
          [sg.Text("Re-enter Password:", size=(12,2), font=("Helvetica", 13)), sg.InputText(key='password_re-entry', password_char='*', size=(100,2))],
          [sg.Text("Email Address:", size=(12,1), font=("Helvetica", 13)), sg.InputText(key='newemail', size=(100,2))],
          # Creates 4 labelled input boxes for the user to enter their username/password/password re-entry/email address, which can then be validated.
          [sg.Button('BACK', font=("Helvetica", 15), key='BACK1', button_color=('red', 'white')), sg.Text('', size=(12,2)), sg.Button('Submit', key='NewAccount', font=("Helvetica", 15))],
          # Creates a red back button (to take the user back to the Start Screen) and a submit button for the user to validate the details they entered and create their account.
          [sg.Text('', key='signup_error', font=("Helvetica", 15, "bold"), text_color='red', justification='center', size=(45,3))]]
#          Empty red error message which can be updated to contain text if an error occurs.
```

I have then created the code which updates the element to contain the error message if the username is already taken (and so valid_username = False):

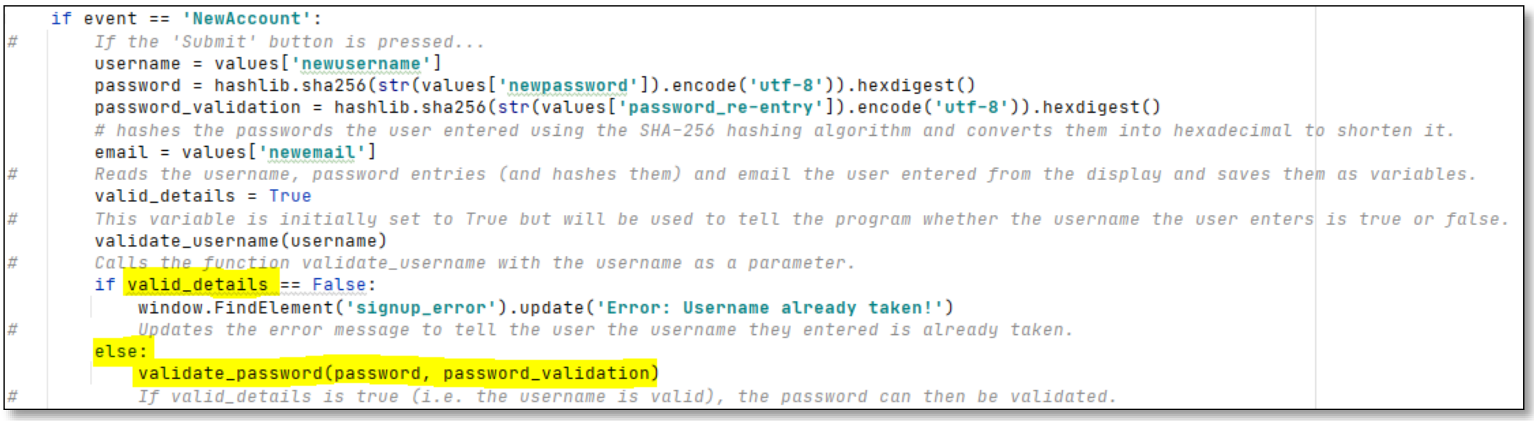


```
        if valid_username == False:
            window.FindElement('signup_error').update('Error: Username already taken!')
#           Updates the error message to tell the user the username they entered is already taken.
```

When a username already present in the file is entered, the following error message appears on the screen:

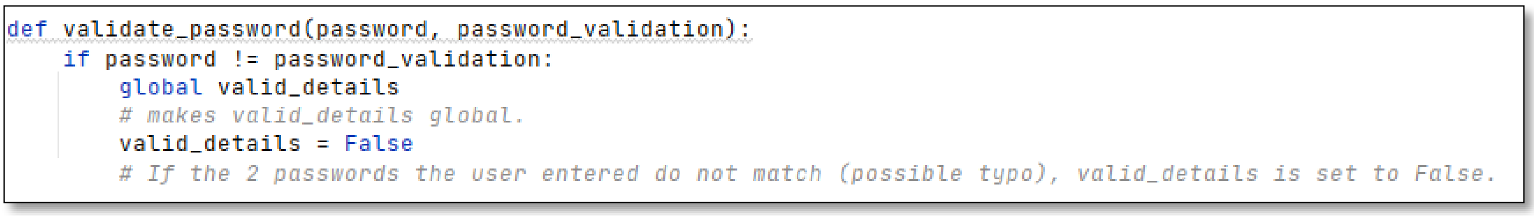

Revision App
Please enter the username and password you would like for your new account below:
Username:
Password:
Re-enter Password:
Email Address:
BACK
Submit
Error: Username already taken!

Now that the username validation is in place, I can move onto **password validation**. I am going to approach this slightly differently than in my original pseudocode. Instead of using a nested loop, I am going to use a separate subroutine for each part of the validation process (i.e. validating username, password and email). I believe this will make the development process easier, allow for quicker error checking and correction, and easier maintenance of the code in the long term for any future updates.
To start with, I have changed the valid_username variable to valid_details throughout my program. This is to better the programs space efficiency as I will not need a different variable for each entry, I can just use this single variable throughout the validation process and check if it stays true.
If valid_details is true after the username has been validated, the subroutine validate_password will be called, with the parameters password and password_verification. I will create this subroutine next.



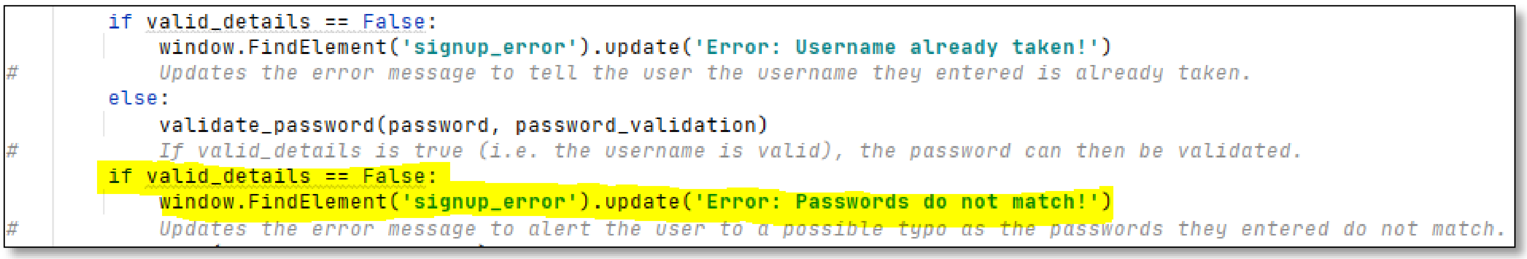
```
    if event == 'NewAccount':
#       If the 'Submit' button is pressed...
        username = values['newusername']
        password = hashlib.sha256(str(values['newpassword']).encode('utf-8')).hexdigest()
        password_validation = hashlib.sha256(str(values['password_re-entry']).encode('utf-8')).hexdigest()
        # hashes the passwords the user entered using the SHA-256 hashing algorithm and converts them into hexadecimal to shorten it.
        email = values['newemail']
#       Reads the username, password entries (and hashes them) and email the user entered from the display and saves them as variables.
        valid_details = True
#       This variable is initially set to True but will be used to tell the program whether the username the user enters is true or false.
        validate_username(username)
#       Calls the function validate_username with the username as a parameter.
        if valid_details == False:
            window.FindElement('signup_error').update('Error: Username already taken!')
#           Updates the error message to tell the user the username they entered is already taken.
        else:
            validate_password(password, password_validation)
#           If valid_details is true (i.e. the username is valid), the password can then be validated.
```

This is the validate_password subroutine, which is required only to check whether the 2 passwords the user entered are the same. This is to avoid a possible typo as the user cannot actually see their password as password characters are being used in their place to make the app more secure and avoid shouldering, as mentioned earlier.



```
def validate_password(password, password_validation):
    if password != password_validation:
        global valid_details
        # makes valid_details global.
        valid_details = False
        # If the 2 passwords the user entered do not match (possible typo), valid_details is set to False.
```

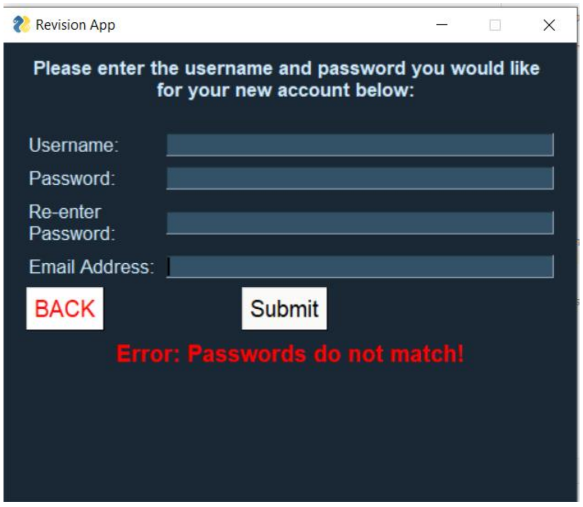
I have then made it so that the error message is updated to alert the user to this issue:



```
        if valid_details == False:
            window.FindElement('signup_error').update('Error: Username already taken!')
#           Updates the error message to tell the user the username they entered is already taken.
        else:
            validate_password(password, password_validation)
#           If valid_details is true (i.e. the username is valid), the password can then be validated.
        if valid_details == False:
            window.FindElement('signup_error').update('Error: Passwords do not match!')
#           Updates the error message to alert the user to a possible typo as the passwords they entered do not match.
```

This should work as the program goes through the code in logical order (from top to bottom, so for example the username will always be validated before the password).

I then manually cleared 'userinfo.txt' to ensure that the username I enter will be valid (as I am focusing on password verification), and entered two different passwords ('password12' and 'password123') into the 2 password input boxes, and the error message was correctly displayed:



Revision App

Please enter the username and password you would like for your new account below:

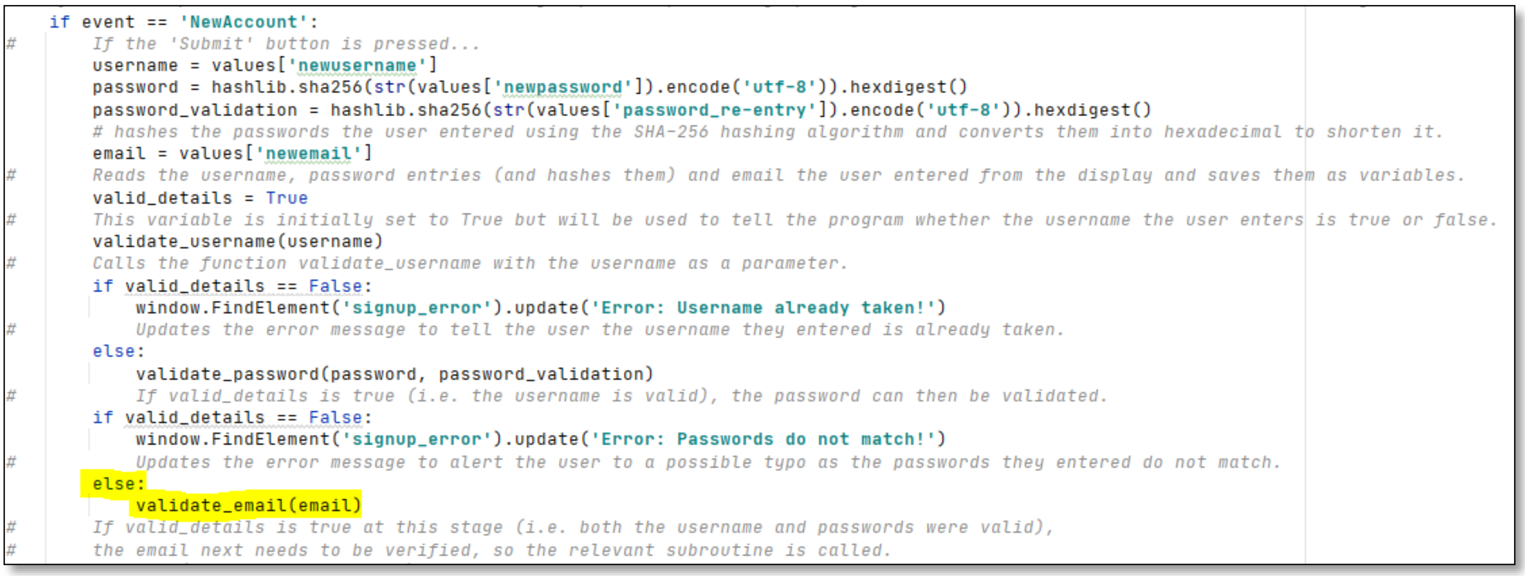
Username:
Password:
Re-enter Password:
Email Address:

Error: Passwords do not match!

(This feature will be tested again in the later iterations of testing and testing evidence of this will then be put on the testing evidence powerpoint).

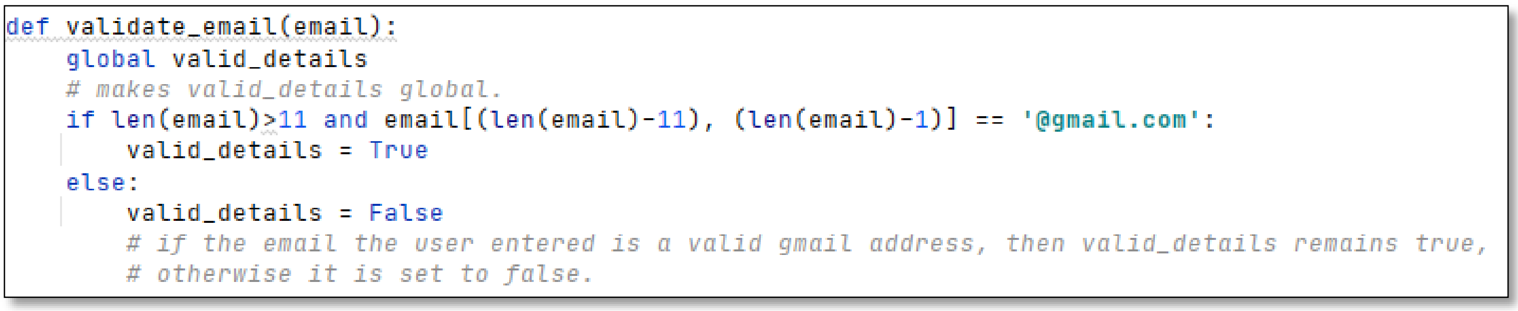
At this stage, the input boxes are being cleared when there is an error (which I do not want to happen), but I will fix this later when I have completed all of the verification.

Next, I will move onto **email verification**.
Users emails will be required to be gmail (to be able to use the email notifications feature). Whilst this limitation is still subject to change, at this moment in time emails would have to include the substring '@gmail.com', and would thus have to also be longer than 10 characters long.
If valid_details is still true after both the username and password have been validated, the email needs to be validated so the subroutine (which is yet to be defined) needs to be called:



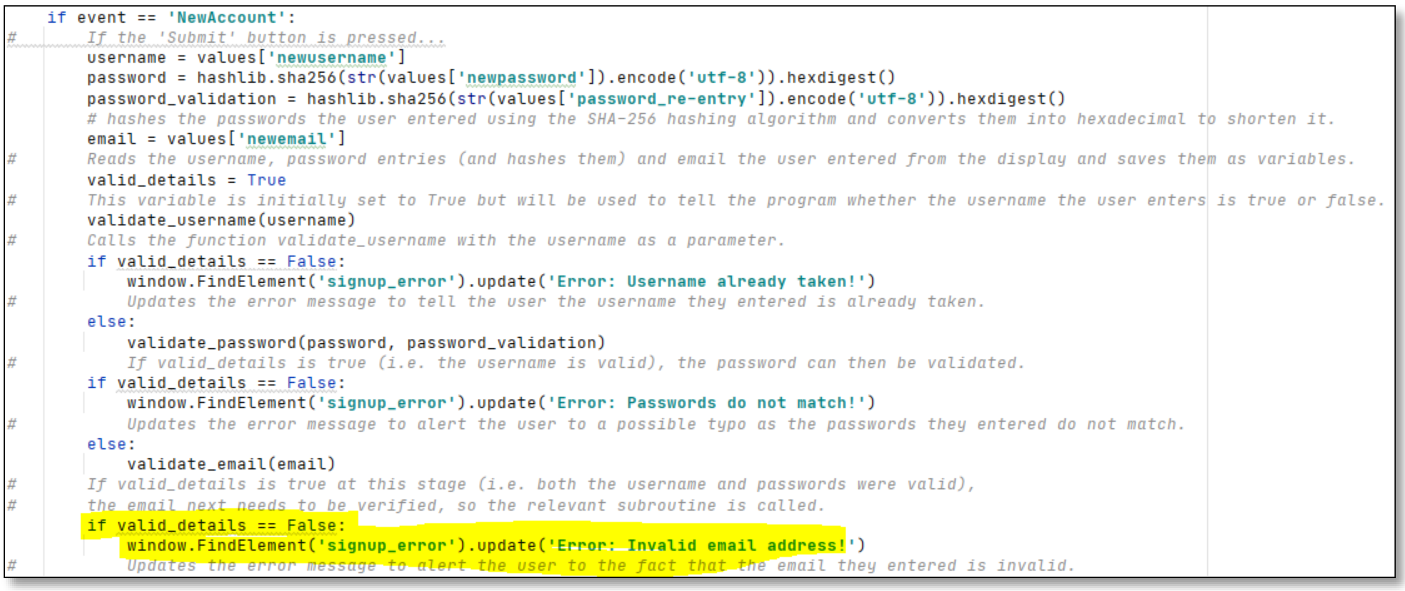
```
    if event == 'NewAccount':
#       If the 'Submit' button is pressed...
        username = values['newusername']
        password = hashlib.sha256(str(values['newpassword']).encode('utf-8')).hexdigest()
        password_validation = hashlib.sha256(str(values['password_re-entry']).encode('utf-8')).hexdigest()
        # hashes the passwords the user entered using the SHA-256 hashing algorithm and converts them into hexadecimal to shorten it.
        email = values['newemail']
#       Reads the username, password entries (and hashes them) and email the user entered from the display and saves them as variables.
        valid_details = True
#       This variable is initially set to True but will be used to tell the program whether the username the user enters is true or false.
        validate_username(username)
#       Calls the function validate_username with the username as a parameter.
        if valid_details == False:
            window.FindElement('signup_error').update('Error: Username already taken!')
#           Updates the error message to tell the user the username they entered is already taken.
        else:
            validate_password(password, password_validation)
#           If valid_details is true (i.e. the username is valid), the password can then be validated.
        if valid_details == False:
            window.FindElement('signup_error').update('Error: Passwords do not match!')
#           Updates the error message to alert the user to a possible typo as the passwords they entered do not match.
        else:
            validate_email(email)
#       If valid_details is true at this stage (i.e. both the username and passwords were valid),
#       the email next needs to be verified, so the relevant subroutine is called.
```

Next, I need to define the email_verification subroutine. The requirements for emails at this stage of development were detailed above.



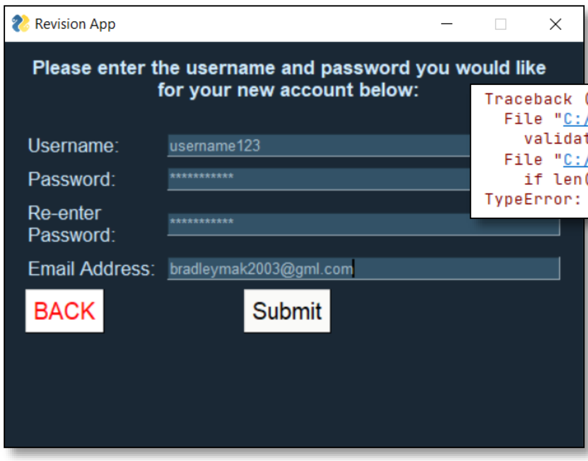
```
def validate_email(email):
    global valid_details
    # makes valid_details global.
    if len(email)>11 and email[(len(email)-11), (len(email)-1)] == '@gmail.com':
        valid_details = True
    else:
        valid_details = False
        # if the email the user entered is a valid gmail address, then valid_details remains true,
        # otherwise it is set to false.
```

Similar to the other validation stages, I also updated the error message to alert the user to the invalid email if that is the case:

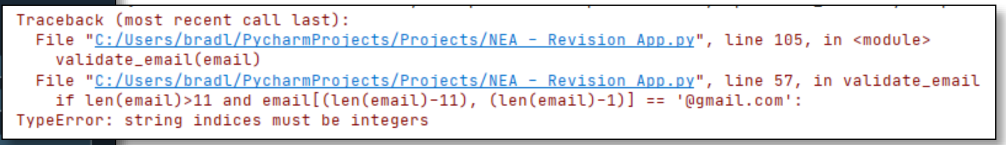


```
    if event == 'NewAccount':
#       If the 'Submit' button is pressed...
        username = values['newusername']
        password = hashlib.sha256(str(values['newpassword']).encode('utf-8')).hexdigest()
        password_validation = hashlib.sha256(str(values['password_re-entry']).encode('utf-8')).hexdigest()
        # hashes the passwords the user entered using the SHA-256 hashing algorithm and converts them into hexadecimal to shorten it.
        email = values['newemail']
#       Reads the username, password entries (and hashes them) and email the user entered from the display and saves them as variables.
        valid_details = True
#       This variable is initially set to True but will be used to tell the program whether the username the user enters is true or false.
        validate_username(username)
#       Calls the function validate_username with the username as a parameter.
        if valid_details == False:
            window.FindElement('signup_error').update('Error: Username already taken!')
#           Updates the error message to tell the user the username they entered is already taken.
        else:
            validate_password(password, password_validation)
#           If valid_details is true (i.e. the username is valid), the password can then be validated.
        if valid_details == False:
            window.FindElement('signup_error').update('Error: Passwords do not match!')
#           Updates the error message to alert the user to a possible typo as the passwords they entered do not match.
        else:
            validate_email(email)
#       If valid_details is true at this stage (i.e. both the username and passwords were valid),
#       the email next needs to be verified, so the relevant subroutine is called.
        if valid_details == False:
            window.FindElement('signup_error').update('Error: Invalid email address!')
#           Updates the error message to alert the user to the fact that the email they entered is invalid.
```

Again, I manually cleared 'userinfo.txt' to ensure that there was no duplicate username being entered (as this test is about the email verification) and entered an invalid email address 'bradleymak2003@gml.com'.

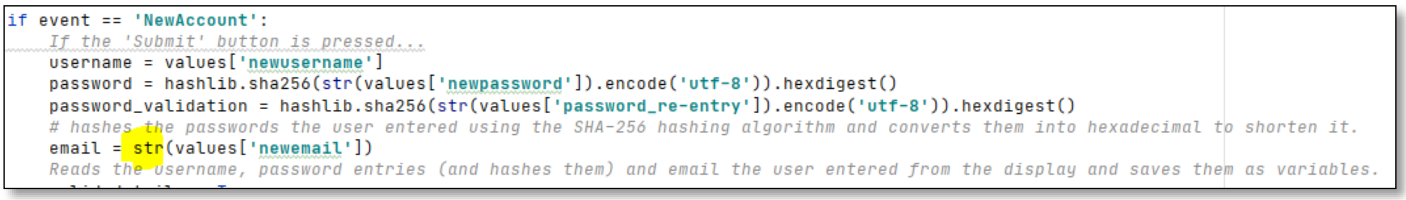


However, then I received the following error:



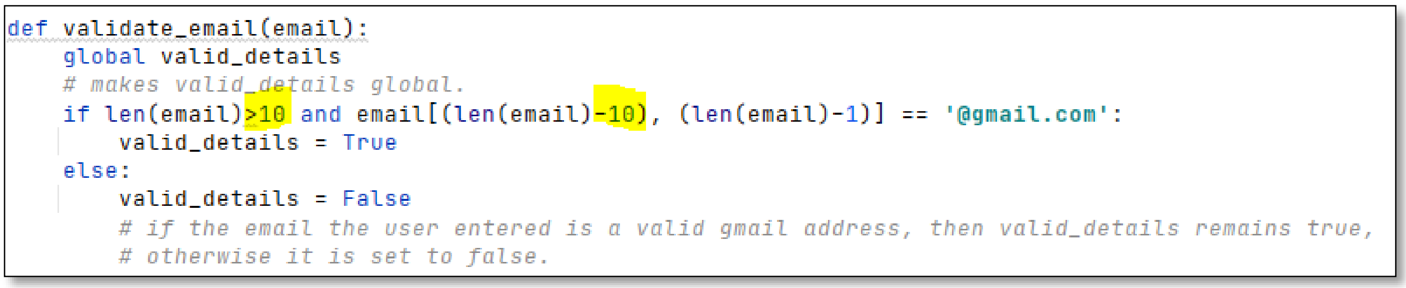
```
Traceback (most recent call last):
  File "C:/Users/bradl/PycharmProjects/Projects/NEA - Revision App.py", line 105, in <module>
    validate_email(email)
  File "C:/Users/bradl/PycharmProjects/Projects/NEA - Revision App.py", line 57, in validate_email
    if len(email)>11 and email[(len(email)-11), (len(email)-1)] == '@gmail.com':
TypeError: string indices must be integers
```

I defined the email as a string:



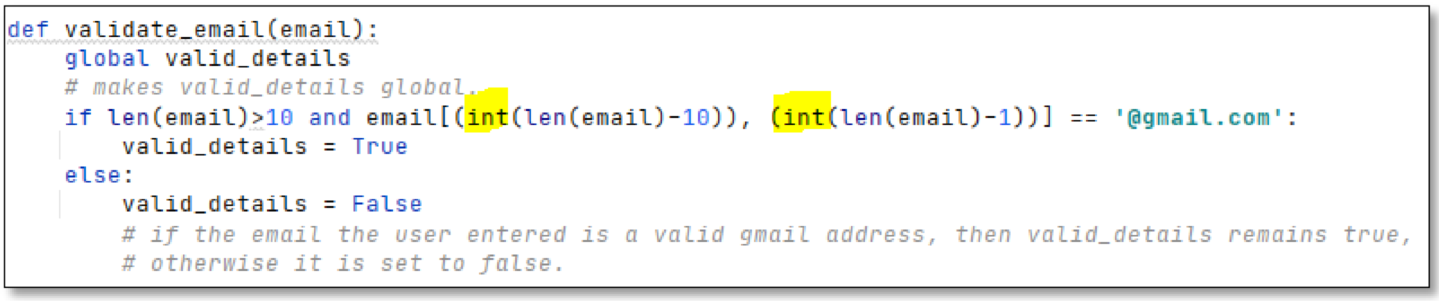
```
if event == 'NewAccount':
    If the 'Submit' button is pressed...
    username = values['newusername']
    password = hashlib.sha256(str(values['newpassword']).encode('utf-8')).hexdigest()
    password_validation = hashlib.sha256(str(values['password_re-entry']).encode('utf-8')).hexdigest()
    # hashes the passwords the user entered using the SHA-256 hashing algorithm and converts them into hexadecimal to shorten it.
    email = str(values['newemail'])
    Reads the username, password entries (and hashes them) and email the user entered from the display and saves them as variables.
```

I also realised that the index values/length I was checking for in my email validation were slightly incorrect, so I also updated them:



```
def validate_email(email):
    global valid_details
    # makes valid_details global.
    if len(email)>10 and email[(len(email)-10), (len(email)-1)] == '@gmail.com':
        valid_details = True
    else:
        valid_details = False
        # if the email the user entered is a valid gmail address, then valid_details remains true,
        # otherwise it is set to false.
```

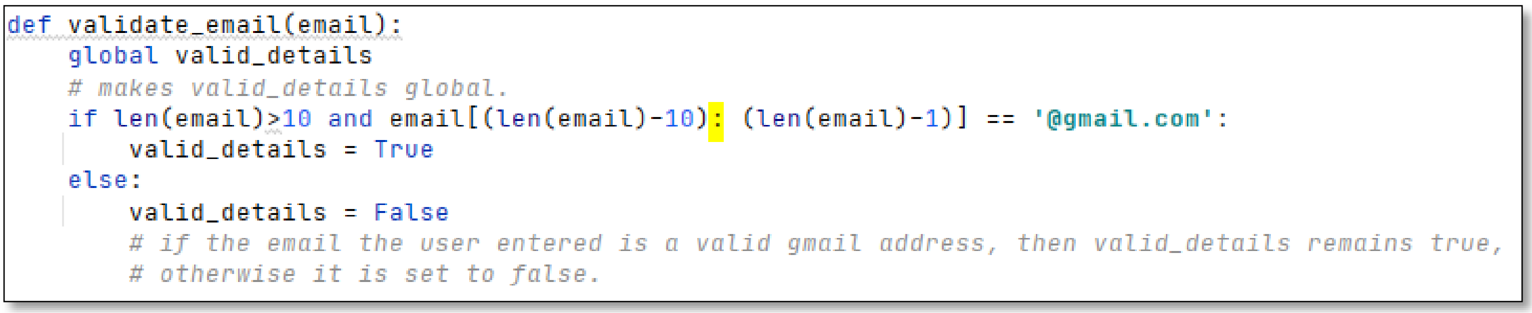
I then tried explicitly setting the string indices to integers to try and solve this problem:



```
def validate_email(email):
    global valid_details
    # makes valid_details global.
    if len(email)>10 and email[(int(len(email)-10)), (int(len(email)-1))] == '@gmail.com':
        valid_details = True
    else:
        valid_details = False
        # if the email the user entered is a valid gmail address, then valid_details remains true,
        # otherwise it is set to false.
```

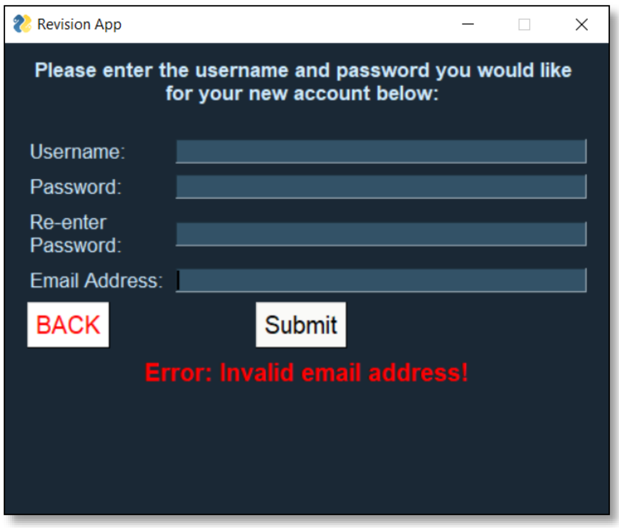
I now carried out the test again with the same inputs as before: 'username123', 'password123', 'password123', 'bradleymak2003@gml.com'.

However, the same error still appeared. After looking at the website https://stackoverflow.com/questions/6077675/why-am-i-seeing-typeerror-string-indices-must-be-integers, I realised that I need to use a colon rather than a comma, so I changed this and ran my program again:



```
def validate_email(email):
    global valid_details
    # makes valid_details global.
    if len(email)>10 and email[(len(email)-10): (len(email)-1)] == '@gmail.com':
        valid_details = True
    else:
        valid_details = False
        # if the email the user entered is a valid gmail address, then valid_details remains true,
        # otherwise it is set to false.
```

I manually cleared 'userinfo.txt', and carried out another test with the same inputs as above…



Revision App

Please enter the username and password you would like for your new account below:

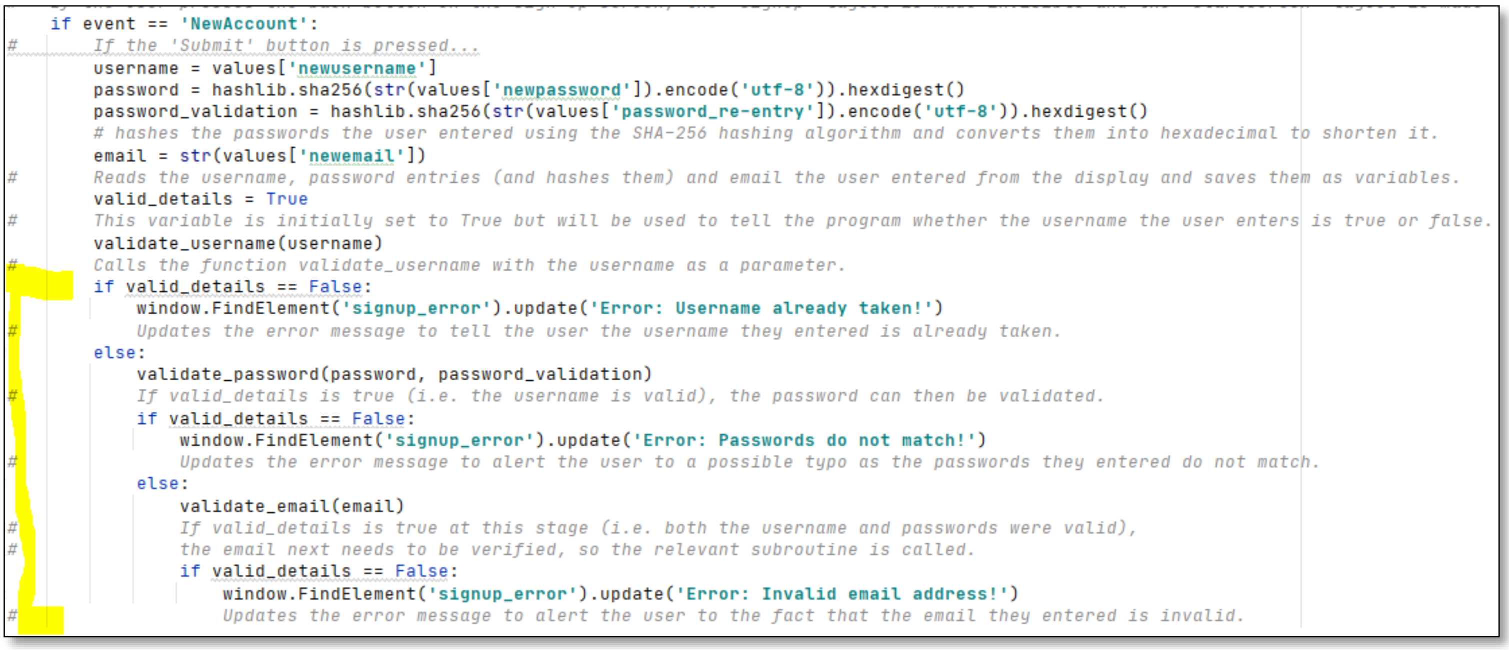
Username:
Password:
Re-enter Password:
Email Address:

BACK   Submit

Error: Invalid email address!

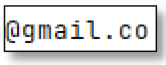
And it came up with the correct error message (again testing proof for this will be shown in the testing evidence powerpoint later).

However, now when I input a duplicate username, I receive an invalid email error. This was because I am using the same variable for all validations (valid_details), so originally because the validations were not nested, it would automatically update the error message to the email error message (as this is validated last, and so is the last validation code to be ran). I have now nested the inputs which solves this issue:

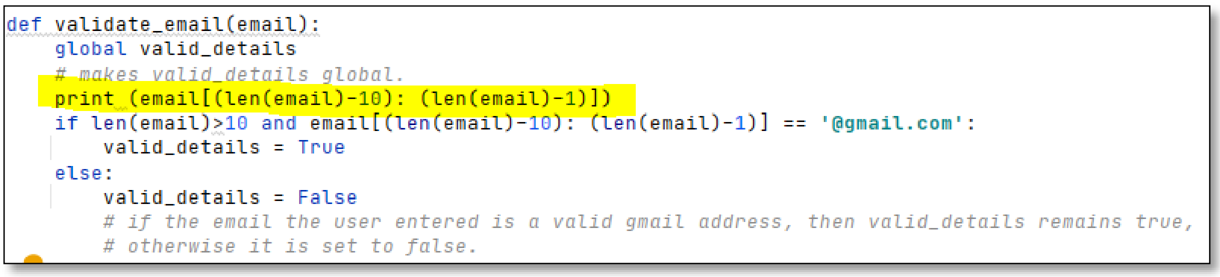


```
    if event == 'NewAccount':
#       If the 'Submit' button is pressed...
        username = values['newusername']
        password = hashlib.sha256(str(values['newpassword']).encode('utf-8')).hexdigest()
        password_validation = hashlib.sha256(str(values['password_re-entry']).encode('utf-8')).hexdigest()
        # hashes the passwords the user entered using the SHA-256 hashing algorithm and converts them into hexadecimal to shorten it.
        email = str(values['newemail'])
#       Reads the username, password entries (and hashes them) and email the user entered from the display and saves them as variables.
        valid_details = True
#       This variable is initially set to True but will be used to tell the program whether the username the user enters is true or false.
        validate_username(username)
#       Calls the function validate_username with the username as a parameter.
        if valid_details == False:
            window.FindElement('signup_error').update('Error: Username already taken!')
#           Updates the error message to tell the user the username they entered is already taken.
        else:
            validate_password(password, password_validation)
#           If valid_details is true (i.e. the username is valid), the password can then be validated.
            if valid_details == False:
                window.FindElement('signup_error').update('Error: Passwords do not match!')
#               Updates the error message to alert the user to a possible typo as the passwords they entered do not match.
            else:
                validate_email(email)
#               If valid_details is true at this stage (i.e. both the username and passwords were valid),
#               the email next needs to be verified, so the relevant subroutine is called.
                if valid_details == False:
                    window.FindElement('signup_error').update('Error: Invalid email address!')
#                   Updates the error message to alert the user to the fact that the email they entered is invalid.
```

There was no error when I entered 'bradleymak2003@gmail.com' (a valid email address) into the email entry field, but it displayed the error message 'Invalid email'. This must be because the substring I am taking out is incorrect, so I put in a print statement so that the next time I ran the program I could see what the substring being taken out of the email is:
When I ran the program with the input 'bradleymak2003@gmail.com' again, the following was printed:

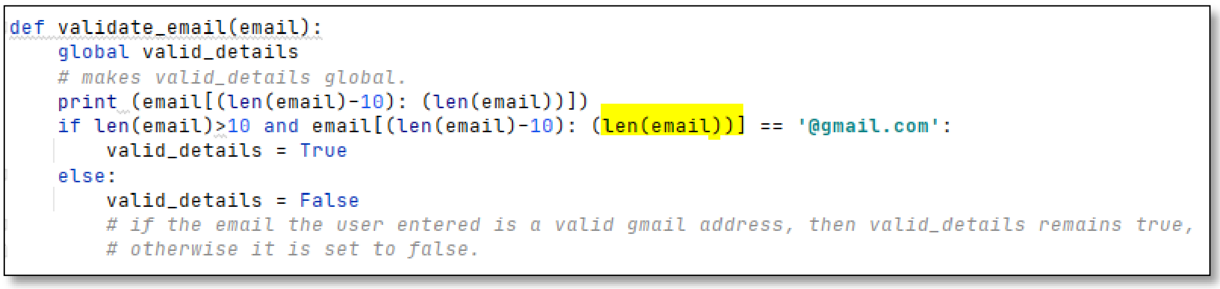


```
@gmail.co
```



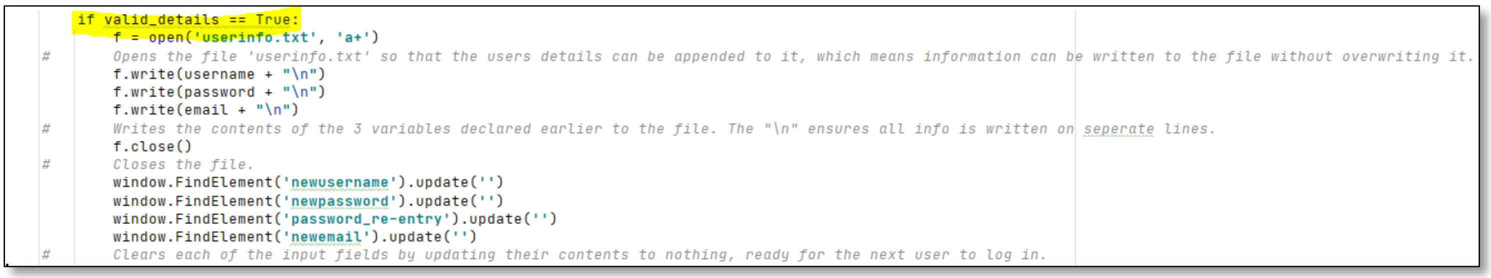
```
def validate_email(email):
    global valid_details
    # makes valid_details global.
    print (email[(len(email)-10): (len(email)-1)])
    if len(email)>10 and email[(len(email)-10): (len(email)-1)] == '@gmail.com':
        valid_details = True
    else:
        valid_details = False
        # if the email the user entered is a valid gmail address, then valid_details remains true,
        # otherwise it is set to false.
```

This shows that the substring is being cut off to early so I increased the upper string indices by 1 (i.e. removed the -1):

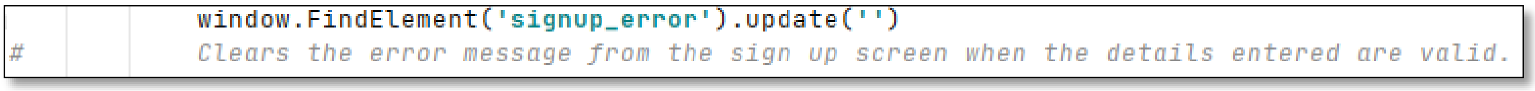

```
def validate_email(email):
    global valid_details
    # makes valid_details global.
    print (email[(len(email)-10): (len(email))])
    if len(email)>10 and email[(len(email)-10): (len(email))] == '@gmail.com':
        valid_details = True
    else:
        valid_details = False
        # if the email the user entered is a valid gmail address, then valid_details remains true,
        # otherwise it is set to false.
```

When I then ran the program again with the same inputs, they were accepted and successfully written to 'userinfo.txt'.

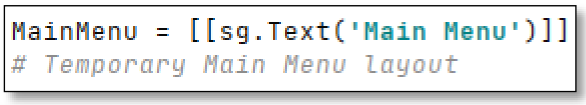
However, information is still being written to the file if it is incorrect, so to fix this (and stop clearing the entry fields if there is an error), I have placed all of the writing of information to the file and clearing of entry fields into an if statement, which required valid_details to be true:

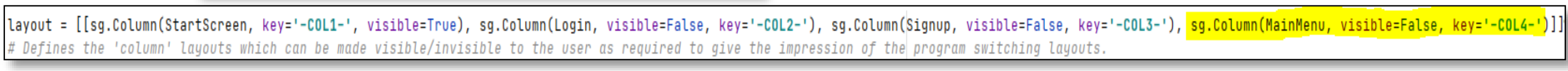

```
    if valid_details == True:
        f = open('userinfo.txt', 'a+')
#       Opens the file 'userinfo.txt' so that the users details can be appended to it, which means information can be written to the file without overwriting it.
        f.write(username + "\n")
        f.write(password + "\n")
        f.write(email + "\n")
#       Writes the contents of the 3 variables declared earlier to the file. The "\n" ensures all info is written on seperate lines.
        f.close()
#       Closes the file.
        window.FindElement('newusername').update('')
        window.FindElement('newpassword').update('')
        window.FindElement('password_re-entry').update('')
        window.FindElement('newemail').update('')
#       Clears each of the input fields by updating their contents to nothing, ready for the next user to log in.
```

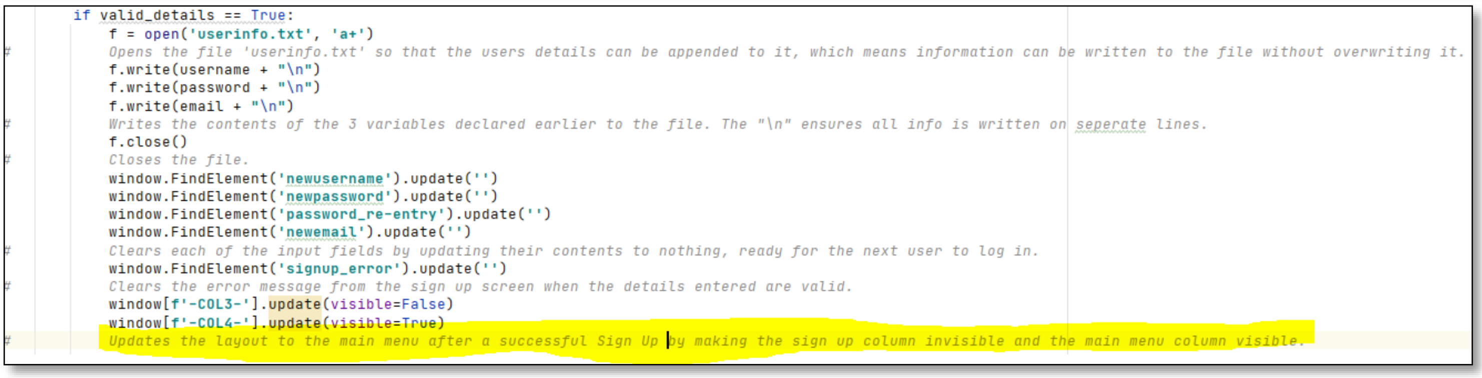
I also added the following piece of code, which removes the error message on the sign up screen when the details entered are valid. This ensures the error message will not remain when the current user logs out and another user goes to create an account.


```
        window.FindElement('signup_error').update('')
#       Clears the error message from the sign up screen when the details entered are valid.
```

I have next created a temporary main menu layout, added it to the column layout so it can be made visible/invisible as necessary, and told the program to update the layout to the main menu once there is a successful sign up, to allow me to test whether my app is working correctly.


```
MainMenu = [[sg.Text('Main Menu')]]
# Temporary Main Menu layout
```


```
layout = [[sg.Column(StartScreen, key='-COL1-', visible=True), sg.Column(Login, visible=False, key='-COL2-'), sg.Column(Signup, visible=False, key='-COL3-'), sg.Column(MainMenu, visible=False, key='-COL4-')]]
# Defines the 'column' layouts which can be made visible/invisible to the user as required to give the impression of the program switching layouts.
```


```
    if valid_details == True:
        f = open('userinfo.txt', 'a+')
#       Opens the file 'userinfo.txt' so that the users details can be appended to it, which means information can be written to the file without overwriting it.
        f.write(username + "\n")
        f.write(password + "\n")
        f.write(email + "\n")
#       Writes the contents of the 3 variables declared earlier to the file. The "\n" ensures all info is written on seperate lines.
        f.close()
#       Closes the file.
        window.FindElement('newusername').update('')
        window.FindElement('newpassword').update('')
        window.FindElement('password_re-entry').update('')
        window.FindElement('newemail').update('')
#       Clears each of the input fields by updating their contents to nothing, ready for the next user to log in.
        window.FindElement('signup_error').update('')
#       Clears the error message from the sign up screen when the details entered are valid.
        window[f'-COL3-'].update(visible=False)
        window[f'-COL4-'].update(visible=True)
#       Updates the layout to the main menu after a successful Sign Up by making the sign up column invisible and the main menu column visible.
```

Now I can begin testing. Testing evidence for all of the following tests can be found in the testing evidence powerpoint. (The file userinfo.txt was cleared between tests to ensure that a duplicate username did not get in the way of other tests)

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
2.3.1	Entering username, password and email into relevant fields	String 'username123' String 'password123' String 'password123' String 'bradleymak2003 @gmail.com' **VALID DATA**	App will allow user to enter these strings into the relevant input boxes. The password should be dotted.	Checking whether the user can enter a string into the textual input boxes that should appear on screen.	**Allows user to successfully enter details into the input boxes.**
2.3.2	Selection of button to confirm account details	Mouse click	Entry fields will clear and strings will be written to 'userinfo.txt', and layout will be updated to MAIN MENU.	To check whether the confirm account button takes the user to the main menu once the details they have entered have been validated and stored.	**Valid details are written to the file, the layout is updated to the main menu and the entry fields clear (as shown earlier).**

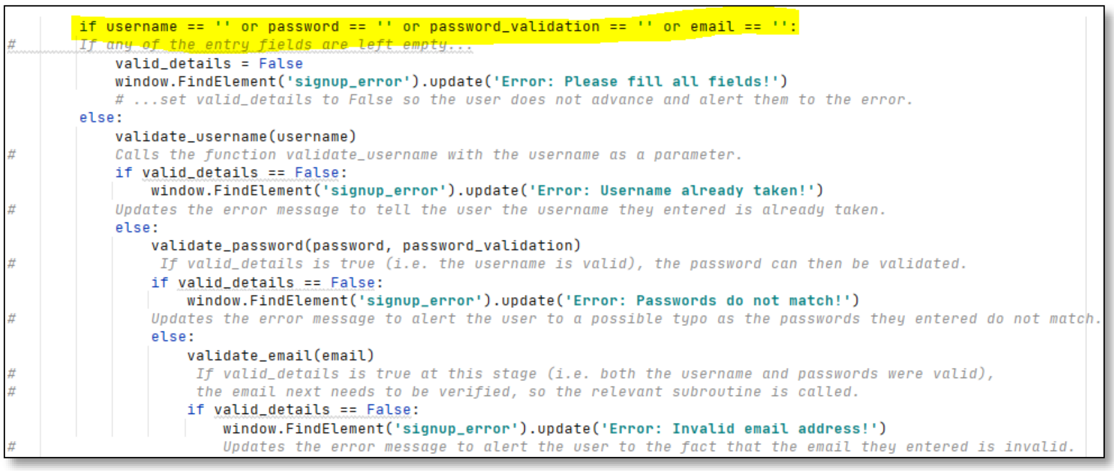
Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
2.4.1	Entering duplicate username and invalid email	String 'username123' String 'password123' String 'password123' String 'bradleymak2003@gml.com' **INVALID DATA**	App will allow user to enter these strings into the relevant input boxes. The password should be dotted.	Checking whether the user can enter a string into the textual input boxes that should appear on screen.	**Allows user to successfully enter details into the input boxes.**
2.4.2	Selection of button to confirm account details	Mouse click	Error message will appear at bottom of screen stating invalid username at first (as this username was used in a previous test), so change username to '111' and try again, now error message should say invalid email and layout will remain the same.	To check whether the validation algorithm detects an invalid email and displays the issue to the user (and does not proceed to the main menu).	**Both error messages appear correctly and the layout remains the same until all of the inputs are valid.**

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
2.5.1	Entering passwords that do not match	String '123' String 'password123' String 'password'	App will allow user to enter these strings into the relevant input	Checking whether the user can enter a string into the	**Allows user to successfully enter details into the input boxes.**

		String 'bradleymak2003 @gmail.com' **INVALID DATA**	boxes. The password should be dotted.	textual input boxes that should appear on screen.	
2.5.2	Selection of button to confirm account details	Mouse click	Error message will appear at bottom of screen stating that the user mistyped their password (i.e. the 2 password input boxes do not match).	To check whether the validation algorithm detects an invalid password re-entry and displays the issue to the user (and does not proceed to the main menu).	**The error message appears correctly and the layout remains the same. The layout changes when the error is fixed and valid inputs are submitted.**

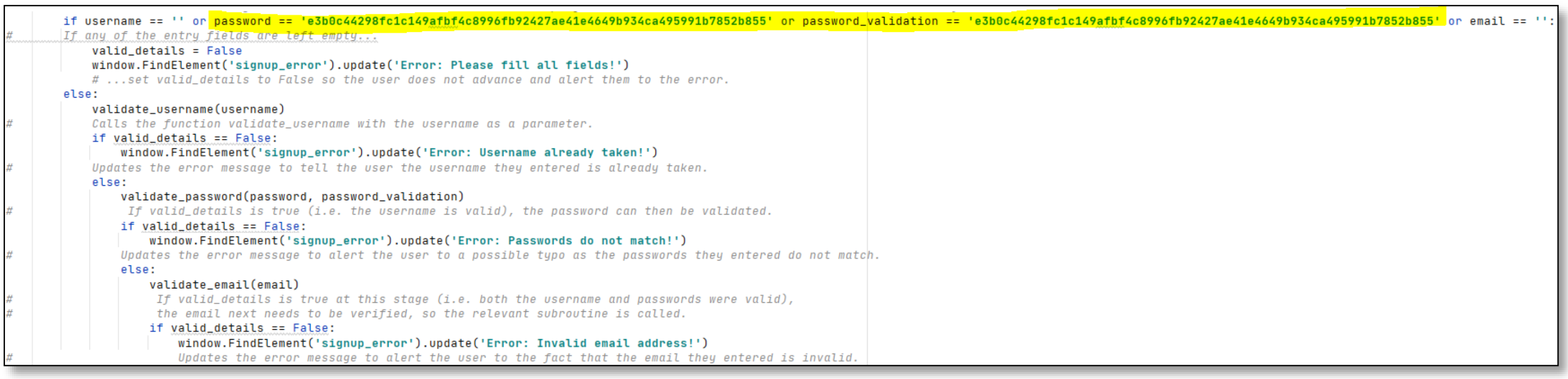
Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
2.6.1	Missing field	String '123' String 'password123' String 'password123' String ' ' **INVALID DATA**	App will allow user to enter these strings into the relevant input boxes. The password should be dotted.	Checking whether the user can enter a string into the textual input boxes that should appear on screen.	**Allows user to successfully enter details into the input boxes.**
2.6.2	Selection of button to confirm account details	Mouse click	Error message will appear at bottom of screen stating that all fields need to be entered (as email is missing)	To check whether the validation algorithm detects missing inputs, as all inputs are required.	**Error message does appear stating invalid email. But if I leave the password fields empty instead, the program advances when it shouldn't do as the field hasn't been filled.**

The issue with test 2.6.2 was that it allowed me to advance and wrote the details I did enter to the file even though I didn't fill the password fields (which shouldn't happen as you obviously need a password). To fix this, I wrote the code below:



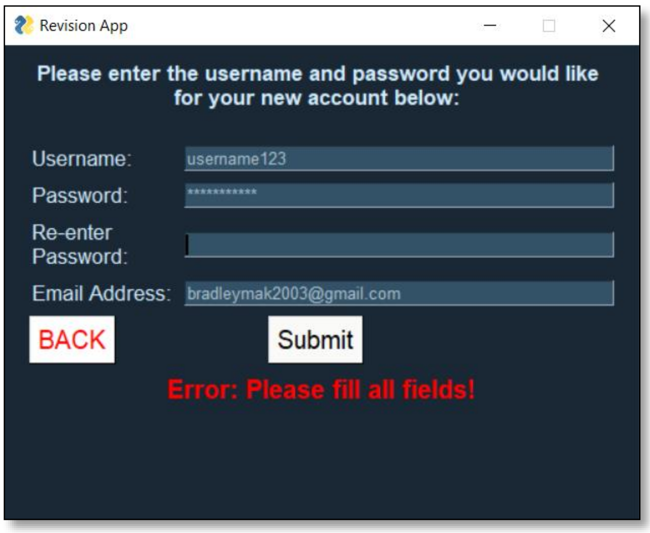
```
        if username == '' or password == '' or password_validation == '' or email == '':
#       If any of the entry fields are left empty...
            valid_details = False
            window.FindElement('signup_error').update('Error: Please fill all fields!')
            # ...set valid_details to False so the user does not advance and alert them to the error.
        else:
            validate_username(username)
#           Calls the function validate_username with the username as a parameter.
            if valid_details == False:
                window.FindElement('signup_error').update('Error: Username already taken!')
#           Updates the error message to tell the user the username they entered is already taken.
            else:
                validate_password(password, password_validation)
#                If valid_details is true (i.e. the username is valid), the password can then be validated.
                if valid_details == False:
                    window.FindElement('signup_error').update('Error: Passwords do not match!')
#               Updates the error message to alert the user to a possible typo as the passwords they entered do not match.
                else:
                    validate_email(email)
#                    If valid_details is true at this stage (i.e. both the username and passwords were valid),
#                    the email next needs to be verified, so the relevant subroutine is called.
                    if valid_details == False:
                        window.FindElement('signup_error').update('Error: Invalid email address!')
#                       Updates the error message to alert the user to the fact that the email they entered is invalid.
```

When I ran my program with the inputs ‘username12345’, ‘password123’, ‘’ and ‘bradleymak2003@gmail.com’, the error message came up with ‘passwords do not match’ rather than ‘Fill all fields’. I looked at the variables I was using and realised that I am checking if the hash value of the password is equal to nothing, which cannot be the case as the SHA-256 hashing algorithm produces a has value even for an empty string. So, to fix this, I went on the website https://xorbin.com/tools/sha256-hash-calculator and got the hash value for an empty string (e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855), and used this in my code:



```
    if username == '' or password == 'e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855' or password_validation == 'e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855' or email == '':
#   If any of the entry fields are left empty...
        valid_details = False
        window.FindElement('signup_error').update('Error: Please fill all fields!')
        # ...set valid_details to False so the user does not advance and alert them to the error.
    else:
        validate_username(username)
#       Calls the function validate_username with the username as a parameter.
        if valid_details == False:
            window.FindElement('signup_error').update('Error: Username already taken!')
#       Updates the error message to tell the user the username they entered is already taken.
        else:
            validate_password(password, password_validation)
#            If valid_details is true (i.e. the username is valid), the password can then be validated.
            if valid_details == False:
                window.FindElement('signup_error').update('Error: Passwords do not match!')
#           Updates the error message to alert the user to a possible typo as the passwords they entered do not match.
            else:
                validate_email(email)
#                If valid_details is true at this stage (i.e. both the username and passwords were valid),
#                the email next needs to be verified, so the relevant subroutine is called.
                if valid_details == False:
                    window.FindElement('signup_error').update('Error: Invalid email address!')
#                   Updates the error message to alert the user to the fact that the email they entered is invalid.
```

When I ran the program again with the same inputs as previously, the app successfully notified me that an input field was missing:



Revision App

Please enter the username and password you would like for your new account below:

Username: username123

Password: ***********

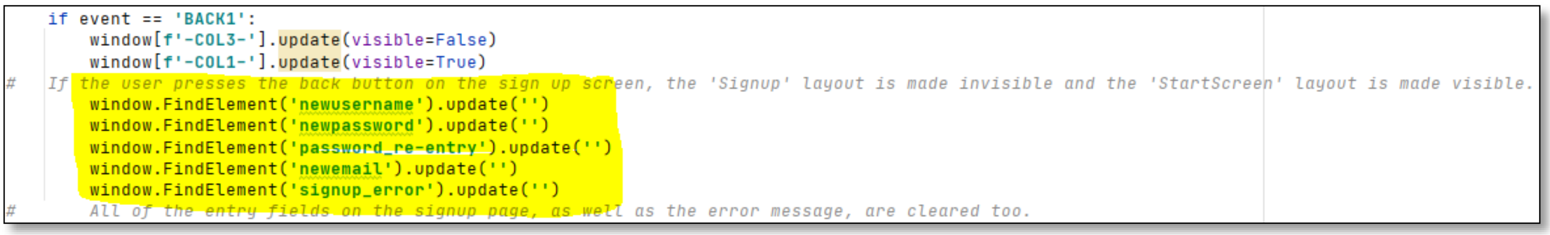
Re-enter Password:

Email Address: bradleymak2003@gmail.com

BACK   Submit

Error: Please fill all fields!

I also noticed that when the back button is pressed and the user goes back onto the signup screen, the information entered remains there. This should not happen as this is a security issue, as if another user goes to sign up, the previous users entry details will be there if they forgot to manually clear them before pressing the back button. To fix this, I cleared all of the entry fields when the back button is pressed:



```
    if event == 'BACK1':
        window[f'-COL3-'].update(visible=False)
        window[f'-COL1-'].update(visible=True)
#   If the user presses the back button on the sign up screen, the 'Signup' layout is made invisible and the 'StartScreen' layout is made visible.
        window.FindElement('newusername').update('')
        window.FindElement('newpassword').update('')
        window.FindElement('password_re-entry').update('')
        window.FindElement('newemail').update('')
        window.FindElement('signup_error').update('')
#       All of the entry fields on the signup page, as well as the error message, are cleared too.
```

Sign up – Review 2

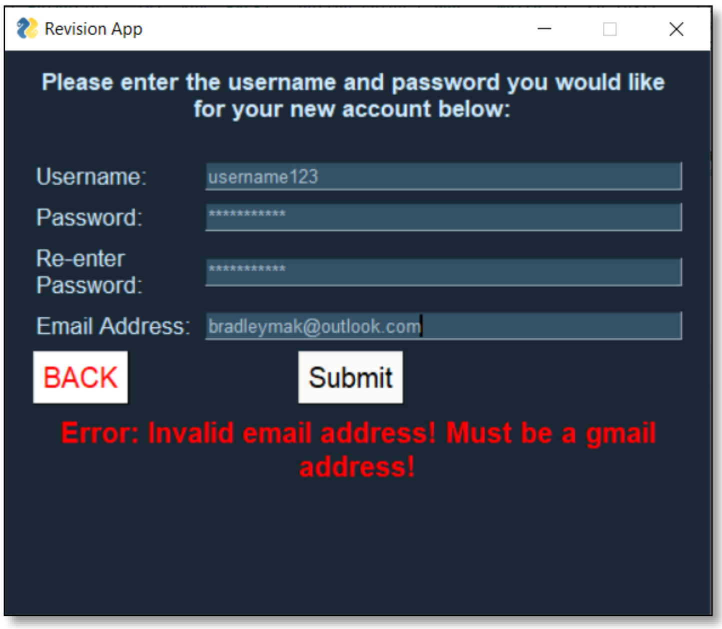
Success Criteria

At this point, the sign-up feature is not fully completed, so I have not yet fully met success criteria 1 yet. I have so far met:

- SC10 - The stakeholder would like the system to be user-friendly and easy to understand.
- SC1 - The stakeholder would like the system to allow them to create an account if they haven't already got one.

Stakeholder feedback

Now that I have completed the sign-up feature and extensively tested it, I sent this new prototype of my solution (along with the external files) to one of my stakeholders, Vivek, to get his opinion on the sign-up process and to test how robust it is.
He said that the layout looked clean and simple and the program was easy to use. The only issue he encountered was that he used an outlook email (and my program only allows gmail addresses to be entered at this point), so he was confused as to why his email wasn't working as the error message just says 'invalid email', so I have changed the error message top notify the user that it has to be a gmail address to avoid this confusion in the future:


Revision App

Please enter the username and password you would like for your new account below:

Username: username123
Password: ***********
Re-enter Password: ***********
Email Address: bradleymak@outlook.com

BACK Submit

Error: Invalid email address! Must be a gmail address!

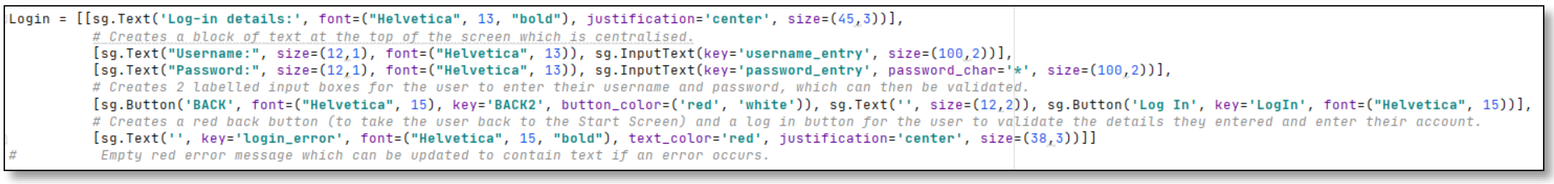
Summary of progress made and how this prototype compares to the previous

I now have an effective sign up system that is robust and does not crash no matter what the inputs are, as tested by my stakeholders. It can deal with duplicate usernames, passwords that do not match, invalid emails and missing fields all together, and has a clear interface and method of notifying the user of the issue so they can fix their inputs. The stakeholder feedback has made my program more suited to its target audience and has improved my solution to make it more appealing to students.

Log in – Stage 1

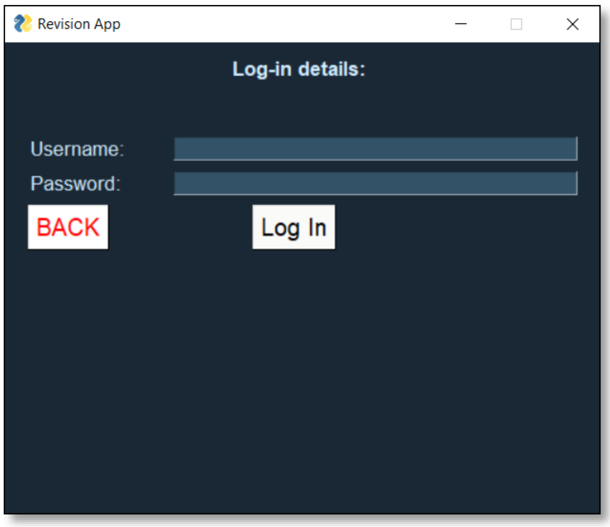
Next, I am going to develop the log in feature, to allow the user to log in to an account they have previously created.

To start, I am going to create the layout for the log in screen. In order to continue meeting success criteria 10, this layout needs to be user friendly, clear and easy to use. In order to achieve this, I have coded the following:



```
Login = [[sg.Text('Log-in details:', font=("Helvetica", 13, "bold"), justification='center', size=(45,3))],
          # Creates a block of text at the top of the screen which is centralised.
          [sg.Text("Username:", size=(12,1), font=("Helvetica", 13)), sg.InputText(key='username_entry', size=(100,2))],
          [sg.Text("Password:", size=(12,1), font=("Helvetica", 13)), sg.InputText(key='password_entry', password_char='*', size=(100,2))],
          # Creates 2 labelled input boxes for the user to enter their username and password, which can then be validated.
          [sg.Button('BACK', font=("Helvetica", 15), key='BACK2', button_color=('red', 'white')), sg.Text('', size=(12,2)), sg.Button('Log In', key='LogIn', font=("Helvetica", 15))],
          # Creates a red back button (to take the user back to the Start Screen) and a log in button for the user to validate the details they entered and enter their account.
          [sg.Text('', key='login_error', font=("Helvetica", 15, "bold"), text_color='red', justification='center', size=(38,3))]]
#          Empty red error message which can be updated to contain text if an error occurs.
```

The result is a layout which looks like this:



Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
3.1.1	Entering username and password into relevant fields	String 'username123' String 'password123' **VALID DATA**	App will allow user to enter these strings into the relevant input boxes	Checking whether the user can enter a string into the textual input boxes that should appear on screen.	**Program allowed the user to enter their details successfully.**
3.1.2	Selection of button to confirm account details	Mouse click	Nothing, as functionality of button has not been implemented yet.	To check whether the confirm button on screen appears in the correct place and allows user interaction.	**Nothing happened upon pressing the button.**

Testing evidence for these tests can be found in the testing evidence powerpoint.

Log in – Review 1

Success Criteria

I have not met any new success criteria since the last review. Currently I have met:

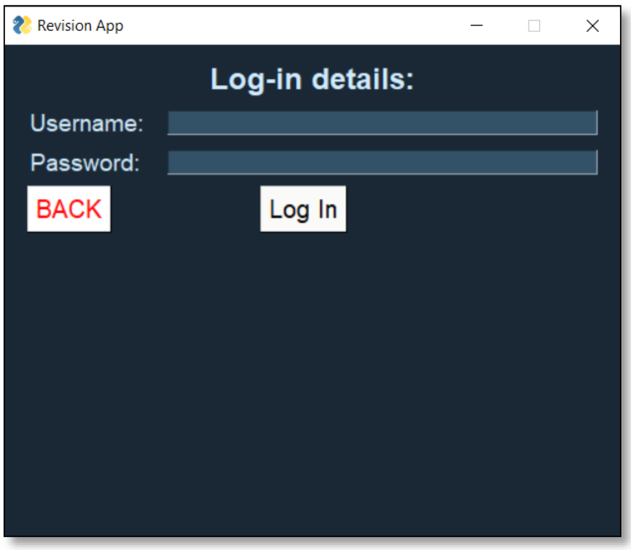
- SC10 - The stakeholder would like the system to be user-friendly and easy to understand.
- SC1 - The stakeholder would like the system to allow them to create an account if they haven't already got one.

Stakeholder feedback

I asked one of my stakeholders, Marcus, for feedback on the layout of the log in screen. He said that it “looks ok but it’d be clearer if the text was a bit bigger and filled a bit more of the screen, because for example there is a large gap between the title at the top and the input boxes”.

Following his feedback, I have increased the size of the text to make it clearer and take up a bit more of the screen. However, I do not want to increase the text size too much as it will begin to look unauthentic.

This is the new layout after I have made these minor changes:

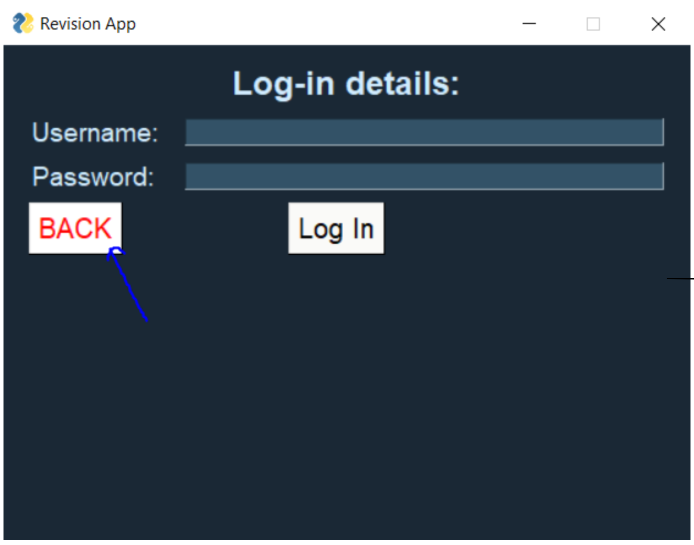


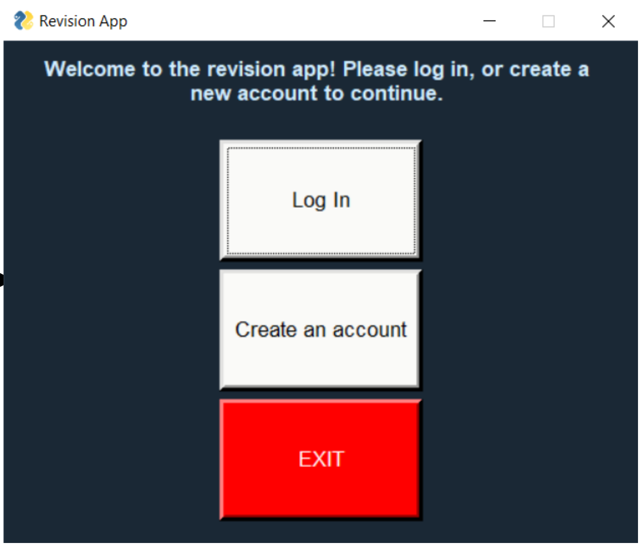
Log in – Stage 2

Now that the interface is created and has been approved by stakeholders, I can begin work on implementing the functionality of the buttons.

I will start with the back button. This is a very similar process to the back button on the sign up screen – if it is pressed it updates the layout to the start screen by making the log in layout invisible and making the start screen layout visible. Again, similar to the sign up back button, I have cleared the input fields on the login screen upon pressing the back button. This is again for security reasons.

```
    if event == 'BACK2':
        window[f'-COL2-'].update(visible=False)
        window[f'-COL1-'].update(visible=True)
#       If the user presses the back button on the log in screen, the 'Login' layout is made invisible and the 'StartScreen' layout is made visible.
        window.FindElement('username_entry').update('')
        window.FindElement('password_entry').update('')
        window.FindElement('login_error').update('')
#       The input fields, as well as the error message, on the log in screen are cleared for security reasons upon pressing the back button.
```




Revision App
Welcome to the revision app! Please log in, or create a new account to continue.
Log In
Create an account
EXIT

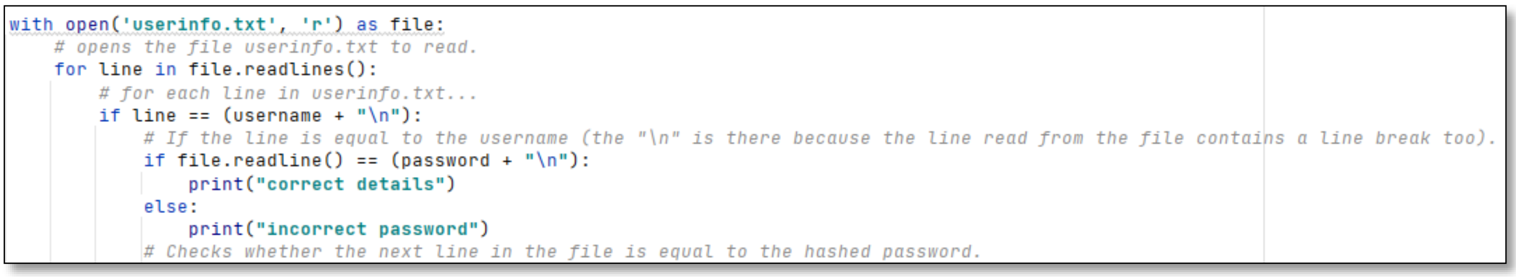
Now I will work on the functionality of the button to validate your details and log into your account.

It will work by sequentially searching through the file 'userinfo.txt' for the username the user enters, and then checking the next line in the file as this should be the password. This is because the way the information in the file has been stored is in blocks of three. For example:
Line 1: User 1 username
Line 2: User 1 hashed password
Line 3: User 1 email
Line 4: User 2 username
Line 5: User 2 hashed password
Line 6: User 2 email
Line 7: User 3 username
Line 8: User 3 hashed password
Line 9: User 3 email
So the program will search the first line in the file, if it is the password, the next line will be checked to see if it matches the password the user entered, if not it will add 3 to the counter and thus check the next username. This process will repeat until either the username is found and the password entered is correct (if it is not this will be notified to the user) or it does not exist and this will be notified to the user.

I have started by saving the username and password entered by the user as variables, with the password being hashed using the same hashing algorithm (SHA-256) as it was hashed with when the account was created (so it can be compared fairly).

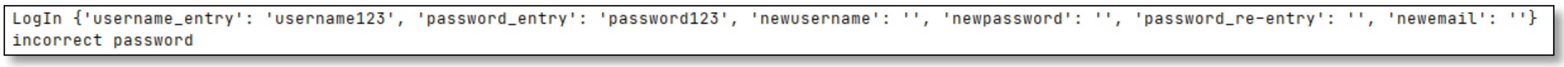
```
if event == 'LogIn':
    # If the Log in button on the LOGIN layout is pressed...
    username = values['username_entry']
    password = hashlib.sha256(str(values['password_entry']).encode('utf-8')).hexdigest()
    # The username and password the user entered are saved as variables (and the password is hashed before being saved).
```

Next, I used the same logic as in the password validation subroutine, by sequentially going through the lines in the file and if they are equal to the username, the next line in the file has to equal to the password for the user to successfully log in.



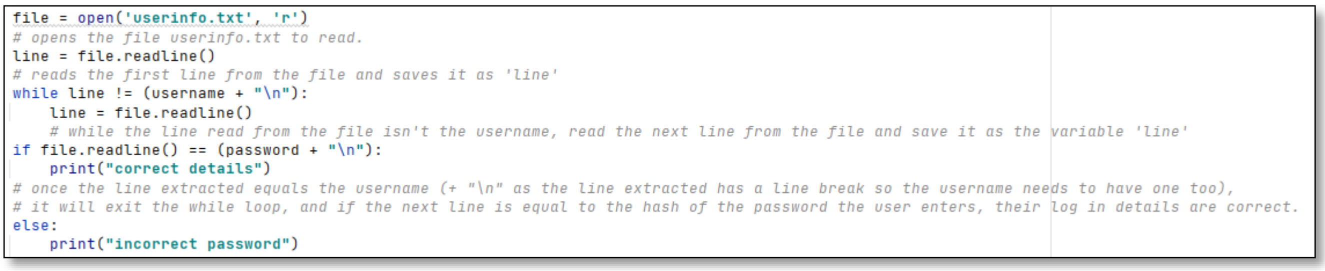
```python
with open('userinfo.txt', 'r') as file:
    # opens the file userinfo.txt to read.
    for line in file.readlines():
        # for each line in userinfo.txt...
        if line == (username + "\n"):
            # If the line is equal to the username (the "\n" is there because the line read from the file contains a line break too).
            if file.readline() == (password + "\n"):
                print("correct details")
            else:
                print("incorrect password")
            # Checks whether the next line in the file is equal to the hashed password.
```

However, when I create an account with the username ‘username123’ and password ‘password123’, and then enter these details to the log in screen, incorrect password is being printed to the console. This means that the program is detecting that the username is there but not detecting that the line after it is the same as the hash value of the password I entered



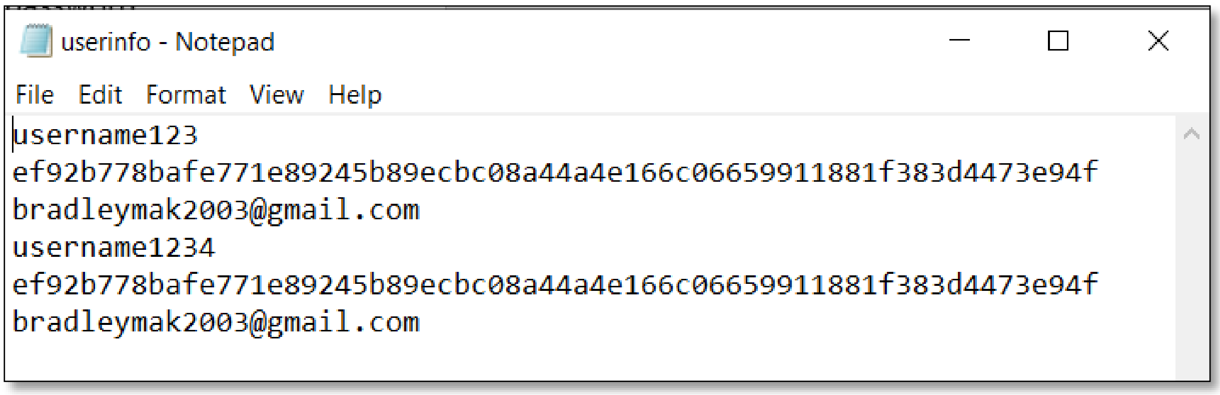
```
LogIn {'username_entry': 'username123', 'password_entry': 'password123', 'newusername': '', 'newpassword': '', 'password_re-entry': '', 'newemail': ''}
incorrect password
```

I rewrote my code using a different approach of saving each line as a variable first and use a while loop:



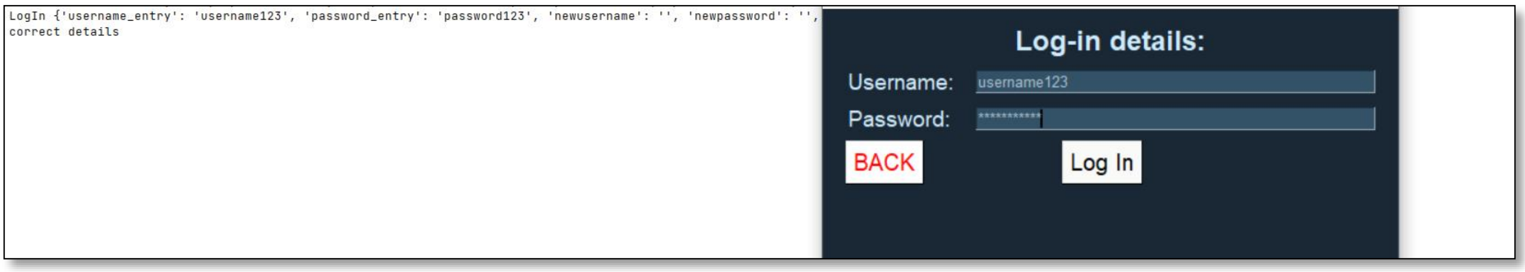
```python
file = open('userinfo.txt', 'r')
# opens the file userinfo.txt to read.
line = file.readline()
# reads the first line from the file and saves it as 'line'
while line != (username + "\n"):
    line = file.readline()
    # while the line read from the file isn't the username, read the next line from the file and save it as the variable 'line'
if file.readline() == (password + "\n"):
    print("correct details")
# once the line extracted equals the username (+ "\n" as the line extracted has a line break so the username needs to have one too),
# it will exit the while loop, and if the next line is equal to the hash of the password the user enters, their log in details are correct.
else:
    print("incorrect password")
```

I then tested my new approach to see if, when I entered the correct username (‘username123’) and password (‘password123’), ‘correct details’ was printed.
This was the contents of ‘userinfo.txt’:



```
username123
ef92b778bafe771e89245b89ecbc08a44a4e166c06659911881f383d4473e94f
bradleymak2003@gmail.com
username1234
ef92b778bafe771e89245b89ecbc08a44a4e166c06659911881f383d4473e94f
bradleymak2003@gmail.com
```

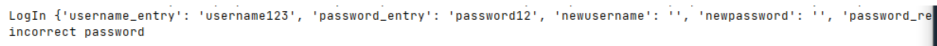
This was what was printed when I entered the details:



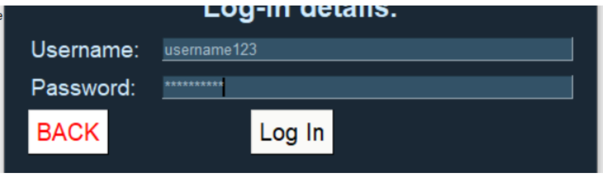
```
LogIn {'username_entry': 'username123', 'password_entry': 'password123', 'newusername': '', 'newpassword': '',
correct details
```

You can see that ‘correct details’ is printed when a username and corresponding password that are in the file are entered.

I then tried this again using an incorrect password (‘password12’):

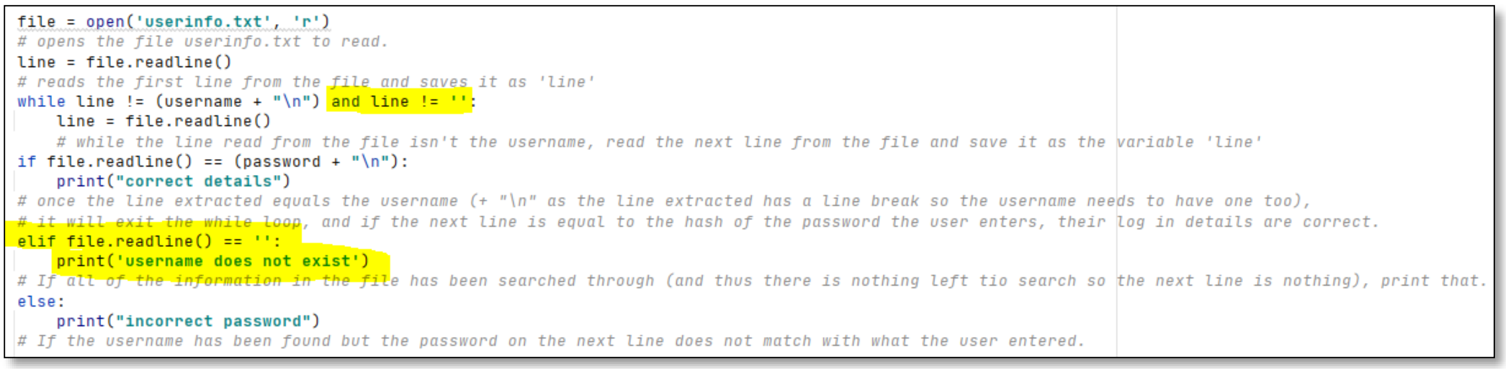


```
LogIn {'username_entry': 'username123', 'password_entry': 'password12', 'newusername': '', 'newpassword': '', 'password_re
incorrect password
```



As you can see, ‘incorrect password’ is correctly stated.

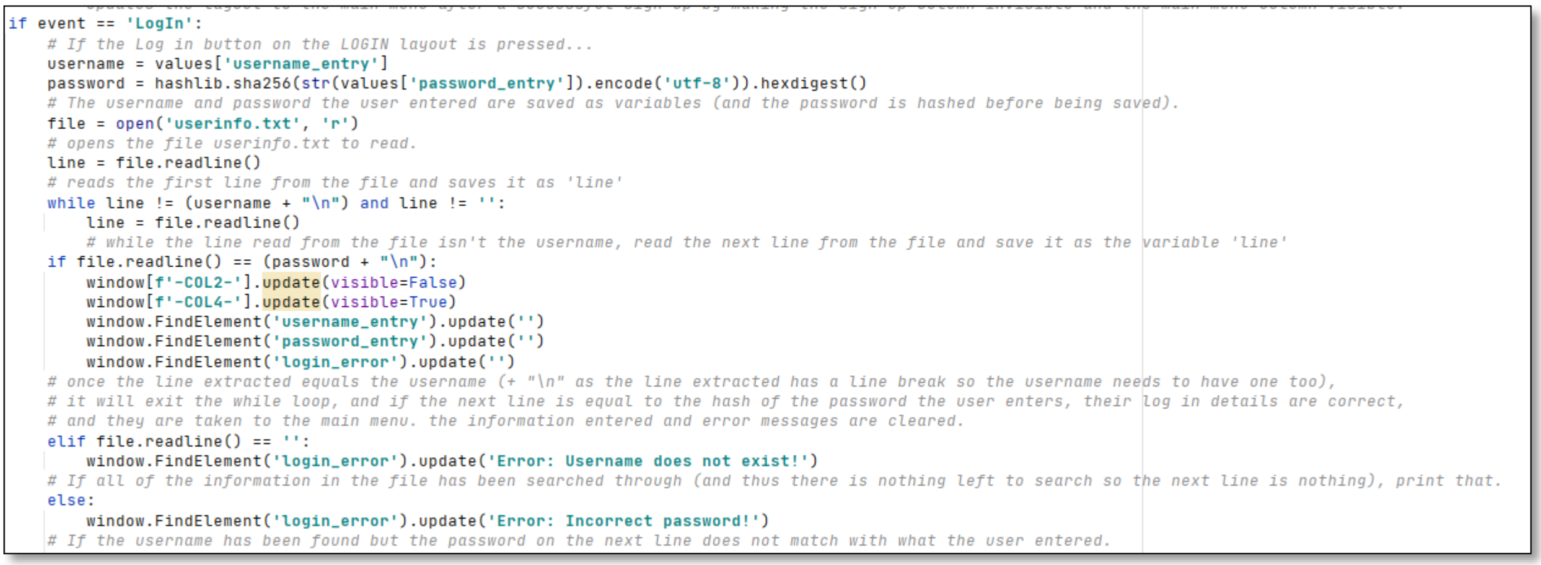
I then tried this again with a username that did not exist in the file (‘123’) and the program crashed (‘Initial Log In Test 1’ in testing evidence powerpoint). This is because the program enters an infinite loop as the username can never be found. To limit this, I have edited the whole loop to add a condition to keep searching until an empty line is found (as this will be the end of the information in the file, as a user has to enter something for all fields when they sign up):



```
file = open('userinfo.txt', 'r')
# opens the file userinfo.txt to read.
line = file.readline()
# reads the first line from the file and saves it as 'line'
while line != (username + "\n") and line != '':
    line = file.readline()
    # while the line read from the file isn't the username, read the next line from the file and save it as the variable 'line'
if file.readline() == (password + "\n"):
    print("correct details")
# once the line extracted equals the username (+ "\n" as the line extracted has a line break so the username needs to have one too),
# it will exit the while loop, and if the next line is equal to the hash of the password the user enters, their log in details are correct.
elif file.readline() == '':
    print('username does not exist')
# If all of the information in the file has been searched through (and thus there is nothing left tio search so the next line is nothing), print that.
else:
    print("incorrect password")
# If the username has been found but the password on the next line does not match with what the user entered.
```

Now when I run the program with the same inputs as in the previous test, the correct statements are printed (‘Initial Log In Test 2’ in testing evidence powerpoint).

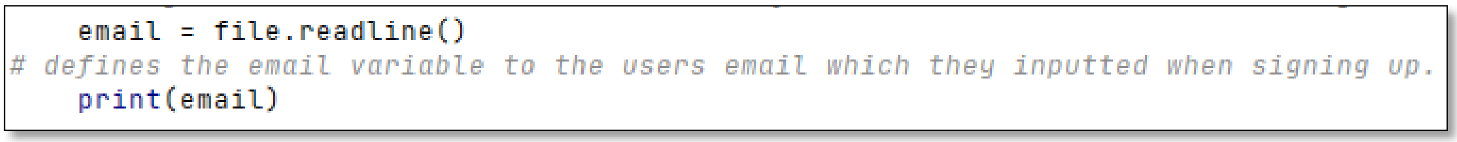
Now I know that the backbone of the log-in process is working, I can replace the print statements with what I actually want the program to do in each of the scenarios:



```
if event == 'LogIn':
    # If the Log in button on the LOGIN layout is pressed...
    username = values['username_entry']
    password = hashlib.sha256(str(values['password_entry']).encode('utf-8')).hexdigest()
    # The username and password the user entered are saved as variables (and the password is hashed before being saved).
    file = open('userinfo.txt', 'r')
    # opens the file userinfo.txt to read.
    line = file.readline()
    # reads the first line from the file and saves it as 'line'
    while line != (username + "\n") and line != '':
        line = file.readline()
        # while the line read from the file isn't the username, read the next line from the file and save it as the variable 'line'
    if file.readline() == (password + "\n"):
        window[f'-COL2-'].update(visible=False)
        window[f'-COL4-'].update(visible=True)
        window.FindElement('username_entry').update('')
        window.FindElement('password_entry').update('')
        window.FindElement('login_error').update('')
    # once the line extracted equals the username (+ "\n" as the line extracted has a line break so the username needs to have one too),
    # it will exit the while loop, and if the next line is equal to the hash of the password the user enters, their log in details are correct,
    # and they are taken to the main menu. the information entered and error messages are cleared.
    elif file.readline() == '':
        window.FindElement('login_error').update('Error: Username does not exist!')
    # If all of the information in the file has been searched through (and thus there is nothing left to search so the next line is nothing), print that.
    else:
        window.FindElement('login_error').update('Error: Incorrect password!')
    # If the username has been found but the password on the next line does not match with what the user entered.
```

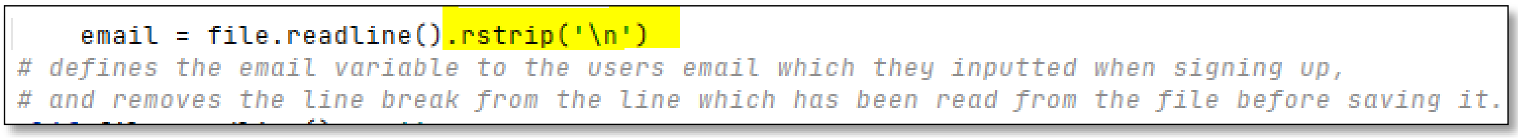
Now when the correct details are entered, this prototype should take the user to the main menu and reset the inputs/error message ready for the next user.

I also need to assign the email variable upon logging in so that it can be used in features that I will develop in the future. The way the information is stored, as stated earlier, means the email will be the data in the line after the password, so once the password has been read from the file and verified, if it I correct I can then just read the next line and set the email variable equal to it:



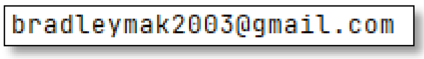
```
    email = file.readline()
# defines the email variable to the users email which they inputted when signing up.
    print(email)
```

However, when I run the program, the email which is saved includes a line break (which it cannot do as it needs to be referenced for future features, such as email notifications). Therefore, I need to remove the line break before it is saved to the variable. I did some research and found the website https://stackoverflow.com/questions/15233340/getting-rid-of-n-when-using-readlines, which said that I have to use the .rstrip() function to remove the line break from the line which has been read from the file. This is it implemented:



```
    email = file.readline().rstrip('\n')
# defines the email variable to the users email which they inputted when signing up,
# and removes the line break from the line which has been read from the file before saving it.
```

And now the correct email is printed with no line break, so I can remove the print statement as I know that this feature is working.



```
bradleymak2003@gmail.com
```

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
3.2.1	Entering username and password into relevant fields	String 'username123' String 'password123' **VALID DATA**	App will allow user to enter these strings into the relevant input boxes	Checking whether the user can enter a string into the textual input boxes that should appear on screen.	**Allows the user to enter these details into the input boxes successfully.**
3.2.2	Selection of button to confirm account details	Mouse click	Layout will be updated to MAINMENU and user will be taken to the main menu.	To check whether the button takes the user to the main menu of the app if their log-in details are valid and correct.	**Layout is updated to main menu if correct details are entered.**

Testing evidence can be found in the testing evidence powerpoint.

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
3.3.1	Entering incorrect password	String 'username123' String 'password1' **INVALID DATA**	App will allow user to enter these strings into the relevant input boxes	Checking whether the user can enter a string into the textual input boxes that should appear on screen.	**Allows the user to enter these details into the input**

					boxes successfully.
3.3.2	Selection of button to confirm account details	Mouse click	Error message will appear at bottom of screen stating invalid password, and layout will remain the same.	To check whether the confirm account button takes the user to the main menu if the details they enter are incorrect.	**Error message appears telling the user their password is invalid.**
3.3.3	Entering username that doesn't exist	String 'username1' String 'password123' **INVALID DATA**	App will allow user to enter these strings into the relevant input boxes	Checking whether the user can enter a string into the textual input boxes that should appear on screen.	**Allows the user to enter these details into the input boxes successfully.**
3.3.4	Selection of button to confirm account details	Mouse click	Error message will appear at bottom of screen stating that the username doesn't exist.	To check whether the confirm account button takes the user to the main menu if the details they enter are incorrect.	**Error message appears telling the user their username doesn't exist.**

Testing evidence can be found in the testing evidence powerpoint.

Log in – Review 2

Success Criteria

I have now completed the log in feature which allows the user to log into their account if they have created one and the details they enter are correct. It is robust and provides correct error messages in the correct scenarios. So far I have met:

- SC10 - The stakeholder would like the system to be user-friendly and easy to understand.
- SC1 - The stakeholder would like the system to allow them to create an account if they haven't already got one.
- SC2 - The stakeholder would like the system to allow them to log in to their account if they have already created one.

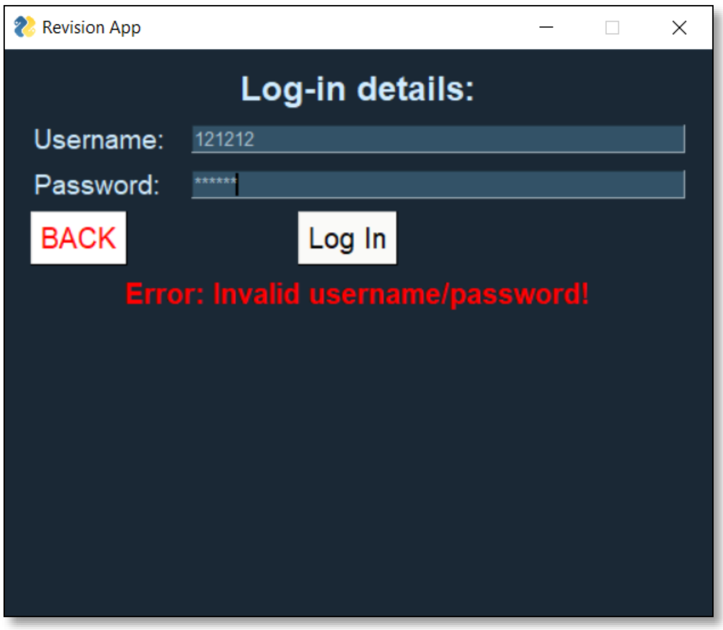
Stakeholder feedback

I sent this prototype of my solution to two of my stakeholders, Sam and Vivek, to test and provide feedback on any errors/issues they may occur. Vivek had no issues and said that the system worked perfectly and provided the correct error messages and did not crash when he tried to crash the app. Sam said the same but said that he feels it'd be more secure to provide the error message 'Invalid username/password' for both if the username doesn't exist or if the password is incorrect. He thought this because it makes the app less secure to tell a potential hacker whether it is the password or username that is incorrect (for example, if the 'password incorrect' message appears, the hacker will know there is an account with the username they entered, whereas if there was the same message for both scenarios the hacker would not know).

I agree with same idea and so I have changed the error message to 'Invalid username/password' for both when the user enters a username that doesn't exist and when the user enters an incorrect password in order to make my solution even more secure.

```
elif file.readline() == '':
    window.FindElement('login_error').update('Error: Invalid username/password!')
    If all of the information in the file has been searched through (and thus there is nothing left to search so the next line is nothing), print that.
else:
    window.FindElement('login_error').update('Error: Invalid username/password!')
    If the username has been found but the password on the next line does not match with what the user entered.
```

The error message now appears like this on the interface:



Revision App

Log-in details:

Username: 121212

Password: ******

BACK

Log In

Error: Invalid username/password!

Main Menu – Stage 1

Next, I am going to create the main menu for my program. As stated earlier, this is the main hub of the app as it is where all of the main features can be accessed from via a button press. To start with, I have created the layout:

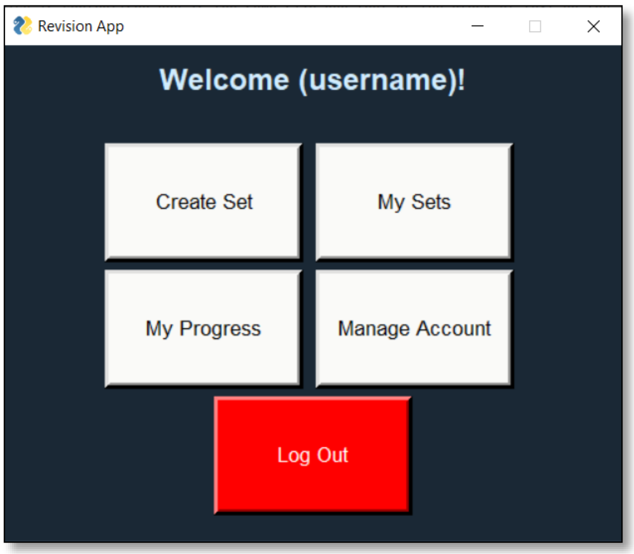
```
MainMenu = [[sg.Text('Welcome (username)!', key='welcomemsg', font=("Helvetica", 18, "bold"), size=(45, 2), justification='center')],
            # Creates a centred welcome message for the layout which will appear at the top of the screen.
            [sg.Text("            "), sg.Button('Create Set', font=("Helvetica", 13), key='CREATESET', border_width=5, size=(16, 4)), sg.Button('My Sets', font=("Helvetica", 13), key='MYSETS', border_width=5, size=(16,
            [sg.Text("            "), sg.Button('My Progress', font=("Helvetica", 13), key='PROGRESS', border_width=5, size=(16, 4)), sg.Button('Manage Account', font=("Helvetica", 13), key='ACCOUNT', border_width=5, s
            [sg.Text("                                   "), sg.Button('Log Out', font=("Helvetica", 13), key='LOGOUT', border_width=5, size=(16, 4), button_color=('white', 'red'))]]
#           Defines the Main Menu layout which consists of 5 symmetrically placed buttons on the screen with large text at the top welcoming the user.
```

I have used a similar design to the final Start Menu design as this was a layout my stakeholders liked and was created after consulting them for their feedback (i.e. larger text and buttons).

This layout is also placed in the column layout so that it can be made visible/invisible as required:

```
layout = [[sg.Column(StartScreen, key='-COL1-', visible=True), sg.Column(Login, visible=False, key='-COL2-'), sg.Column(Signup, visible=False, key='-COL3-'), sg.Column(MainMenu, visible=False, key='-COL4-')]]
# Defines the 'column' layouts which can be made visible/invisible to the user as required to give the impression of the program switching layouts.
```

Now when the program is ran and I log in using an existing account, this is the main menu layout which appears on the screen:



The username has not been inserted into the welcome message yet, but this will be implemented later. For now, I have used a placeholder (username) to represent where the user's username would appear.

I have decided to remove the exit button from the main menu as it is already present on the start screen and it makes more logical sense to sign out before exiting the application, and thus it makes the menu look less cluttered and reduces the amount of code needed to be written.

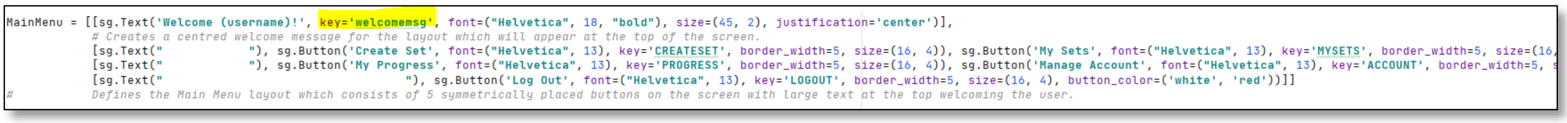
I am now ready to carry out the first iteration of my main menu testing:

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
4.1.1	Selection of option via 'new_set' button	Mouse click	Nothing will happen as the functionality of the button has not been implemented yet, and the SETNAME layout has not been defined.	Checking whether the button appears in the correct format on screen and can be interacted with by the user.	Nothing happens as functionality not implemented yet but button is visible in correct position and pressable.
4.1.2	Selection of option via 'my_sets' button	Mouse click	Nothing will happen as the functionality of the button has not been implemented yet, and the SETS layout has not been defined.	Checking whether the button appears in the correct format on screen and can be interacted with by the user.	Nothing happens as functionality not implemented yet but button is visible in correct position and pressable.
4.1.3	Selection of option via 'progress_tracker' button	Mouse click	Nothing will happen as the functionality of the button has not been implemented yet, and the PROGRESS layout has not been defined.	Checking whether the button appears in the correct format on screen and can be interacted with by the user.	Nothing happens as functionality not implemented yet but button is visible in correct position and pressable.
4.1.4	Selection of option via 'manage_account' button	Mouse click	Nothing will happen as the functionality of the button has not been implemented yet, and the ACCOUNT layout has not been defined.	Checking whether the button appears in the correct format on screen and can be interacted with by the user.	Nothing happens as functionality not implemented yet but button is visible in correct position and pressable.

4.1.5	Selection of option via 'log_out' button	Mouse click	Nothing will happen as the functionality of the button has not been implemented yet.	Checking whether the button appears in the correct format on screen and can be interacted with by the user.	Nothing happens as functionality not implemented yet but button is visible in correct position and pressable.
4.1.6	Selection of option via 'Exit' button	Mouse click	Nothing will happen as the functionality of the button has not been implemented yet.	Checking whether the button appears in the correct format on screen and can be interacted with by the user.	Button not present due to a change in my original plan (as detailed in the development section).

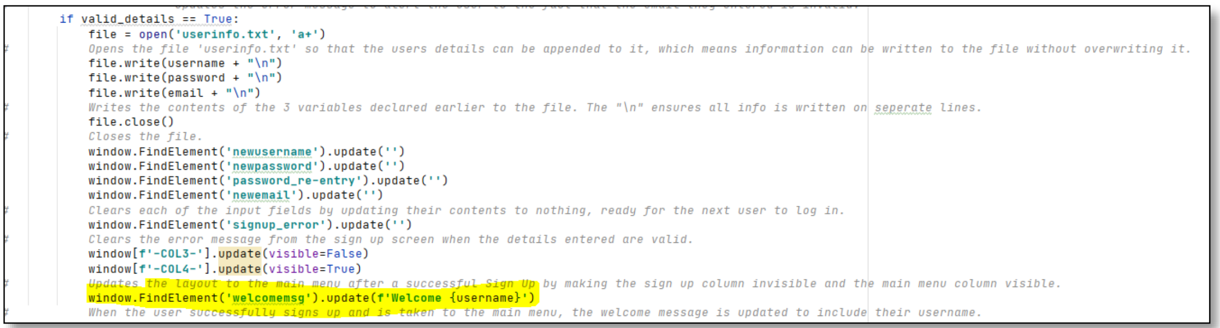
Main Menu – Stage 2

Next, I am going to implement the functionality of the main menu. First, I need to update the text at the top of the screen to display the user's username. First, I have given the text element a key so it can be identified and updated using using the update element command:

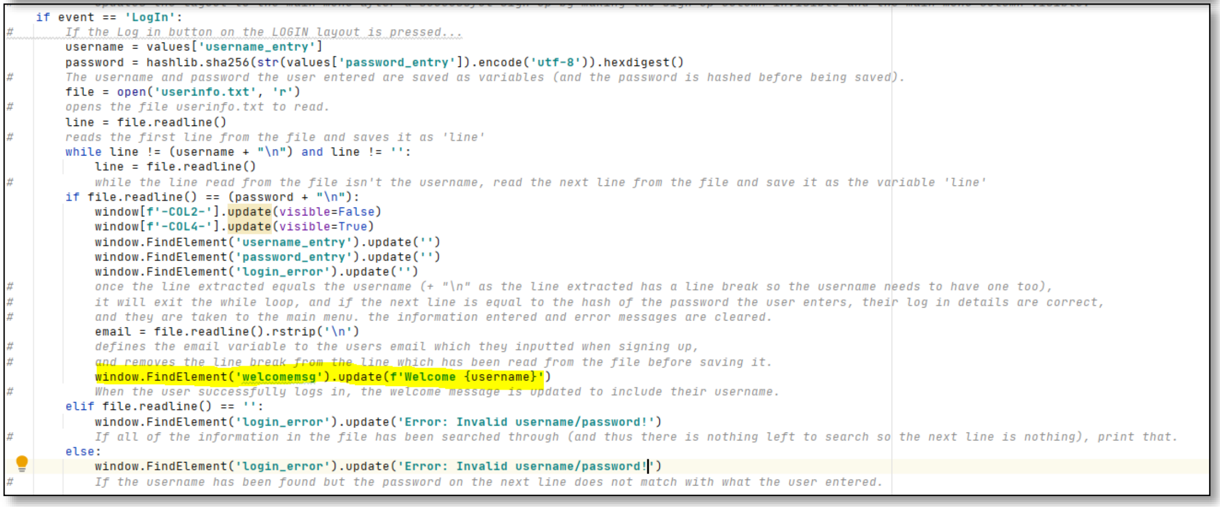


```
MainMenu = [[sg.Text('Welcome (username)!', key='welcomemsg', font=("Helvetica", 18, "bold"), size=(45, 2), justification='center')],
            # Creates a centred welcome message for the layout which will appear at the top of the screen.
            [sg.Text("            "), sg.Button('Create Set', font=("Helvetica", 13), key='CREATESET', border_width=5, size=(16, 4)), sg.Button('My Sets', font=("Helvetica", 13), key='MYSETS', border_width=5, size=(16,
            [sg.Text("            "), sg.Button('My Progress', font=("Helvetica", 13), key='PROGRESS', border_width=5, size=(16, 4)), sg.Button('Manage Account', font=("Helvetica", 13), key='ACCOUNT', border_width=5, s
            [sg.Text("                                  "), sg.Button('Log Out', font=("Helvetica", 13), key='LOGOUT', border_width=5, size=(16, 4), button_color=('white', 'red'))]]
#           Defines the Main Menu layout which consists of 5 symmetrically placed buttons on the screen with large text at the top welcoming the user.
```

I have then updated the element whenever the user successfully creates an account/logs in to show their username:

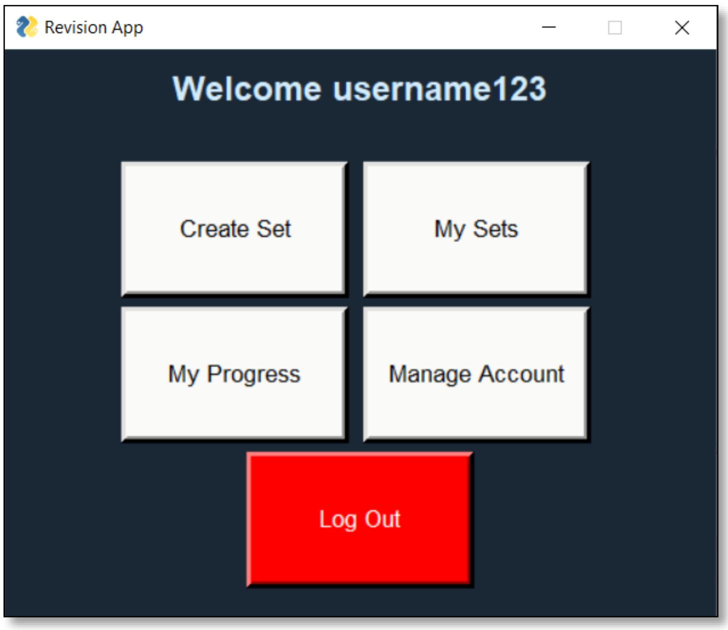


```
    if valid_details == True:
        file = open('userinfo.txt', 'a+')
#       Opens the file 'userinfo.txt' so that the users details can be appended to it, which means information can be written to the file without overwriting it.
        file.write(username + "\n")
        file.write(password + "\n")
        file.write(email + "\n")
#       Writes the contents of the 3 variables declared earlier to the file. The "\n" ensures all info is written on seperate lines.
        file.close()
#       Closes the file.
        window.FindElement('newusername').update('')
        window.FindElement('newpassword').update('')
        window.FindElement('password_re-entry').update('')
        window.FindElement('newemail').update('')
#       Clears each of the input fields by updating their contents to nothing, ready for the next user to log in.
        window.FindElement('signup_error').update('')
#       Clears the error message from the sign up screen when the details entered are valid.
        window[f'-COL3-'].update(visible=False)
        window[f'-COL4-'].update(visible=True)
#       Updates the layout to the main menu after a successful Sign Up by making the sign up column invisible and the main menu column visible.
        window.FindElement('welcomemsg').update(f'Welcome {username}')
#       When the user successfully signs up and is taken to the main menu, the welcome message is updated to include their username.
```

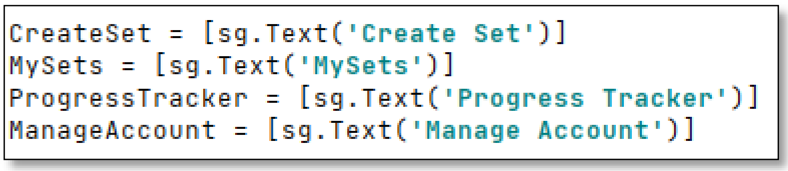


```
if event == 'LogIn':
#   If the Log in button on the LOGIN layout is pressed...
    username = values['username_entry']
    password = hashlib.sha256(str(values['password_entry']).encode('utf-8')).hexdigest()
#   The username and password the user entered are saved as variables (and the password is hashed before being saved).
    file = open('userinfo.txt', 'r')
#   opens the file userinfo.txt to read.
    line = file.readline()
#   reads the first line from the file and saves it as 'line'
    while line != (username + "\n") and line != '':
        line = file.readline()
#       while the line read from the file isn't the username, read the next line from the file and save it as the variable 'line'
    if file.readline() == (password + "\n"):
        window[f'-COL2-'].update(visible=False)
        window[f'-COL4-'].update(visible=True)
        window.FindElement('username_entry').update('')
        window.FindElement('password_entry').update('')
        window.FindElement('login_error').update('')
#       once the line extracted equals the username (+ "\n" as the line extracted has a line break so the username needs to have one too),
#       it will exit the while loop, and if the next line is equal to the hash of the password the user enters, their log in details are correct,
#       and they are taken to the main menu. the information entered and error messages are cleared.
        email = file.readline().rstrip('\n')
#       defines the email variable to the users email which they inputted when signing up,
#       and removes the line break from the line which has been read from the file before saving it.
        window.FindElement('welcomemsg').update(f'Welcome {username}')
#       When the user successfully logs in, the welcome message is updated to include their username.
    elif file.readline() == '':
        window.FindElement('login_error').update('Error: Invalid username/password!')
#       If all of the information in the file has been searched through (and thus there is nothing left to search so the next line is nothing), print that.
    else:
        window.FindElement('login_error').update('Error: Invalid username/password!')
#       If the username has been found but the password on the next line does not match with what the user entered.
```

Now, when I log into the account with the username ‘username123’, the message is correctly displayed at the top of the screen:

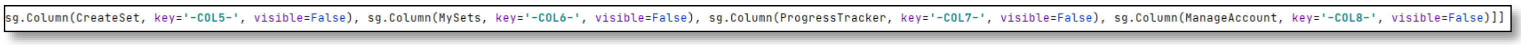


Next, I have created temporary placeholder layouts for each of the features to allow me to test the functionality of the buttons:



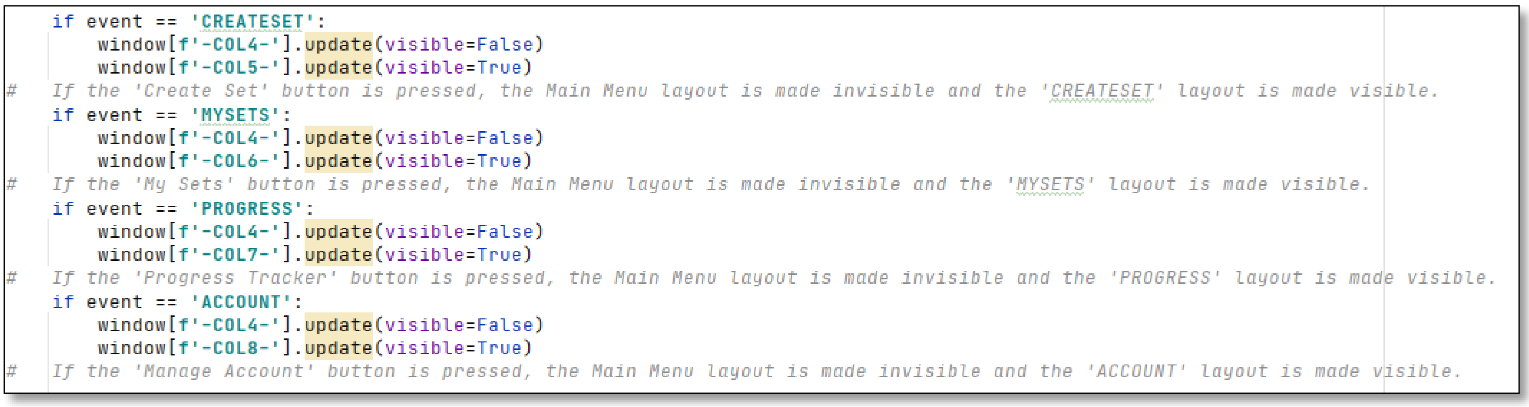
```
CreateSet = [sg.Text('Create Set')]
MySets = [sg.Text('MySets')]
ProgressTracker = [sg.Text('Progress Tracker')]
ManageAccount = [sg.Text('Manage Account')]
```

I have also added these layouts as columns:



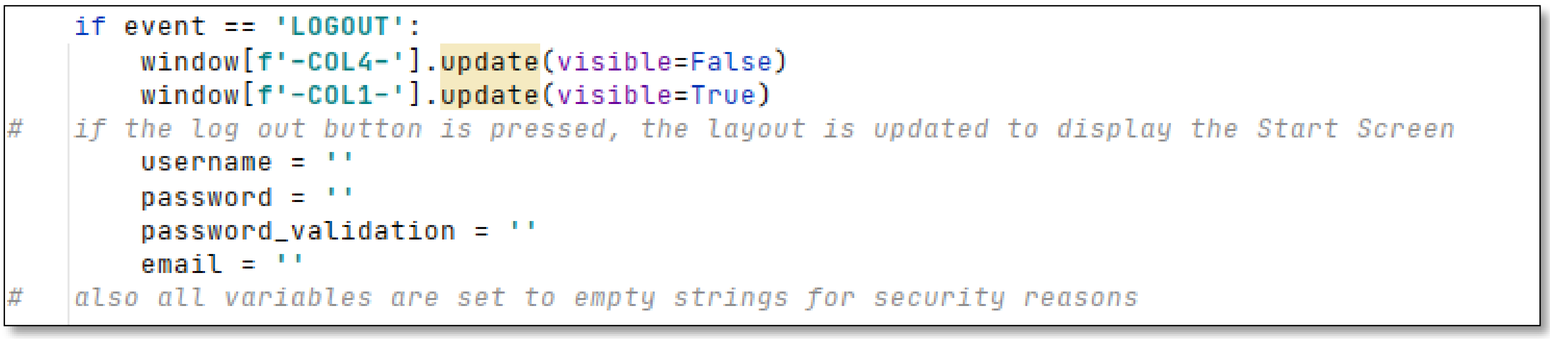
```
sg.Column(CreateSet, key='-COL5-', visible=False), sg.Column(MySets, key='-COL6-', visible=False), sg.Column(ProgressTracker, key='-COL7-', visible=False), sg.Column(ManageAccount, key='-COL8-', visible=False)]]
```

I have then told the program that if one of the buttons is pressed, the relevant columns (layouts) should be made visible/invisible:



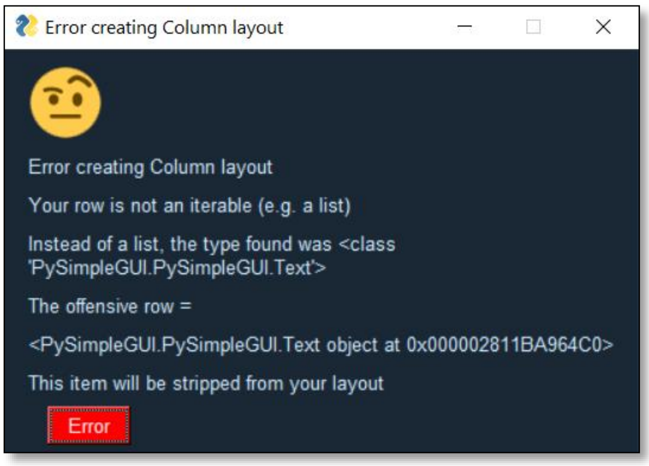
```
    if event == 'CREATESET':
        window[f'-COL4-'].update(visible=False)
        window[f'-COL5-'].update(visible=True)
#   If the 'Create Set' button is pressed, the Main Menu layout is made invisible and the 'CREATESET' layout is made visible.
    if event == 'MYSETS':
        window[f'-COL4-'].update(visible=False)
        window[f'-COL6-'].update(visible=True)
#   If the 'My Sets' button is pressed, the Main Menu layout is made invisible and the 'MYSETS' layout is made visible.
    if event == 'PROGRESS':
        window[f'-COL4-'].update(visible=False)
        window[f'-COL7-'].update(visible=True)
#   If the 'Progress Tracker' button is pressed, the Main Menu layout is made invisible and the 'PROGRESS' layout is made visible.
    if event == 'ACCOUNT':
        window[f'-COL4-'].update(visible=False)
        window[f'-COL8-'].update(visible=True)
#   If the 'Manage Account' button is pressed, the Main Menu layout is made invisible and the 'ACCOUNT' layout is made visible.
```

I have next implemented the functionality of the log out button, by updating the layout to the Start Screen and setting all of the variables to empty strings (for security reasons):



```
    if event == 'LOGOUT':
        window[f'-COL4-'].update(visible=False)
        window[f'-COL1-'].update(visible=True)
#   if the log out button is pressed, the layout is updated to display the Start Screen
        username = ''
        password = ''
        password_validation = ''
        email = ''
#   also all variables are set to empty strings for security reasons
```

When I then ran the program to carry out the second iteration of testing for the main menu, I ran into an error:



Error creating Column layout

Error creating Column layout

Your row is not an iterable (e.g. a list)

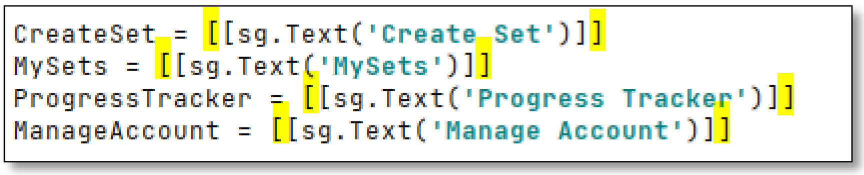
Instead of a list, the type found was <class 'PySimpleGUI.PySimpleGUI.Text'>

The offensive row =

<PySimpleGUI.PySimpleGUI.Text object at 0x000002811BA964C0>

This item will be stripped from your layout

I realised that this was because I hadn't put two square brackets around the temporary layouts I made earlier, so I added them and it solved this issue:



```
CreateSet = [[sg.Text('Create Set')]]
MySets = [[sg.Text('MySets')]]
ProgressTracker = [[sg.Text('Progress Tracker')]]
ManageAccount = [[sg.Text('Manage Account')]]
```

Now when I ran the program and carried out the following tests (evidence for which can be found in the testing evidence PowerPoint), there were no errors.

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
4.2.1	Selection of option via 'new_set' button	Mouse click	Layout will be updated to SETNAME, so the user will be taken to a different screen.	Checking whether the button functions properly (i.e. updates the layout to the corresponding option)	The layout was updated correctly.
4.2.2	Selection of option via 'my_sets' button	Mouse click	Layout will be updated to SETS, so the user will be taken to a different screen.	Checking whether the button functions properly (i.e. updates the layout to the corresponding option)	The layout was updated correctly.
4.2.3	Selection of option via 'progress_tracker' button	Mouse click	Layout will be updated to PROGRESS, so the user will be taken to a different screen.	Checking whether the button functions properly (i.e. updates the layout to the corresponding option)	The layout was updated correctly.
4.2.4	Selection of option via 'manage_account' button	Mouse click	Layout will be updated to ACCOUNT, so the user will be taken to a different screen.	Checking whether the button functions properly (i.e. updates the layout to the corresponding option)	The layout was updated correctly.
4.2.5	Selection of option via 'log_out' button	Mouse click	User will be logged out and taken back to the start menu (layout is updated to STARTMENU)	Checking whether the button functions properly (i.e. updates the layout to the corresponding option)	The layout was updated correctly, and the user was logged out.
4.2.6	Selection of option via 'Exit' button	Mouse click	App will close as event loop is broken. User will be automatically logged out.	Checking whether the button functions properly (i.e. updates the layout to the corresponding option)	Button not present due to a change in my original plan (as detailed in the development section).

Main Menu – Review

Stakeholder feedback

I consulted two of my stakeholders, Vivek and Marcus, about the main menu design as this is an important feature of my program and it a key success criterion. They both said they like the layout as it matches their requests made earlier for previous layouts. They also said that the buttons work as intended.

Success Criteria

After creating the layout of the main menu, I have met:

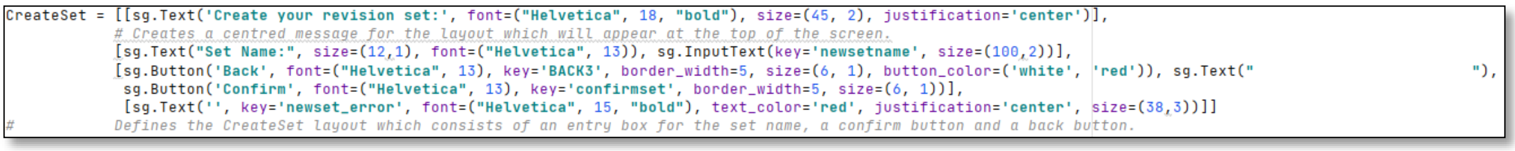
- SC10 - The stakeholder would like the system to be user-friendly and easy to understand.
- SC1 - The stakeholder would like the system to allow them to create an account if they haven't already got one.
- SC2 - The stakeholder would like the system to allow them to log in to their account if they have already created one.
- SC9 - The stakeholder would like the system to have a clear main menu.

Changes made to original plan

I have removed the Exit button from the main menu (as stated earlier) to reduce the amount of code that needs to be written, reduce clutter and increase security (by encouraging the user to log out before exiting the program), as it is not needed anyway.

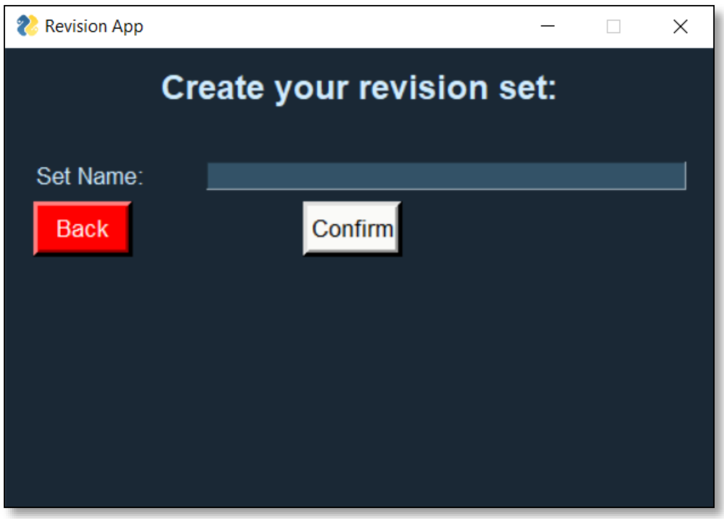
Create Set – Stage 1

Firstly, I need to create the CreateSet layout. The layout will follow the general convention of previous layouts as this is what my stakeholders appear to like.

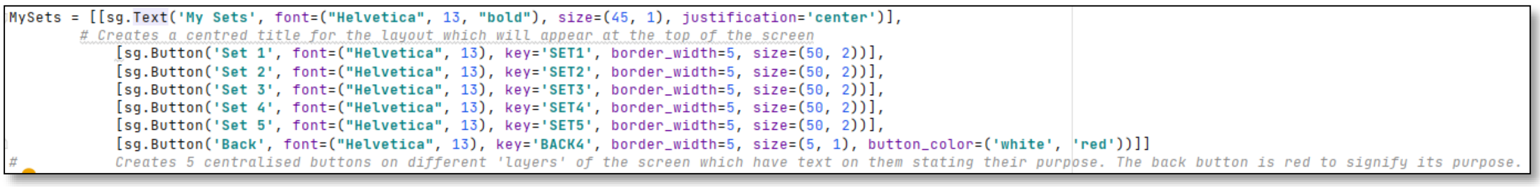


```
CreateSet = [[sg.Text('Create your revision set:', font=("Helvetica", 18, "bold"), size=(45, 2), justification='center')],
             # Creates a centred message for the layout which will appear at the top of the screen.
             [sg.Text("Set Name:", size=(12,1), font=("Helvetica", 13)), sg.InputText(key='newsetname', size=(100,2))],
             [sg.Button('Back', font=("Helvetica", 13), key='BACK3', border_width=5, size=(6, 1), button_color=('white', 'red')), sg.Text("                        "),
              sg.Button('Confirm', font=("Helvetica", 13), key='confirmset', border_width=5, size=(6, 1))],
              [sg.Text('', key='newset_error', font=("Helvetica", 15, "bold"), text_color='red', justification='center', size=(38,3))]]
#            Defines the CreateSet layout which consists of an entry box for the set name, a confirm button and a back button.
```

The result is a layout looking like this:

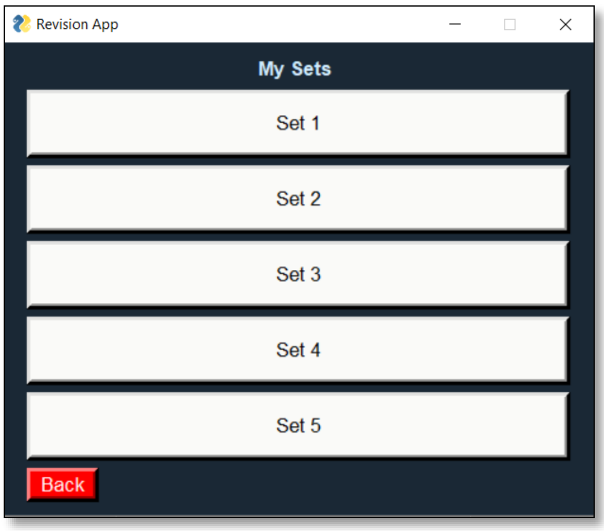


I have also created the MySets layout to allow me to test whether the names of sets being created are being written to the csv file successfully and the set names are displayed on the screen correctly.



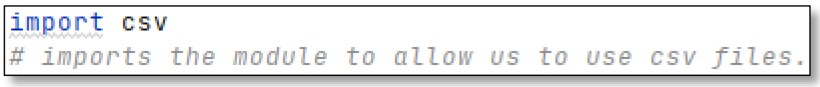
```
MySets = [[sg.Text('My Sets', font=("Helvetica", 13, "bold"), size=(45, 1), justification='center')],
         # Creates a centred title for the layout which will appear at the top of the screen
             [sg.Button('Set 1', font=("Helvetica", 13), key='SET1', border_width=5, size=(50, 2))],
             [sg.Button('Set 2', font=("Helvetica", 13), key='SET2', border_width=5, size=(50, 2))],
             [sg.Button('Set 3', font=("Helvetica", 13), key='SET3', border_width=5, size=(50, 2))],
             [sg.Button('Set 4', font=("Helvetica", 13), key='SET4', border_width=5, size=(50, 2))],
             [sg.Button('Set 5', font=("Helvetica", 13), key='SET5', border_width=5, size=(50, 2))],
             [sg.Button('Back', font=("Helvetica", 13), key='BACK4', border_width=5, size=(5, 1), button_color=('white', 'red'))]]
#            Creates 5 centralised buttons on different 'layers' of the screen which have text on them stating their purpose. The back button is red to signify its purpose.
```

The resultant layout is as shown here (the set buttons are so big to allow room for the name of sets to be added once they are created):



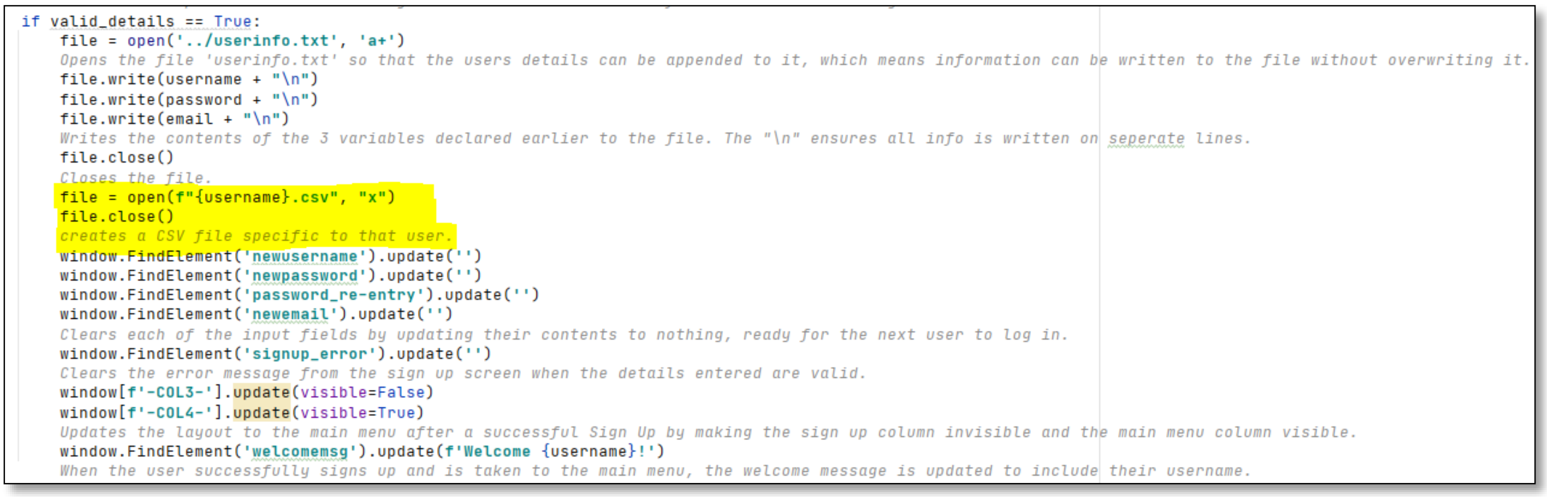
These layouts follow the same design as previous layouts which have been decided and modified by my stakeholders, and so I do not feel the need to contact my stakeholders about the design of these layouts.

I have also realised that no CSV file is being made upon account creation, so to fix this, I need to add a line of code that creates a CSV file specific to that user once their account has been created. Firstly, I have imported the csv library to allow us to use csv files in python:



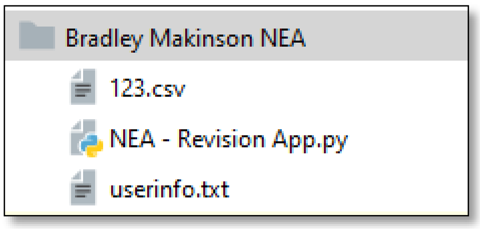
```
import csv
# imports the module to allow us to use csv files.
```

I have then added a piece of code which creates a CSV file called the users username (which must be unique) upon account creation:



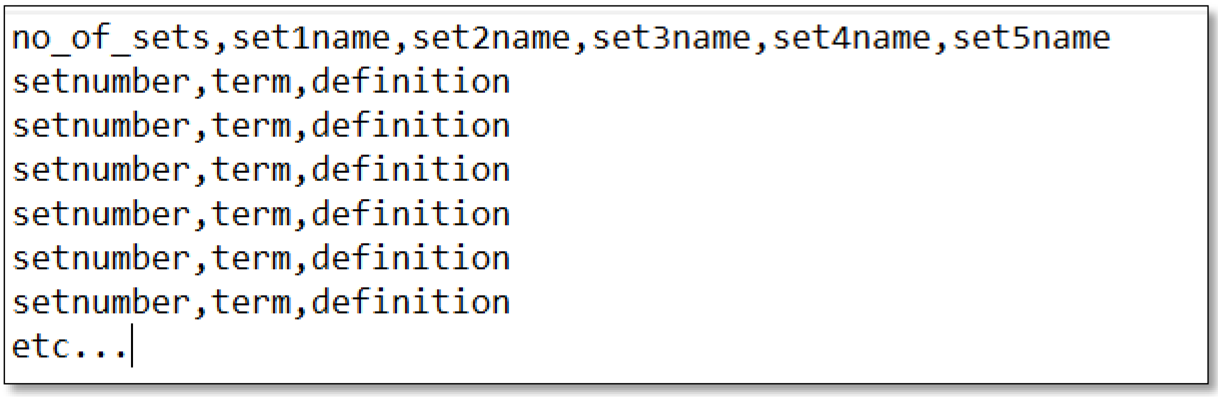
```
if valid_details == True:
    file = open('../userinfo.txt', 'a+')
    Opens the file 'userinfo.txt' so that the users details can be appended to it, which means information can be written to the file without overwriting it.
    file.write(username + "\n")
    file.write(password + "\n")
    file.write(email + "\n")
    Writes the contents of the 3 variables declared earlier to the file. The "\n" ensures all info is written on seperate lines.
    file.close()
    Closes the file.
    file = open(f"{username}.csv", "x")
    file.close()
    creates a CSV file specific to that user.
    window.FindElement('newusername').update('')
    window.FindElement('newpassword').update('')
    window.FindElement('password_re-entry').update('')
    window.FindElement('newemail').update('')
    Clears each of the input fields by updating their contents to nothing, ready for the next user to log in.
    window.FindElement('signup_error').update('')
    Clears the error message from the sign up screen when the details entered are valid.
    window[f'-COL3-'].update(visible=False)
    window[f'-COL4-'].update(visible=True)
    Updates the layout to the main menu after a successful Sign Up by making the sign up column invisible and the main menu column visible.
    window.FindElement('welcomemsg').update(f'Welcome {username}!')
    When the user successfully signs up and is taken to the main menu, the welcome message is updated to include their username.
```

I have tested this and when I create a new account with the username '123', a new CSV file is created called '123.csv' as intended:



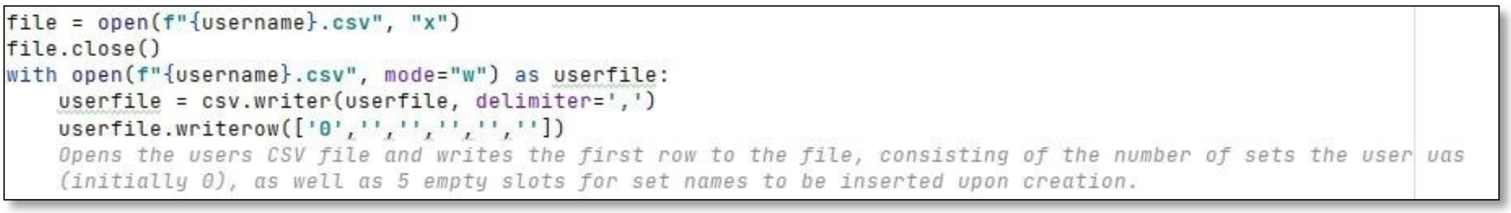
```
Bradley Makinson NEA
    123.csv
    NEA - Revision App.py
    userinfo.txt
```

The way I am planning the CSV file for each user to be structured is as follows:



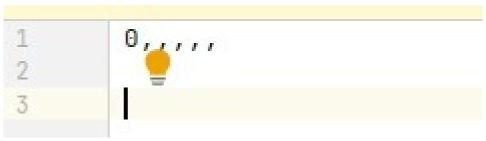
```
no_of_sets,set1name,set2name,set3name,set4name,set5name
setnumber,term,definition
setnumber,term,definition
setnumber,term,definition
setnumber,term,definition
setnumber,term,definition
setnumber,term,definition
etc...
```

So therefore, I will need write the initial number of sets (0) and set names (which will be nothing yet but I need to initialise their presence) to the users CSV file upon account creation.



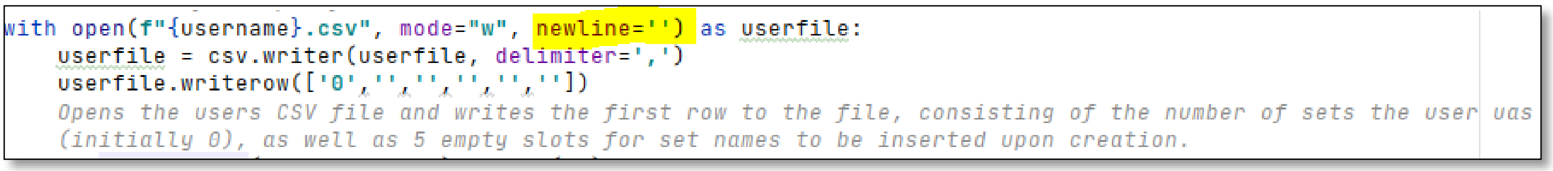
```
file = open(f"{username}.csv", "x")
file.close()
with open(f"{username}.csv", mode="w") as userfile:
    userfile = csv.writer(userfile, delimiter=',')
    userfile.writerow(['0','','','','',''])
    Opens the users CSV file and writes the first row to the file, consisting of the number of sets the user uas
    (initially 0), as well as 5 empty slots for set names to be inserted upon creation.
```

However, this then writes a blank line as well as the desired line of information into the file:



```
0,,,,,
```

I had a look online to solve this issue and found the website https://stackoverflow.com/questions/3348460/csv-file-written-with-python-has-blank-lines-between-each-row which made me realise that I need to add a newline parameter to stop this from happening:



```
with open(f"{username}.csv", mode="w", newline='') as userfile:
    userfile = csv.writer(userfile, delimiter=',')
    userfile.writerow(['0','','','','',''])
    Opens the users CSV file and writes the first row to the file, consisting of the number of sets the user was
    (initially 0), as well as 5 empty slots for set names to be inserted upon creation.
```

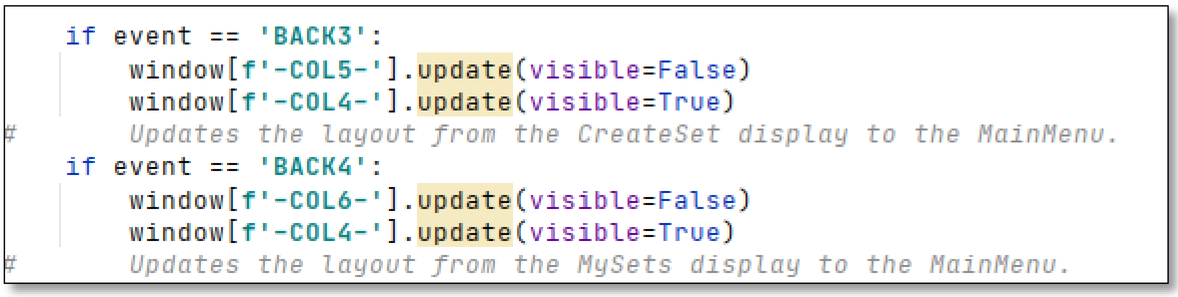
I am now ready to carry out iteration 1 of my testing of the create set feature (testing evidence of this can be found in the testing evidence powerpoint):

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
5.1.1	Entering name of set into textual input boxes	String 'Data Structures' **VALID DATA**	App will allow user to enter these strings into the relevant input boxes	Checking whether the user can enter a string into the textual input boxes that should appear on screen (there is no invalid string for a set name – it can be anything).	The program successfully allows the user to input a name for their set into the input box.
5.1.2	Selection of button to create set with the given name	Mouse click	Nothing will happen as the functionality of the button has not been implemented yet.	To check whether the confirm button on screen appears in the correct place and allows user interaction.	The confirm button appears in the correct place and is pressable. Nothing happens when it is clicked yet because its functionality has not been implemented.

Create Set – Stage 2

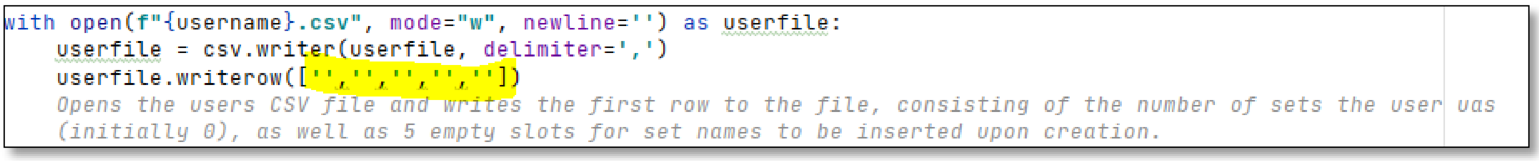
Now that we have everything in place, the next stage involves implementing the functionality of the create set feature.

I have started by quickly implementing the functionality of both back buttons on the two displays created in stage 1.



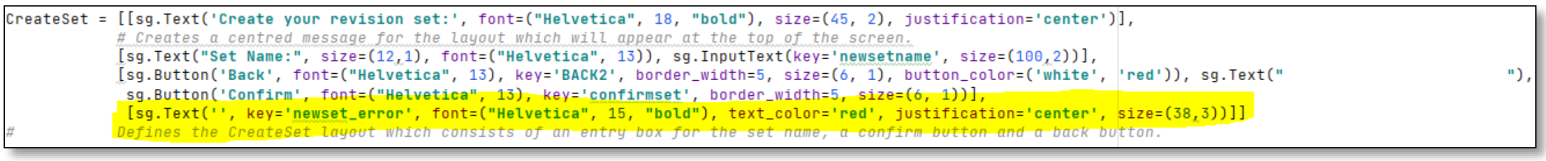
```
    if event == 'BACK3':
        window[f'-COL5-'].update(visible=False)
        window[f'-COL4-'].update(visible=True)
#       Updates the layout from the CreateSet display to the MainMenu.
    if event == 'BACK4':
        window[f'-COL6-'].update(visible=False)
        window[f'-COL4-'].update(visible=True)
#       Updates the layout from the MySets display to the MainMenu.
```

I have then decided to remove the number of sets from being stored in the users CSV file as I feel that it is not necessary and is thus just taking up unnecessary space – the program can just check for empty spaces in the name slots in the CSV file and if there is an empty space, the set name can be inserted there, and if not, the set limit has been reached.



```
with open(f"{username}.csv", mode="w", newline='') as userfile:
    userfile = csv.writer(userfile, delimiter=',')
    userfile.writerow(['','','','',''])
    Opens the users CSV file and writes the first row to the file, consisting of the number of sets the user was
    (initially 0), as well as 5 empty slots for set names to be inserted upon creation.
```

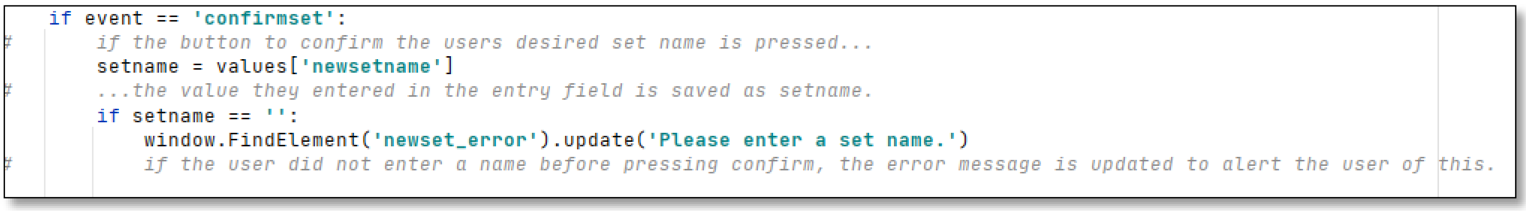
I next need to add an error message slot to the layout to inform the user if there has been an error:



```
CreateSet = [[sg.Text('Create your revision set:', font=("Helvetica", 18, "bold"), size=(45, 2), justification='center')],
            # Creates a centred message for the layout which will appear at the top of the screen.
            [sg.Text("Set Name:", size=(12,1), font=("Helvetica", 13)), sg.InputText(key='newsetname', size=(100,2))],
            [sg.Button('Back', font=("Helvetica", 13), key='BACK2', border_width=5, size=(6, 1), button_color=('white', 'red')), sg.Text("                    "),
             sg.Button('Confirm', font=("Helvetica", 13), key='confirmset', border_width=5, size=(6, 1))],
             [sg.Text('', key='newset_error', font=("Helvetica", 15, "bold"), text_color='red', justification='center', size=(38,3))]]
#           Defines the CreateSet layout which consists of an entry box for the set name, a confirm button and a back button.
```

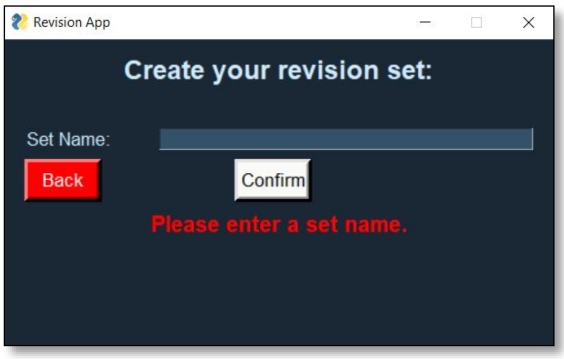
I next need to create some validation of the users input for their desired set name – ensure that the entry field is not left empty upon the confirm button being pressed, and ensure that the user has not exceeded their 5 set limit before creating the set.

I have started with the simpler of the two – checking if the user has inputted anything:



```
    if event == 'confirmset':
#       if the button to confirm the users desired set name is pressed...
        setname = values['newsetname']
#       ...the value they entered in the entry field is saved as setname.
        if setname == '':
            window.FindElement('newset_error').update('Please enter a set name.')
#           if the user did not enter a name before pressing confirm, the error message is updated to alert the user of this.
```

And when I press the submit button without inputting anything, the error message correctly displays on screen:



Revision App

Create your revision set:

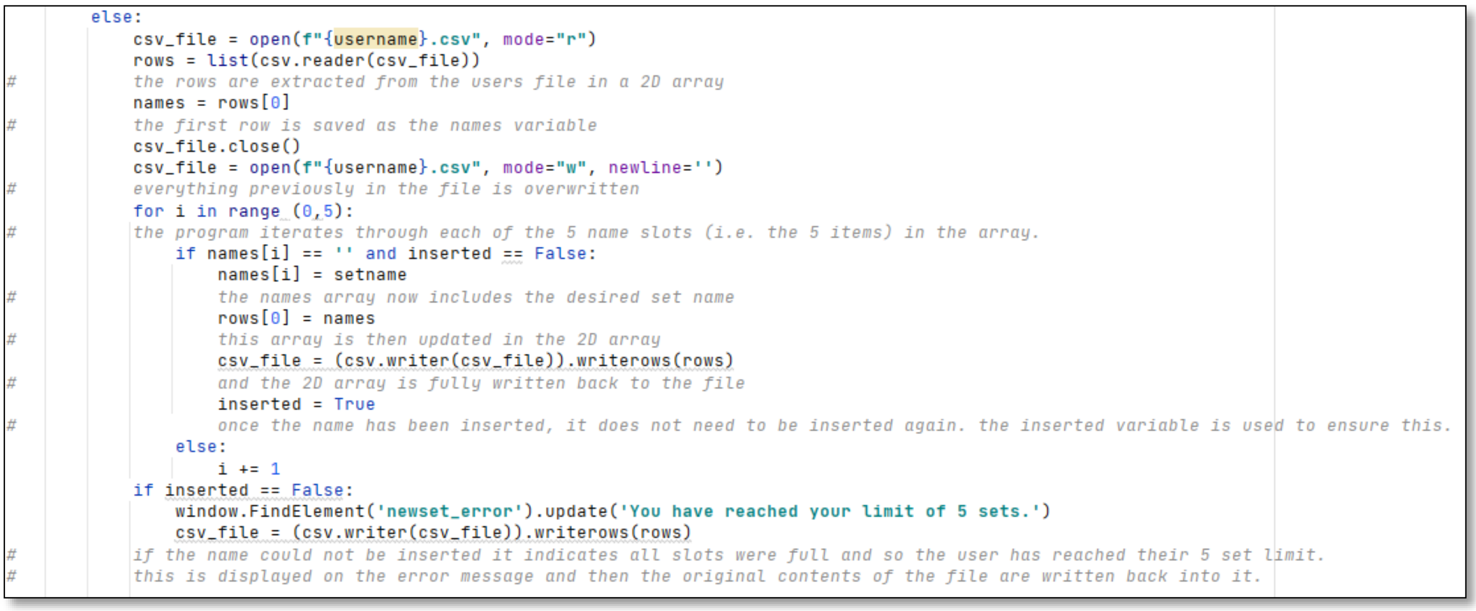
Set Name:

Back

Confirm

Please enter a set name.

Next, if the user has entered something as the set name, I have ensured that the user has not exceeded their 5 set limit:

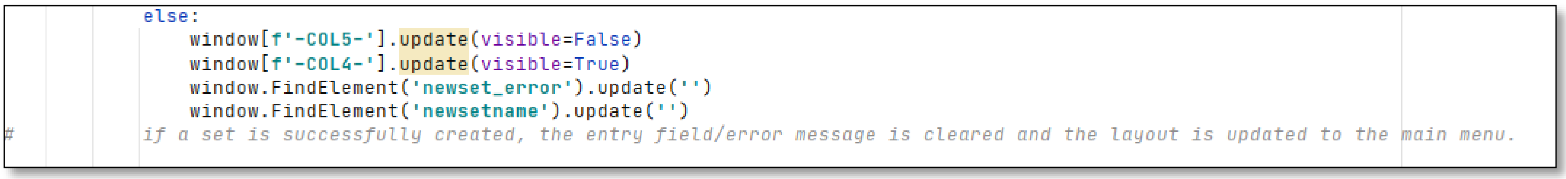


```
    else:
        csv_file = open(f"{username}.csv", mode="r")
        rows = list(csv.reader(csv_file))
#       the rows are extracted from the users file in a 2D array
        names = rows[0]
#       the first row is saved as the names variable
        csv_file.close()
        csv_file = open(f"{username}.csv", mode="w", newline='')
#       everything previously in the file is overwritten
        for i in range (0,5):
#       the program iterates through each of the 5 name slots (i.e. the 5 items) in the array.
            if names[i] == '' and inserted == False:
                names[i] = setname
#               the names array now includes the desired set name
                rows[0] = names
#               this array is then updated in the 2D array
                csv_file = (csv.writer(csv_file)).writerows(rows)
#               and the 2D array is fully written back to the file
                inserted = True
#               once the name has been inserted, it does not need to be inserted again. the inserted variable is used to ensure this.
            else:
                i += 1
        if inserted == False:
            window.FindElement('newset_error').update('You have reached your limit of 5 sets.')
            csv_file = (csv.writer(csv_file)).writerows(rows)
#       if the name could not be inserted it indicates all slots were full and so the user has reached their 5 set limit.
#       this is displayed on the error message and then the original contents of the file are written back into it.
```

I then tested my progress so far on the second iteration of the create set feature (testing evidence can be found in the powerpoint):

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
5.2.1	Entering name of set into textual input boxes	String 'Data Structures' **VALID DATA**	App will allow user to enter these strings into the relevant input boxes	Checking whether the user can enter a string into the textual input boxes that should appear on screen (there is no invalid string for a set name – it can be anything).	The program successfully allows the user to input a name for their set into the input box.
5.2.2	Selection of button to create set with the given name	Mouse click	Name of set will be written to the users personal .csv file, and the relevant button on the choose set layout will be updated to show the new set name.	To check whether the button correctly creates a set for the user with the given name, stores the name in the users .csv file, updates the button text on the 'MYSETS' layout, and updates the relevant variables (i.e. num_of_sets).	The program correctly stores the set name into the users csv file but does not yet update the buttons on the MySets layout.

I realised from these tests I need to clear the entry field whenever a field is created and take the user back to the main menu:



```
    else:
        window[f'-COL5-'].update(visible=False)
        window[f'-COL4-'].update(visible=True)
        window.FindElement('newset_error').update('')
        window.FindElement('newsetname').update('')
#   if a set is successfully created, the entry field/error message is cleared and the layout is updated to the main menu.
```

I have also decided that I am not going to implement the updating of the set buttons yet as this would be more effective if just done when the user presses the mysets button. Therefore, I have decided to conclude that the second iteration of my create set testing (test 5.2.2) has been successful.

Create Sets – Review

Stakeholder feedback
I contacted one of my stakeholders, Vivek, for his opinions on the overall experience of creating sets. He said that whilst the 5 set limit could be frustrating (this is something I will look into increasing further in the future), the program itself works effectively, the validation is effective, and the layout is simple and easy to understand. As a result, I am not going to make any changes to this feature as a result of stakeholder feedback.

Success Criteria
I have now completed the feature which allows users to create new sets of flashcards with a given name. In terms of my success criteria, I have so far met:

- SC10 - The stakeholder would like the system to be user-friendly and easy to understand.
- SC1 - The stakeholder would like the system to allow them to create an account if they haven't already got one.
- SC2 - The stakeholder would like the system to allow them to log in to their account if they have already created one.
- SC9 - The stakeholder would like the system to have a clear main menu.

I have now also partially met:

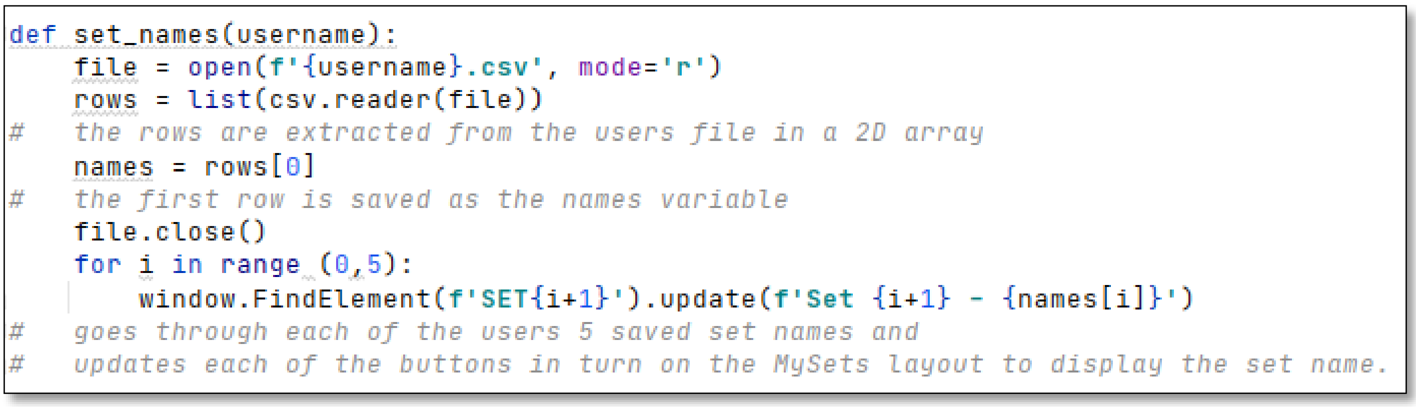
- SC3 - The stakeholder would like the system to allow them to create their own sets of flashcards.

However, to fully meet this I will need to allow the user to add flashcards to the sets they have created (which I will be doing next).

Changes made to original plan
I have decided that I am going to update the text on the buttons on the MySets layout when the button to take the user to this page is pressed (as I feel this will be more efficient) rather than every time a new set is made. I have also removed the need for the integer number of sets being stored in the users CSV file, thus saving space and making my program a bit more efficient.

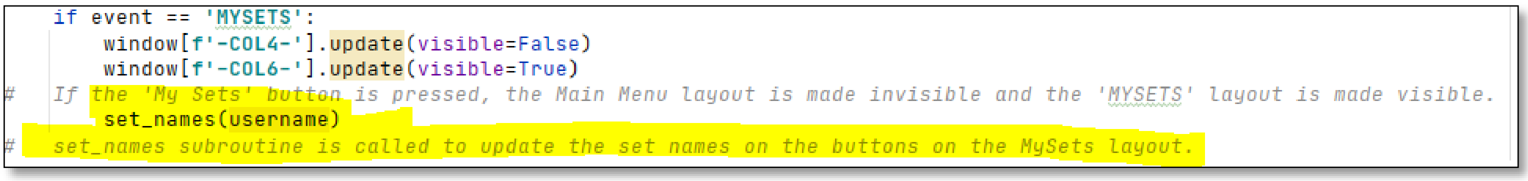
Adding flashcards to sets – Stage 1

Now that the user can create sets with a given name, I need to add the ability for them to actually add flashcards to those sets which can be revised. There will be an ability to add a flashcard to a set once that set has been clicked on from the MySets menu. However, first I need to update the text on the buttons on this display to show the set names so the user knows which set is which.
To do this, I have created a subroutine (as I initially planned to) called set_names that updates the buttons as this subroutine might be used again later on in my development. If it is not then I can just go back and put the code in the event loop instead of as a separate subroutine in order to improve efficiency.

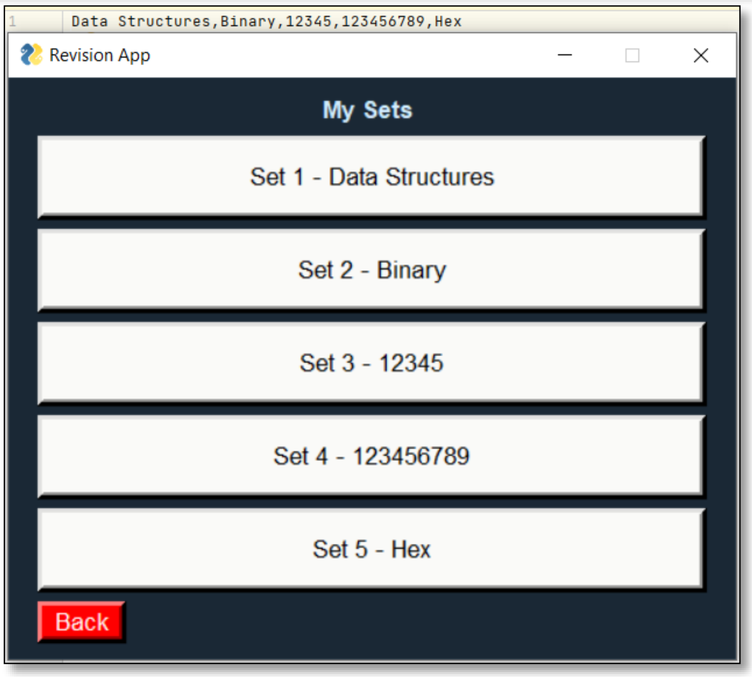


```
def set_names(username):
    file = open(f'{username}.csv', mode='r')
    rows = list(csv.reader(file))
#   the rows are extracted from the users file in a 2D array
    names = rows[0]
#   the first row is saved as the names variable
    file.close()
    for i in range (0,5):
        window.FindElement(f'SET{i+1}').update(f'Set {i+1} - {names[i]}')
#   goes through each of the users 5 saved set names and
#   updates each of the buttons in turn on the MySets layout to display the set name.
```

I have then called this subroutine when the MySets button is pressed, so that now each of the buttons successfully displays all set names in the correct order.

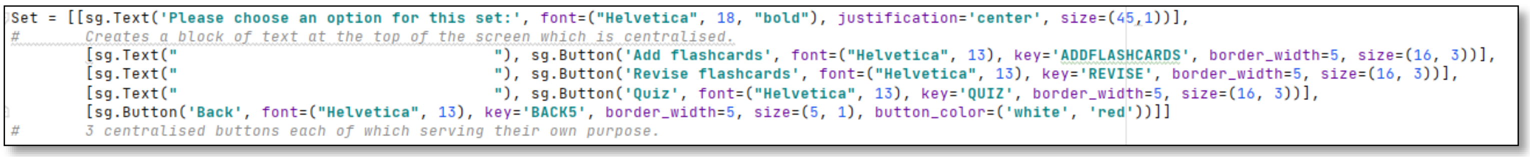


```
    if event == 'MYSETS':
        window[f'-COL4-'].update(visible=False)
        window[f'-COL6-'].update(visible=True)
#   If the 'My Sets' button is pressed, the Main Menu layout is made invisible and the 'MYSETS' layout is made visible.
        set_names(username)
#   set_names subroutine is called to update the set names on the buttons on the MySets layout.
```



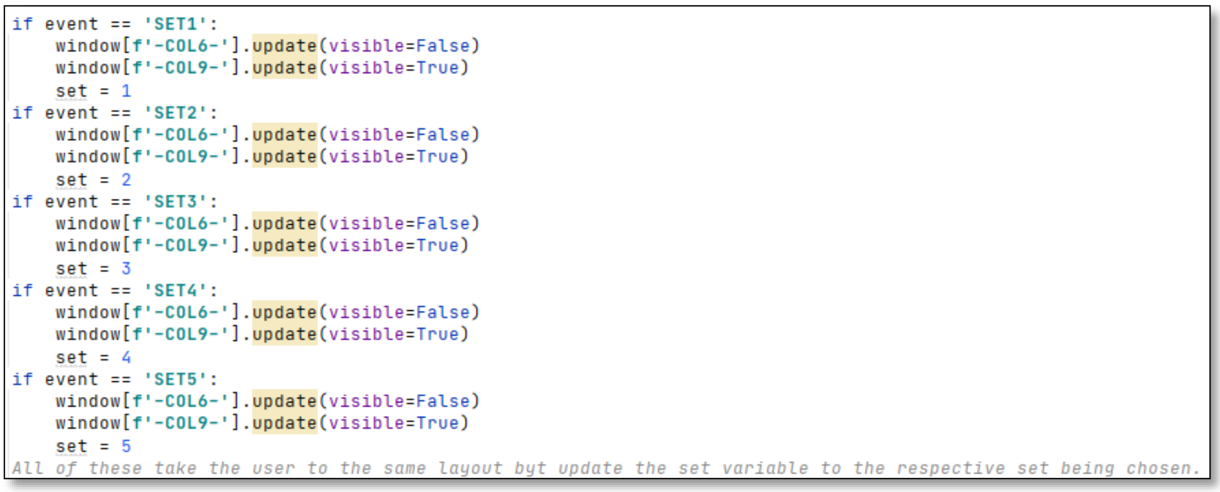
```
1    Data Structures,Binary,12345,123456789,Hex
```

I have then created the general layout for all sets where the user can choose what they would like to do with a set, which has also then been added to the column layout.



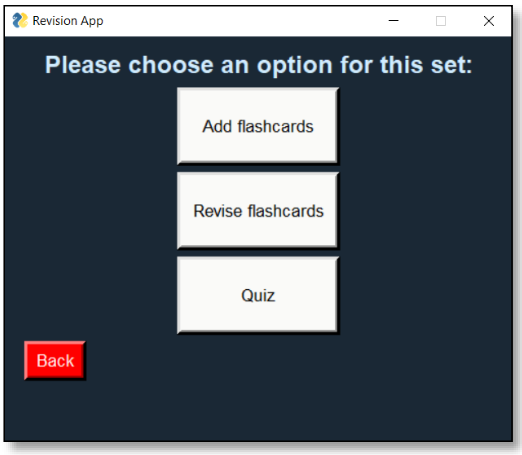
```
Set = [[sg.Text('Please choose an option for this set:', font=("Helvetica", 18, "bold"), justification='center', size=(45,1))],
#       Creates a block of text at the top of the screen which is centralised.
        [sg.Text("                                          "), sg.Button('Add flashcards', font=("Helvetica", 13), key='ADDFLASHCARDS', border_width=5, size=(16, 3))],
        [sg.Text("                                          "), sg.Button('Revise flashcards', font=("Helvetica", 13), key='REVISE', border_width=5, size=(16, 3))],
        [sg.Text("                                          "), sg.Button('Quiz', font=("Helvetica", 13), key='QUIZ', border_width=5, size=(16, 3))],
        [sg.Button('Back', font=("Helvetica", 13), key='BACK5', border_width=5, size=(5, 1), button_color=('white', 'red'))]]
#       3 centralised buttons each of which serving their own purpose.
```

I have then implemented the functionality of all of the set buttons (so that they take the user to a screen where they can choose what they wish to do with that set):

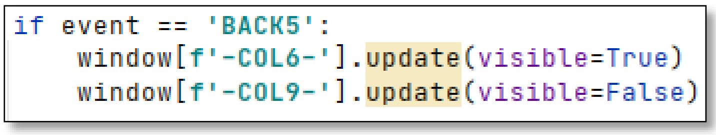


```
if event == 'SET1':
    window[f'-COL6-'].update(visible=False)
    window[f'-COL9-'].update(visible=True)
    set = 1
if event == 'SET2':
    window[f'-COL6-'].update(visible=False)
    window[f'-COL9-'].update(visible=True)
    set = 2
if event == 'SET3':
    window[f'-COL6-'].update(visible=False)
    window[f'-COL9-'].update(visible=True)
    set = 3
if event == 'SET4':
    window[f'-COL6-'].update(visible=False)
    window[f'-COL9-'].update(visible=True)
    set = 4
if event == 'SET5':
    window[f'-COL6-'].update(visible=False)
    window[f'-COL9-'].update(visible=True)
    set = 5
All of these take the user to the same layout byt update the set variable to the respective set being chosen.
```

The resultant layout looks like this, which is again in line with my stakeholders previous requests regarding layouts:

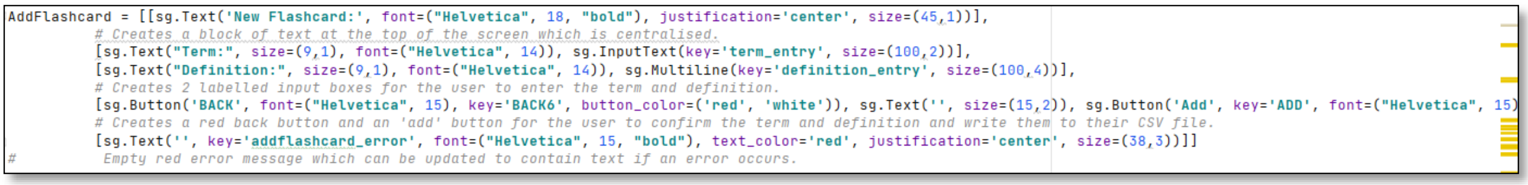


I have also then implemented the functionality of the back button (in the same way as previous back buttons).



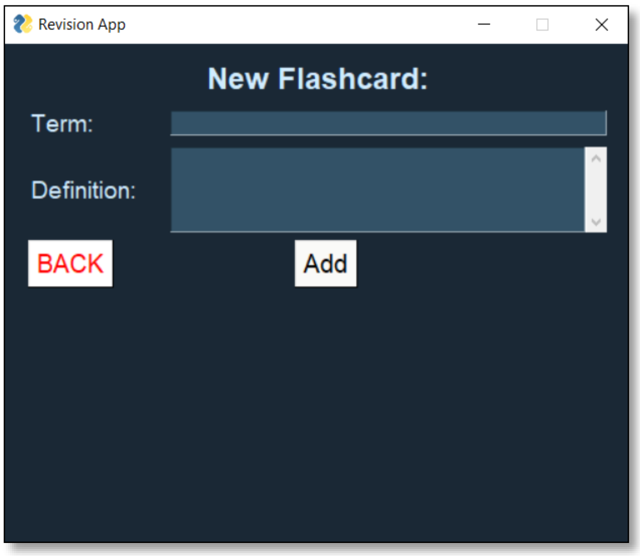
```
if event == 'BACK5':
    window[f'-COL6-'].update(visible=True)
    window[f'-COL9-'].update(visible=False)
```

I next need to create the add flashcard layout before I move onto stage 2 which is where I will implement the functionality of the add flashcards feature:



```
AddFlashcard = [[sg.Text('New Flashcard:', font=("Helvetica", 18, "bold"), justification='center', size=(45,1))],
          # Creates a block of text at the top of the screen which is centralised.
          [sg.Text("Term:", size=(9,1), font=("Helvetica", 14)), sg.InputText(key='term_entry', size=(100,2))],
          [sg.Text("Definition:", size=(9,1), font=("Helvetica", 14)), sg.Multiline(key='definition_entry', size=(100,4))],
          # Creates 2 labelled input boxes for the user to enter the term and definition.
          [sg.Button('BACK', font=("Helvetica", 15), key='BACK6', button_color=('red', 'white')), sg.Text('', size=(15,2)), sg.Button('Add', key='ADD', font=("Helvetica", 15)
          # Creates a red back button and an 'add' button for the user to confirm the term and definition and write them to their CSV file.
          [sg.Text('', key='addflashcard_error', font=("Helvetica", 15, "bold"), text_color='red', justification='center', size=(38,3))]]
#          Empty red error message which can be updated to contain text if an error occurs.
```

This layout now appears like this (and the error message will appear underneath in large bold red text when necessary):



I can now carry out the first iteration of testing:

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
6.1.1	Entering term and definition into textual input boxes	String 'Static' String 'Size cannot change during runtime' **VALID DATA**	App will allow user to enter these strings into the relevant input boxes	To check the user can enter characters into the two textual input boxes.	Allows the user to enter these details into the input boxes.
6.1.2	Selection of button to add flashcard to set	Mouse click	Nothing will happen as the functionality of the button has not been implemented yet.	To check the 'add' button appears correctly formatted on the display and can be interacted with by the user.	The values inputted remain on the screen and nothing is written to the file, as expected.

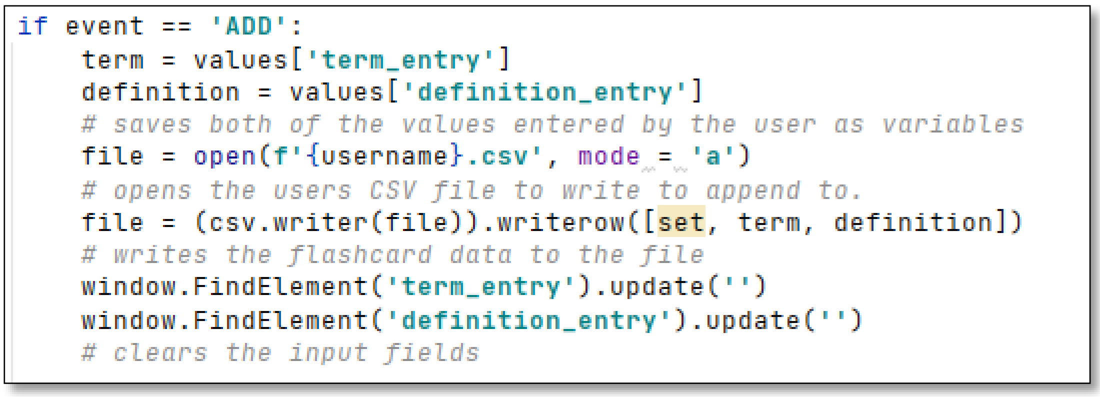
Adding flashcards to sets – Stage 2

Now that I have everything in place, I can begin implementing the functionality of this feature.

I have started by again implementing the functionality of the back button:

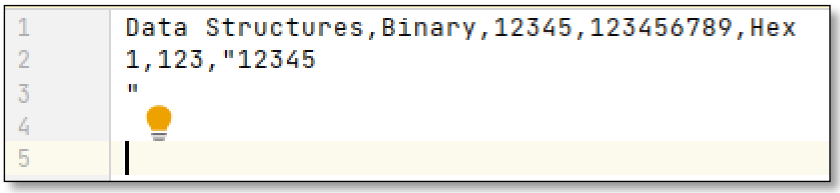
```
    if event == 'BACK6':
        window[f'-COL9-'].update(visible=True)
        window[f'-COL10-'].update(visible=False)
        window.FindElement('term_entry').update('')
        window.FindElement('definition_entry').update('')
#   Takes the user from the AddFlashcard layout to the Set layout, and clears the input fields.
```

I have then written a piece of code which saves the user inputs as variables, and writes a row to the CSV file containing the set number, and the term and definition, and then clears the entry fields:



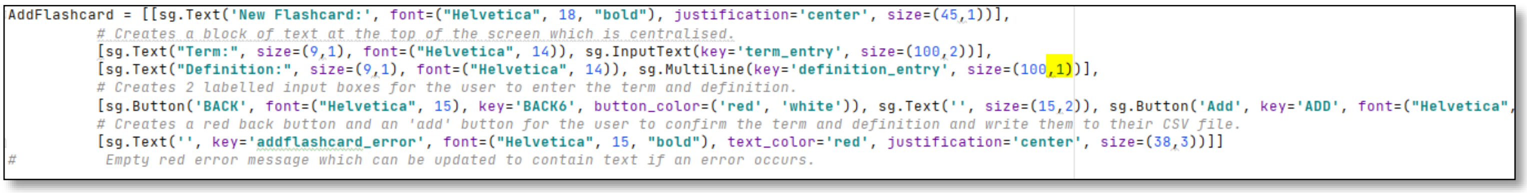
```python
if event == 'ADD':
    term = values['term_entry']
    definition = values['definition_entry']
    # saves both of the values entered by the user as variables
    file = open(f'{username}.csv', mode = 'a')
    # opens the users CSV file to write to append to.
    file = (csv.writer(file)).writerow([set, term, definition])
    # writes the flashcard data to the file
    window.FindElement('term_entry').update('')
    window.FindElement('definition_entry').update('')
    # clears the input fields
```

However, when I add flashcards to set 1, and enter ‘123’ into the term field and ‘12345’ into the definition field, this is the result in the CSV file:



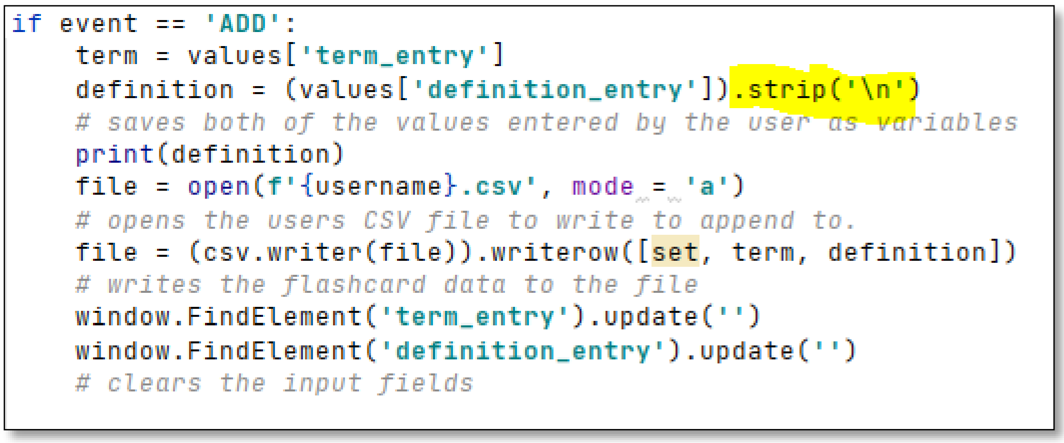
```
Data Structures,Binary,12345,123456789,Hex
1,123,"12345
"


```

The issue here is, the definition is being written with a line break, and then 2 extra empty lines after it, as well as quotation marks. I have tried changing the vertical size of the multiline element to 1 to check if this had an effect, which it did not:



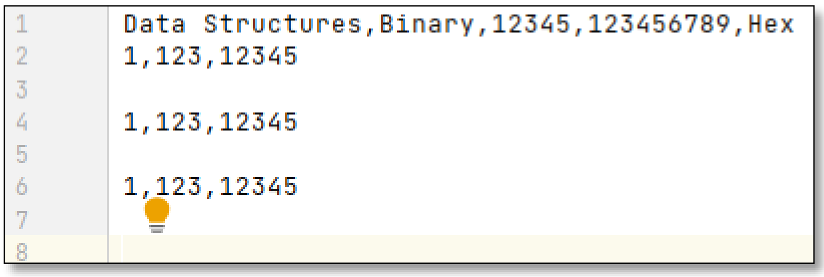
```python
AddFlashcard = [[sg.Text('New Flashcard:', font=("Helvetica", 18, "bold"), justification='center', size=(45,1))],
          # Creates a block of text at the top of the screen which is centralised.
          [sg.Text("Term:", size=(9,1), font=("Helvetica", 14)), sg.InputText(key='term_entry', size=(100,2))],
          [sg.Text("Definition:", size=(9,1), font=("Helvetica", 14)), sg.Multiline(key='definition_entry', size=(100,1))],
          # Creates 2 labelled input boxes for the user to enter the term and definition.
          [sg.Button('BACK', font=("Helvetica", 15), key='BACK6', button_color=('red', 'white')), sg.Text('', size=(15,2)), sg.Button('Add', key='ADD', font=("Helvetica",
          # Creates a red back button and an 'add' button for the user to confirm the term and definition and write them to their CSV file.
          [sg.Text('', key='addflashcard_error', font=("Helvetica", 15, "bold"), text_color='red', justification='center', size=(38,3))]]
#          Empty red error message which can be updated to contain text if an error occurs.
```

It still writes extra empty lines. I have next tried using a technique I have learnt previously in this report – the .strip() function:



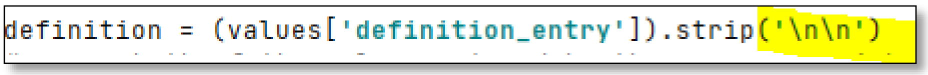
```python
if event == 'ADD':
    term = values['term_entry']
    definition = (values['definition_entry']).strip('\n')
    # saves both of the values entered by the user as variables
    print(definition)
    file = open(f'{username}.csv', mode = 'a')
    # opens the users CSV file to write to append to.
    file = (csv.writer(file)).writerow([set, term, definition])
    # writes the flashcard data to the file
    window.FindElement('term_entry').update('')
    window.FindElement('definition_entry').update('')
    # clears the input fields
```

This has removed one of the empty lines and the quotation marks, but I still have another empty line being written to the file:



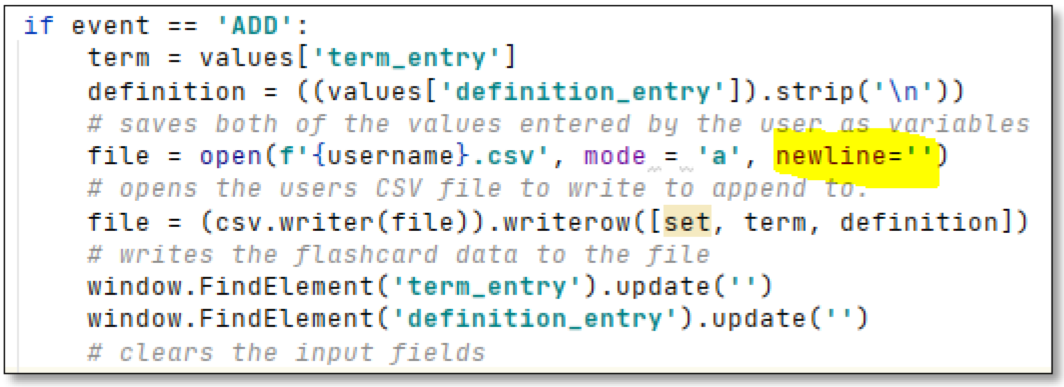
```
Data Structures,Binary,12345,123456789,Hex
1,123,12345

1,123,12345

1,123,12345

```

I tried putting another line break within the strip function:



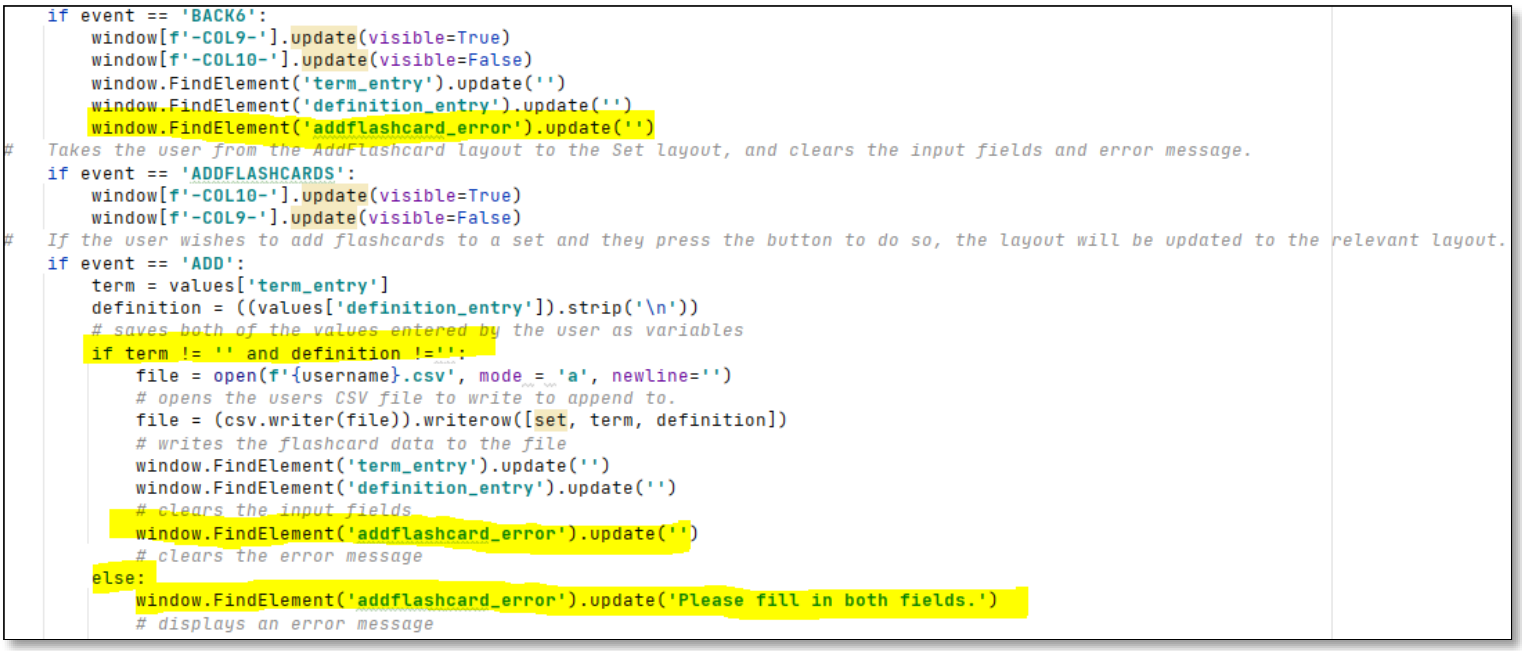
```
definition = (values['definition_entry']).strip('\n\n')
```

However, this made no change. I then added the newline parameter to the file opening and it worked – there were no longer any blank lines left in between rows:



```
if event == 'ADD':
    term = values['term_entry']
    definition = ((values['definition_entry']).strip('\n'))
    # saves both of the values entered by the user as variables
    file = open(f'{username}.csv', mode = 'a', newline='')
    # opens the users CSV file to write to append to.
    file = (csv.writer(file)).writerow([set, term, definition])
    # writes the flashcard data to the file
    window.FindElement('term_entry').update('')
    window.FindElement('definition_entry').update('')
    # clears the input fields
```

Now that problem is solved, I can add some validation to the user inputs (which on this occasion will basically just consist of checking whether the user has actually inputted something, and updating the error message if not).



```
    if event == 'BACK6':
        window[f'-COL9-'].update(visible=True)
        window[f'-COL10-'].update(visible=False)
        window.FindElement('term_entry').update('')
        window.FindElement('definition_entry').update('')
        window.FindElement('addflashcard_error').update('')
#   Takes the user from the AddFlashcard layout to the Set layout, and clears the input fields and error message.
    if event == 'ADDFLASHCARDS':
        window[f'-COL10-'].update(visible=True)
        window[f'-COL9-'].update(visible=False)
#   If the user wishes to add flashcards to a set and they press the button to do so, the layout will be updated to the relevant layout.
    if event == 'ADD':
        term = values['term_entry']
        definition = ((values['definition_entry']).strip('\n'))
        # saves both of the values entered by the user as variables
        if term != '' and definition !='':
            file = open(f'{username}.csv', mode = 'a', newline='')
            # opens the users CSV file to write to append to.
            file = (csv.writer(file)).writerow([set, term, definition])
            # writes the flashcard data to the file
            window.FindElement('term_entry').update('')
            window.FindElement('definition_entry').update('')
            # clears the input fields
            window.FindElement('addflashcard_error').update('')
            # clears the error message
        else:
            window.FindElement('addflashcard_error').update('Please fill in both fields.')
            # displays an error message
```

The program now check whether the user has inputted something in both fields, and if they haven't, an error message is displayed. These error messages are cleared if the user backs off the page/enters details correctly. No other validation is required for this as the user is allowed to enter whatever they wish as their term and definition, as long as it is not left blank.

I am now ready to carry out the 2nd iteration of my testing on this feature:

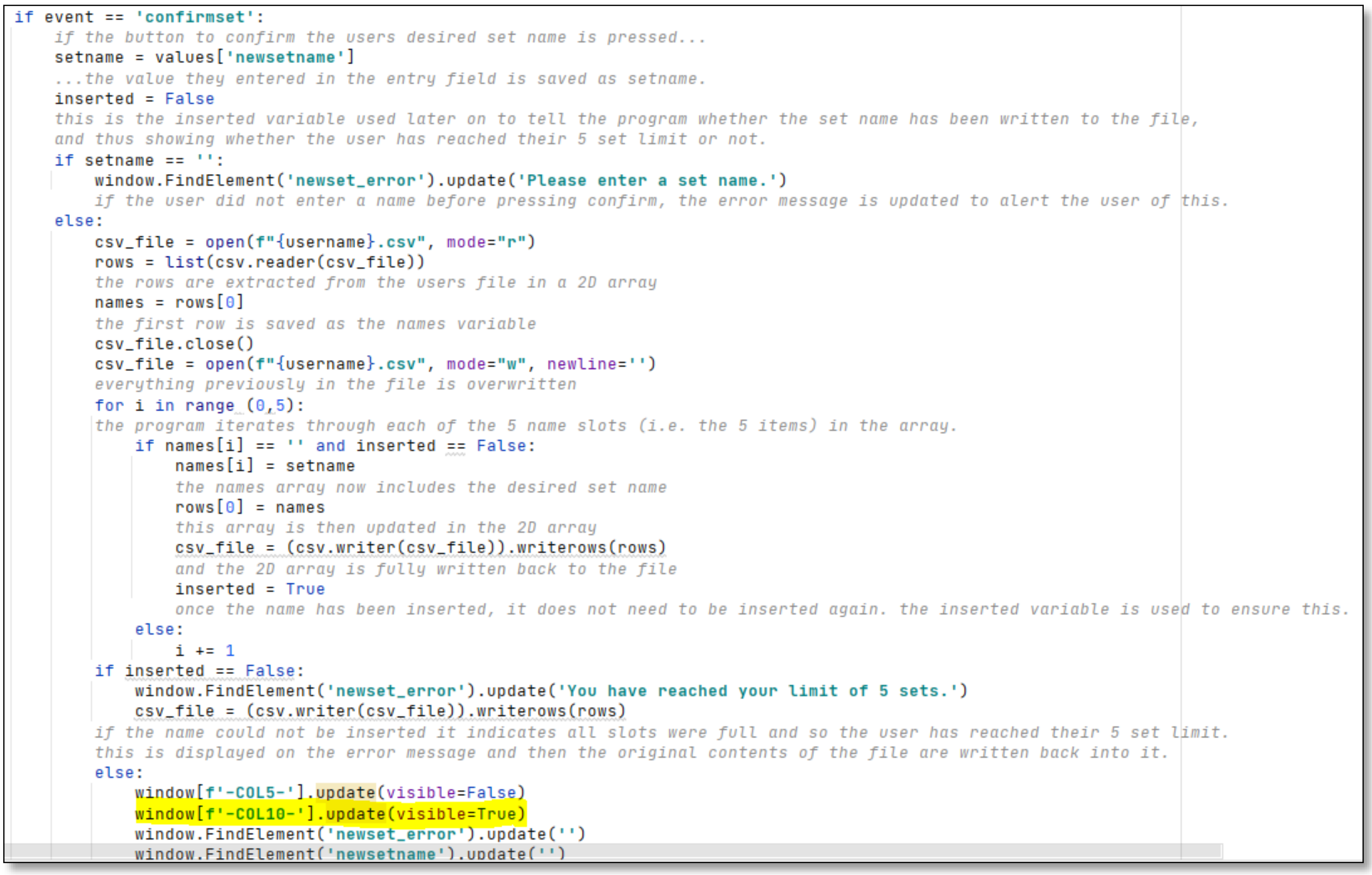
Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
6.2.1	Entering term and definition into textual input boxes	String 'Static' String 'Size cannot change during runtime' **VALID DATA**	App will allow user to enter these strings into the relevant input boxes	To check the user can enter characters into the two textual input boxes.	Allows the user to enter these details into the input boxes.
6.2.2	Selection of button to add flashcard to set	Mouse click	Term and definition will be written to the relative positions in the users personal .csv file, and the term and definition textual input boxes will be cleared.	To check the 'add' button writes the term and definition to the correct positions in the users .csv file when pressed.	Term and definition, along with set number, are successfully written to the file and the input boxes cleared. Also, when an input box is left blank, the error message correctly displays and then clears when the issue is resolved.

Adding flashcards to a set – Review

Stakeholder feedback

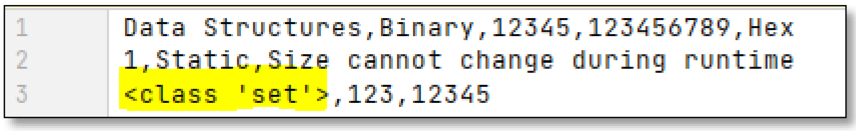
I have contacted one of my stakeholders and sent them this prototype of my solution for them to test. They had no issues with this feature and said that the information they entered was written to their CSV file successfully. However, he did recommend that straight after creating a set you are taken to the add flashcard screen so that you can add flashcards to a set straight after creating it. I agree this would be a good idea to reduce confusion and make my solution quicker and easier to use for students who want to quickly make a set, add flashcards, and revise them.

To do this, I simply need to update the layout once a set has been successfully created:



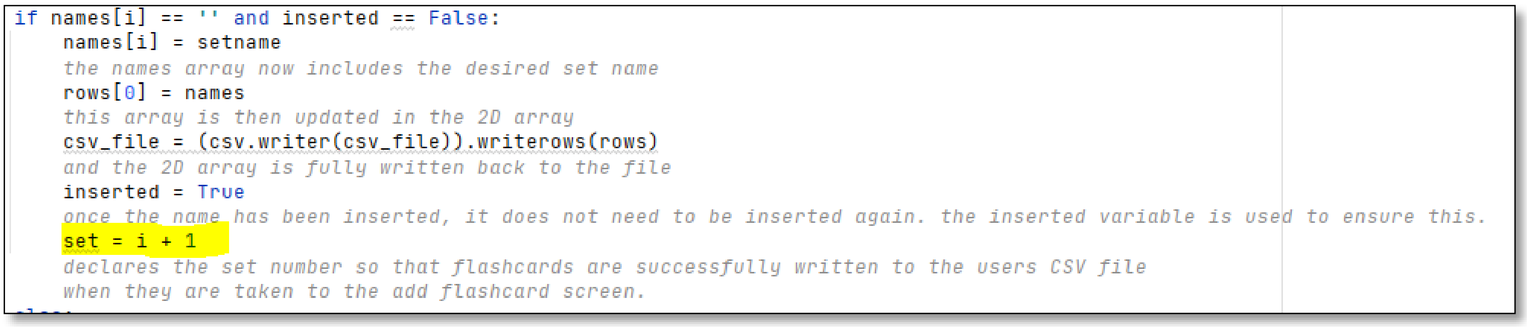
```
if event == 'confirmset':
    if the button to confirm the users desired set name is pressed...
    setname = values['newsetname']
    ...the value they entered in the entry field is saved as setname.
    inserted = False
    this is the inserted variable used later on to tell the program whether the set name has been written to the file,
    and thus showing whether the user has reached their 5 set limit or not.
    if setname == '':
        window.FindElement('newset_error').update('Please enter a set name.')
        if the user did not enter a name before pressing confirm, the error message is updated to alert the user of this.
    else:
        csv_file = open(f"{username}.csv", mode="r")
        rows = list(csv.reader(csv_file))
        the rows are extracted from the users file in a 2D array
        names = rows[0]
        the first row is saved as the names variable
        csv_file.close()
        csv_file = open(f"{username}.csv", mode="w", newline='')
        everything previously in the file is overwritten
        for i in range (0,5):
        the program iterates through each of the 5 name slots (i.e. the 5 items) in the array.
            if names[i] == '' and inserted == False:
                names[i] = setname
                the names array now includes the desired set name
                rows[0] = names
                this array is then updated in the 2D array
                csv_file = (csv.writer(csv_file)).writerows(rows)
                and the 2D array is fully written back to the file
                inserted = True
                once the name has been inserted, it does not need to be inserted again. the inserted variable is used to ensure this.
            else:
                i += 1
        if inserted == False:
            window.FindElement('newset_error').update('You have reached your limit of 5 sets.')
            csv_file = (csv.writer(csv_file)).writerows(rows)
        if the name could not be inserted it indicates all slots were full and so the user has reached their 5 set limit.
        this is displayed on the error message and then the original contents of the file are written back into it.
        else:
            window[f'-COL5-'].update(visible=False)
            window[f'-COL10-'].update(visible=True)
            window.FindElement('newset_error').update('')
            window.FindElement('newsetname').update('')
```

However, when I then add a flashcard, the set number is not correctly written to the users CSV file:



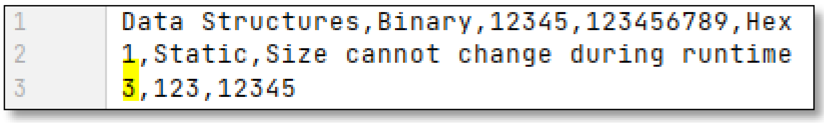
```
Data Structures,Binary,12345,123456789,Hex
1,Static,Size cannot change during runtime
<class 'set'>,123,12345
```

I realised this is because it has not been declared yet as the user has not clicked on one of the set buttons to get to the add flashcard screen. To fix this, I ensured I declared the set variable in the loop as the index position of the insertion + 1:



```
if names[i] == '' and inserted == False:
    names[i] = setname
    the names array now includes the desired set name
    rows[0] = names
    this array is then updated in the 2D array
    csv_file = (csv.writer(csv_file)).writerows(rows)
    and the 2D array is fully written back to the file
    inserted = True
    once the name has been inserted, it does not need to be inserted again. the inserted variable is used to ensure this.
    set = i + 1
    declares the set number so that flashcards are successfully written to the users CSV file
    when they are taken to the add flashcard screen.
```

And the correct set number is now successfully written to the CSV file:



```
Data Structures,Binary,12345,123456789,Hex
1,Static,Size cannot change during runtime
3,123,12345
```

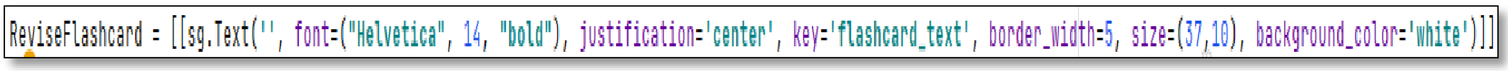
Success Criteria
I have now successfully added the ability for the user to add flashcards to the sets they have created, as well as received and implemented more feedback from my stakeholders on this feature. So far I have met:

- SC10 - The stakeholder would like the system to be user-friendly and easy to understand.
- SC1 - The stakeholder would like the system to allow them to create an account if they haven't already got one.
- SC2 - The stakeholder would like the system to allow them to log in to their account if they have already created one.
- SC9 - The stakeholder would like the system to have a clear main menu.
- SC3 - The stakeholder would like the system to allow them to create their own sets of flashcards.
- SC4 - The stakeholder would like the system to view sets of flashcards they have already created.

Changes made to my original plan
For this feature, I did not really make any changes to my original plan in terms of the interface or the functionality of the feature.

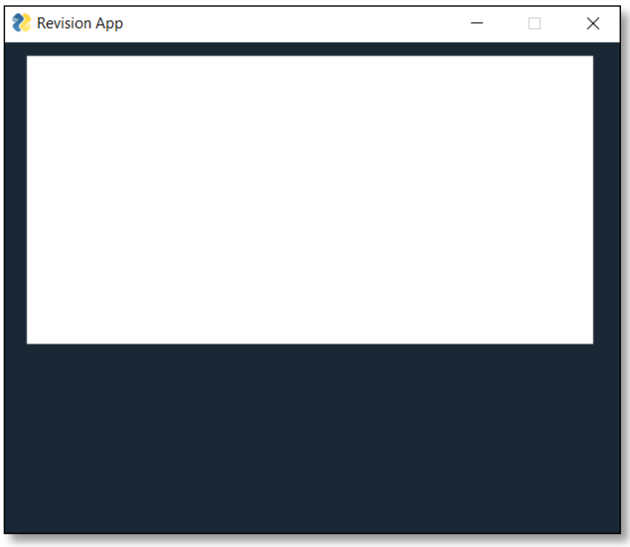
Revising flashcards – Stage 1

I first need to set up the layout for this feature, which will be one of the hardest to set up due to the complexity of its design. I am planning to have a white square in the centre of the screen which will contain the terms and definitions (meant to mimic a white paper flashcard, helping to give the user an authentic experience), and then buttons underneath for next/previous flashcard and 'flip' flashcard. I will also be adding a delete flashcard feature which will also appear as a button here.
I have started by creating the white background text box where the term/definition will be inserted:

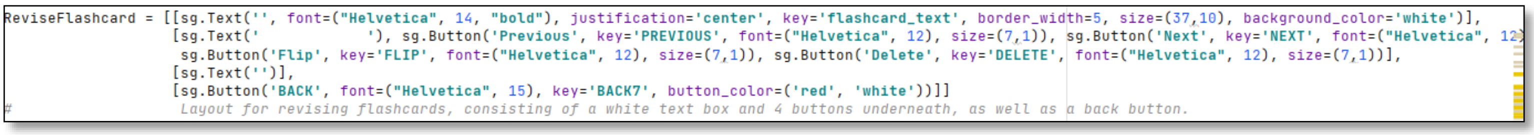


```
ReviseFlashcard = [[sg.Text('', font=("Helvetica", 14, "bold"), justification='center', key='flashcard_text', border_width=5, size=(37,10), background_color='white')]]
```

This appears like this on the screen:

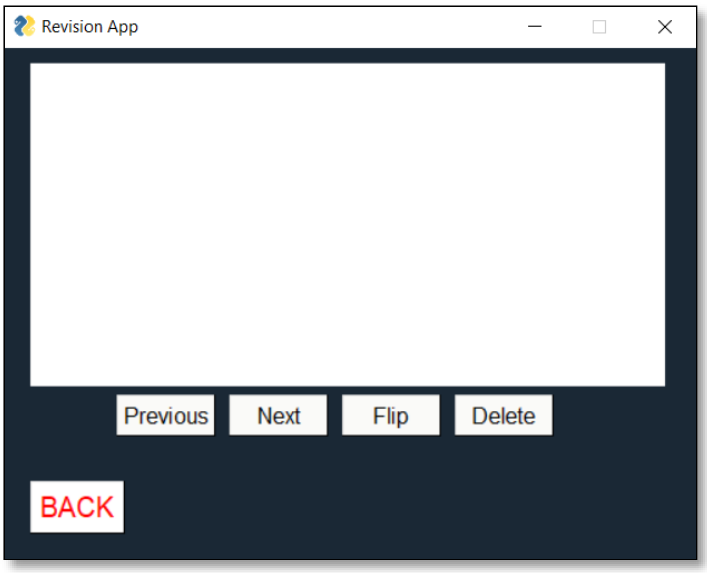


I have then created the buttons as well as a back button underneath:



```
ReviseFlashcard = [[sg.Text('', font=("Helvetica", 14, "bold"), justification='center', key='flashcard_text', border_width=5, size=(37,10), background_color='white')],
                   [sg.Text('             '), sg.Button('Previous', key='PREVIOUS', font=("Helvetica", 12), size=(7,1)), sg.Button('Next', key='NEXT', font=("Helvetica", 12
                    sg.Button('Flip', key='FLIP', font=("Helvetica", 12), size=(7,1)), sg.Button('Delete', key='DELETE', font=("Helvetica", 12), size=(7,1))],
                   [sg.Text('')],
                   [sg.Button('BACK', font=("Helvetica", 15), key='BACK7', button_color=('red', 'white'))]]
#                   Layout for revising flashcards, consisting of a white text box and 4 buttons underneath, as well as a back button.
```

The layout now appears like this:



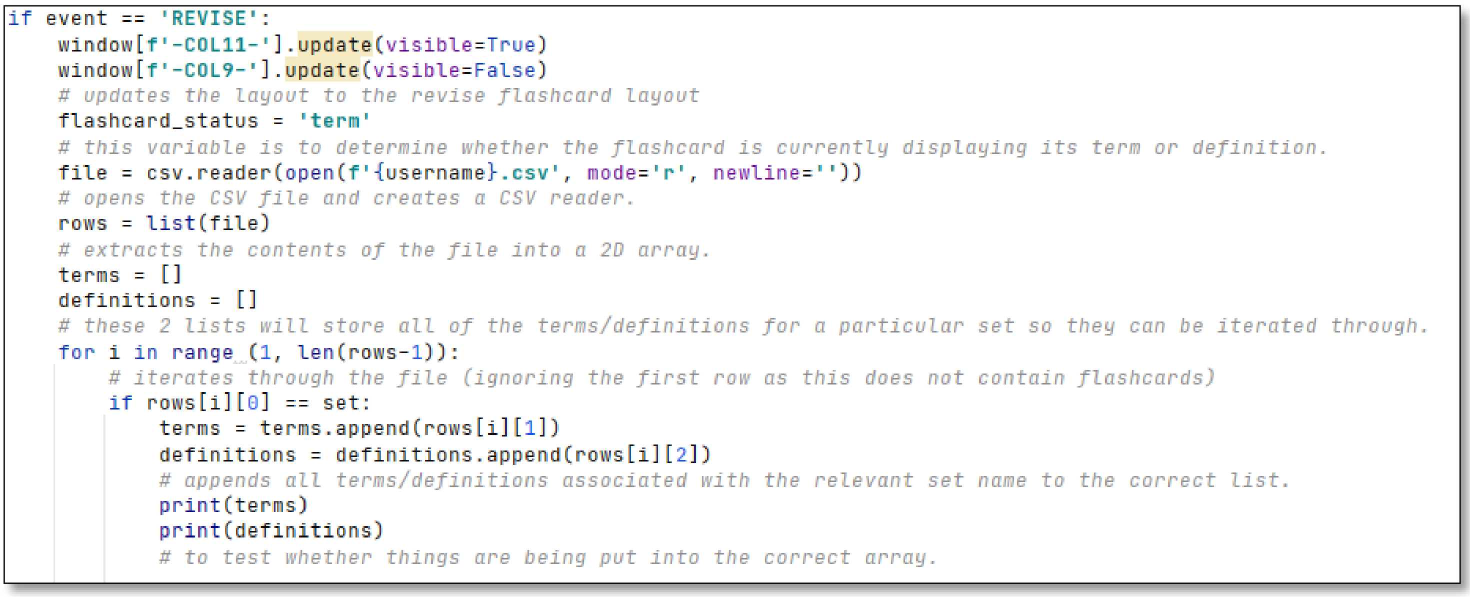
After testing this prototype, I have filled in the first iteration of testing:

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
7.1.1	Flip flashcard via button press	Mouse click	Nothing will happen as the functionality of the button has not been implemented yet.	To check whether the button appears in the correct position on screen and can be interacted with by the user.	Button is pressable and present in the correct position, but nothing happens when it is pressed.
7.1.2	Viewing the next flashcard via a button press	Mouse click	Nothing will happen as the functionality of the button has not been implemented yet.	To check whether the button appears in the correct position on screen and can be interacted with by the user.	Button is pressable and present in the correct position, but nothing happens when it is pressed.
7.1.3	Viewing the previous flashcard via a button press	Mouse click	Nothing will happen as the functionality of the button has not been implemented yet.	To check whether the button appears in the correct position on screen and can be interacted with by the user.	Button is pressable and present in the correct position, but nothing happens when it is pressed.
7.1.4	Deleting flashcard via a button press	Mouse click	Nothing will happen as the functionality of the button has not been implemented yet.	To check whether the button appears in the correct position on screen and can be interacted with by the user.	Button is pressable and present in the correct position, but nothing happens when it is pressed.

Revising flashcards – Stage 2

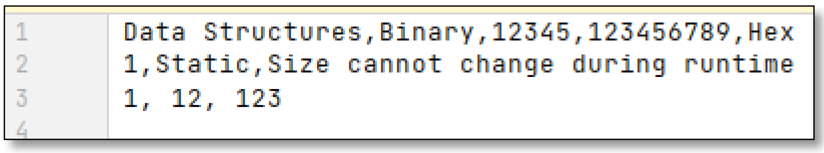
I can now begin to implement the functionality of this feature.

I have started by reading the term and definition from the users CSV file and displaying the term on the screen. The flashcard_status variable determines whether the flashcard is currently showing its term or definition (so that the program knows what to flip to when the flip button is pressed).



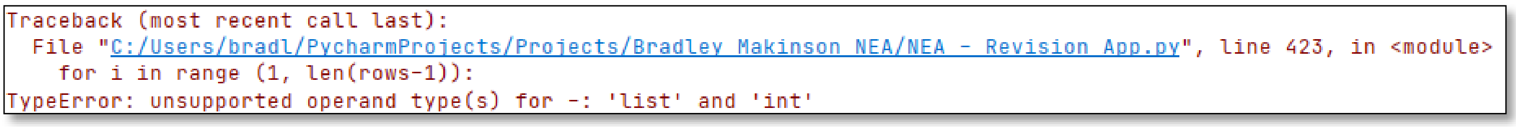
```python
if event == 'REVISE':
    window[f'-COL11-'].update(visible=True)
    window[f'-COL9-'].update(visible=False)
    # updates the layout to the revise flashcard layout
    flashcard_status = 'term'
    # this variable is to determine whether the flashcard is currently displaying its term or definition.
    file = csv.reader(open(f'{username}.csv', mode='r', newline=''))
    # opens the CSV file and creates a CSV reader.
    rows = list(file)
    # extracts the contents of the file into a 2D array.
    terms = []
    definitions = []
    # these 2 lists will store all of the terms/definitions for a particular set so they can be iterated through.
    for i in range (1, len(rows-1)):
        # iterates through the file (ignoring the first row as this does not contain flashcards)
        if rows[i][0] == set:
            terms = terms.append(rows[i][1])
            definitions = definitions.append(rows[i][2])
            # appends all terms/definitions associated with the relevant set name to the correct list.
            print(terms)
            print(definitions)
            # to test whether things are being put into the correct array.
```

However, when I ran this piece of code and went to revise set 1 with the following contents in the CSV file,



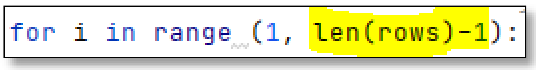
```
Data Structures,Binary,12345,123456789,Hex
1,Static,Size cannot change during runtime
1, 12, 123
```

it came back with the following error:



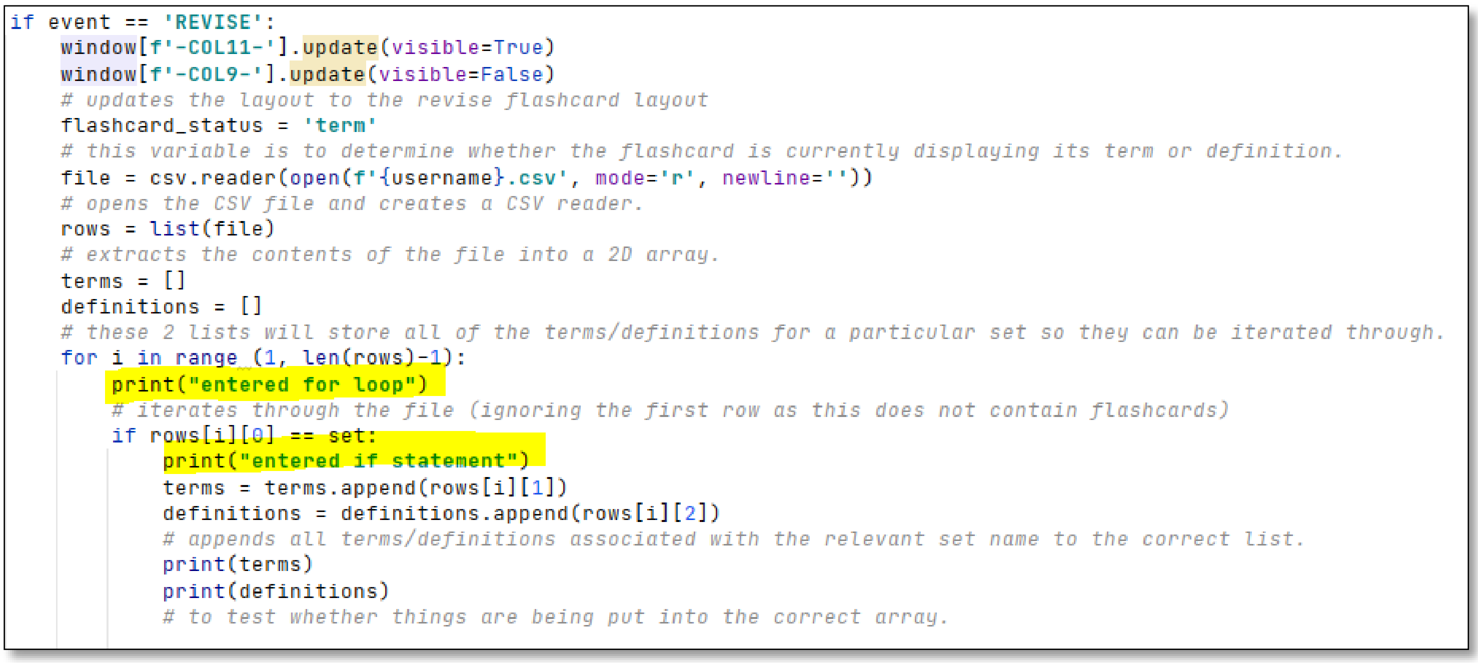
```
Traceback (most recent call last):
  File "C:/Users/bradl/PycharmProjects/Projects/Bradley Makinson NEA/NEA - Revision App.py", line 423, in <module>
    for i in range (1, len(rows-1)):
TypeError: unsupported operand type(s) for -: 'list' and 'int'
```

I realised this was a small error as I was trying to subtract an integer from an array, so I fixed this by moving the -1 outside of the len() function:



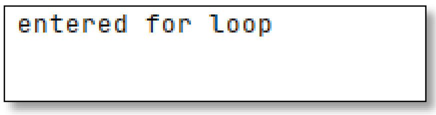
```python
for i in range (1, len(rows)-1):
```

Then when I did run the program again, I got no error but nothing was being printed implying either the program was not entering the for loop or, more likely, not entering the if statement. To investigate this, I inserted a few print statements so I could track the program:



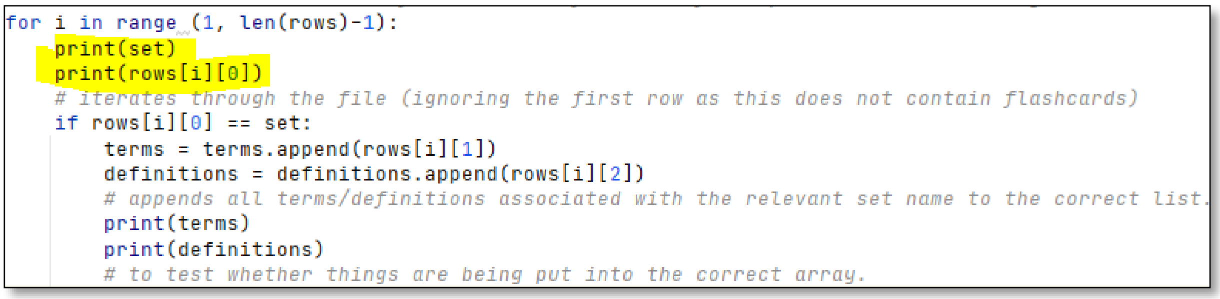
```
if event == 'REVISE':
    window[f'-COL11-'].update(visible=True)
    window[f'-COL9-'].update(visible=False)
    # updates the layout to the revise flashcard layout
    flashcard_status = 'term'
    # this variable is to determine whether the flashcard is currently displaying its term or definition.
    file = csv.reader(open(f'{username}.csv', mode='r', newline=''))
    # opens the CSV file and creates a CSV reader.
    rows = list(file)
    # extracts the contents of the file into a 2D array.
    terms = []
    definitions = []
    # these 2 lists will store all of the terms/definitions for a particular set so they can be iterated through.
    for i in range (1, len(rows)-1):
        print("entered for loop")
        # iterates through the file (ignoring the first row as this does not contain flashcards)
        if rows[i][0] == set:
            print("entered if statement")
            terms = terms.append(rows[i][1])
            definitions = definitions.append(rows[i][2])
            # appends all terms/definitions associated with the relevant set name to the correct list.
            print(terms)
            print(definitions)
            # to test whether things are being put into the correct array.
```

The resultant output was the following:



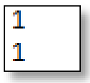
```
entered for loop
```

This shows the program is not entering the if statement. This must be because there is an issue with the parameter I have set, so I changed around the print statements again to investigate this:



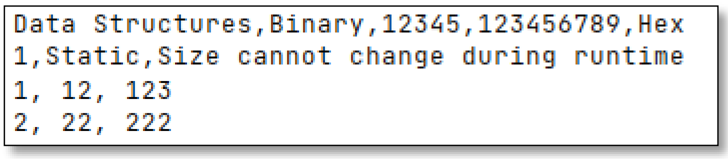
```
for i in range (1, len(rows)-1):
    print(set)
    print(rows[i][0])
    # iterates through the file (ignoring the first row as this does not contain flashcards)
    if rows[i][0] == set:
        terms = terms.append(rows[i][1])
        definitions = definitions.append(rows[i][2])
        # appends all terms/definitions associated with the relevant set name to the correct list.
        print(terms)
        print(definitions)
        # to test whether things are being put into the correct array.
```

When I ran the program again, I received the following output:



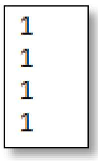
```
1
1
```

This shows that set and rows[i][0] are equal, so it should enter the if statement.
I then added another 'flashcard' to the CSV file that was not in set 1:



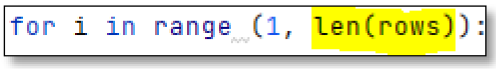
```
Data Structures,Binary,12345,123456789,Hex
1,Static,Size cannot change during runtime
1, 12, 123
2, 22, 222
```

The output I received now was:



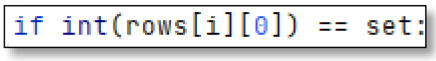
```
1
1
1
1
```

This implies that the range in my for loop is incorrect as it is not picking up the final flashcard in the file, so I removed the -1:



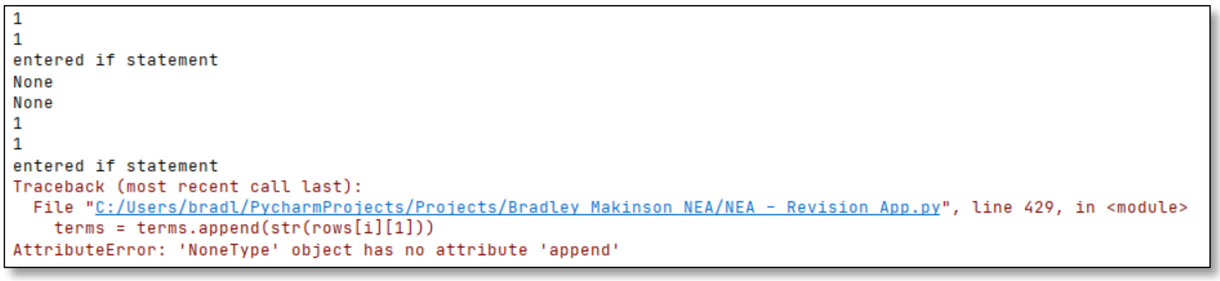
```
for i in range (1, len(rows)):
```

I then realised the value extracted from the 2D array was not an integer, and so need to be made one so the comparison could work as intended:



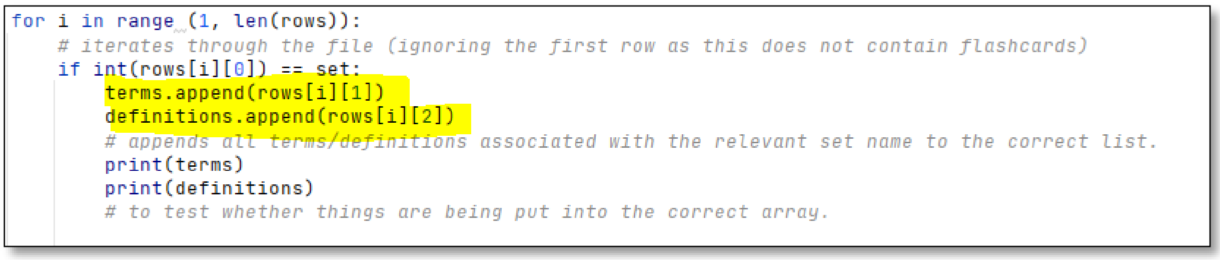
```
if int(rows[i][0]) == set:
```

Now when I run the code I get the following outputs ending in an error:



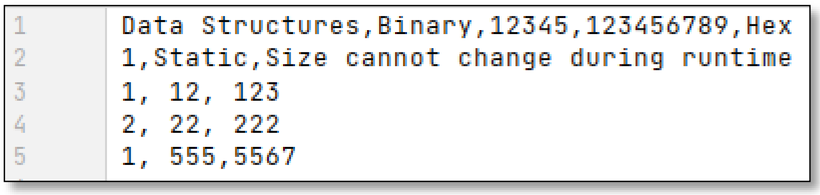
```
1
1
entered if statement
None
None
1
1
entered if statement
Traceback (most recent call last):
  File "C:/Users/bradl/PycharmProjects/Projects/Bradley Makinson NEA/NEA - Revision App.py", line 429, in <module>
    terms = terms.append(str(rows[i][1]))
AttributeError: 'NoneType' object has no attribute 'append'
```

I quickly realised this was because I, for some reason, tried to save the new lists as variables… I changed this so that it just appended the values to the relevant lists:



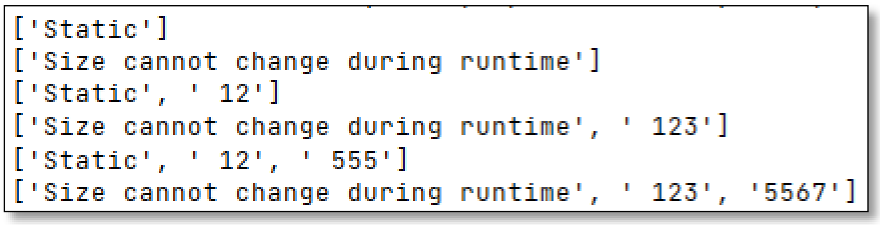
```
for i in range (1, len(rows)):
    # iterates through the file (ignoring the first row as this does not contain flashcards)
    if int(rows[i][0]) == set:
        terms.append(rows[i][1])
        definitions.append(rows[i][2])
        # appends all terms/definitions associated with the relevant set name to the correct list.
        print(terms)
        print(definitions)
        # to test whether things are being put into the correct array.
```

Now when I ran the program with the following contents of the CSV file,



```
Data Structures,Binary,12345,123456789,Hex
1,Static,Size cannot change during runtime
1, 12, 123
2, 22, 222
1, 555,5567
```

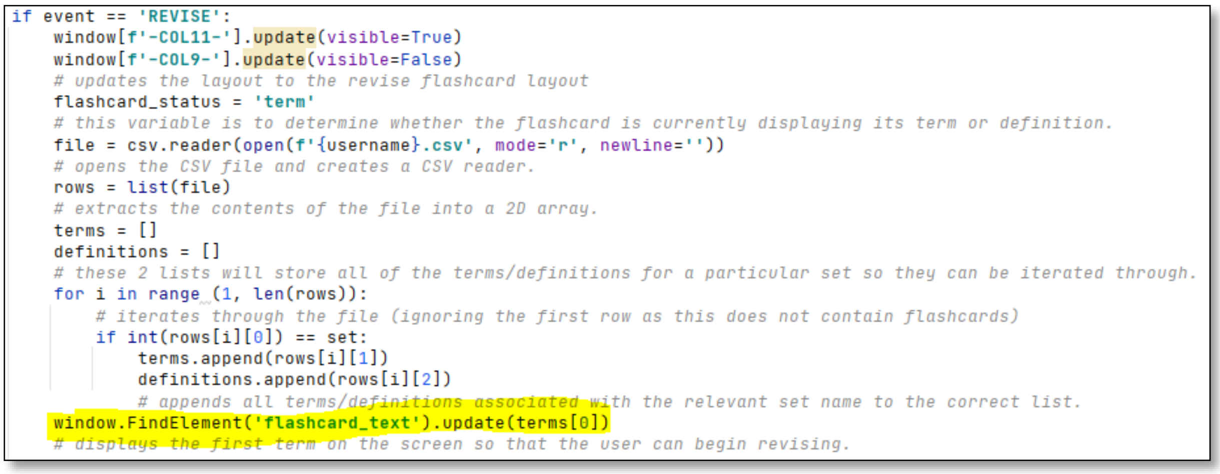
I get the following output:



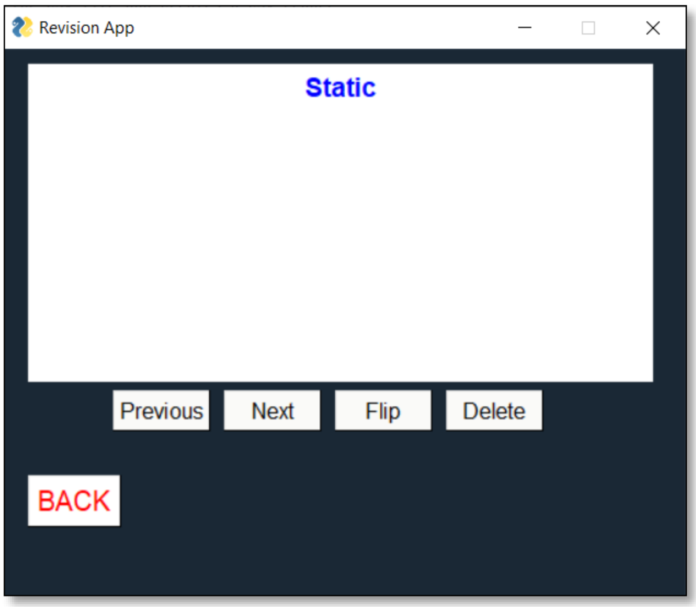
```
['Static']
['Size cannot change during runtime']
['Static', ' 12']
['Size cannot change during runtime', ' 123']
['Static', ' 12', ' 555']
['Size cannot change during runtime', ' 123', '5567']
```

This clearly shows the iterative process with the correct values from the CSV file being added to the correct list.

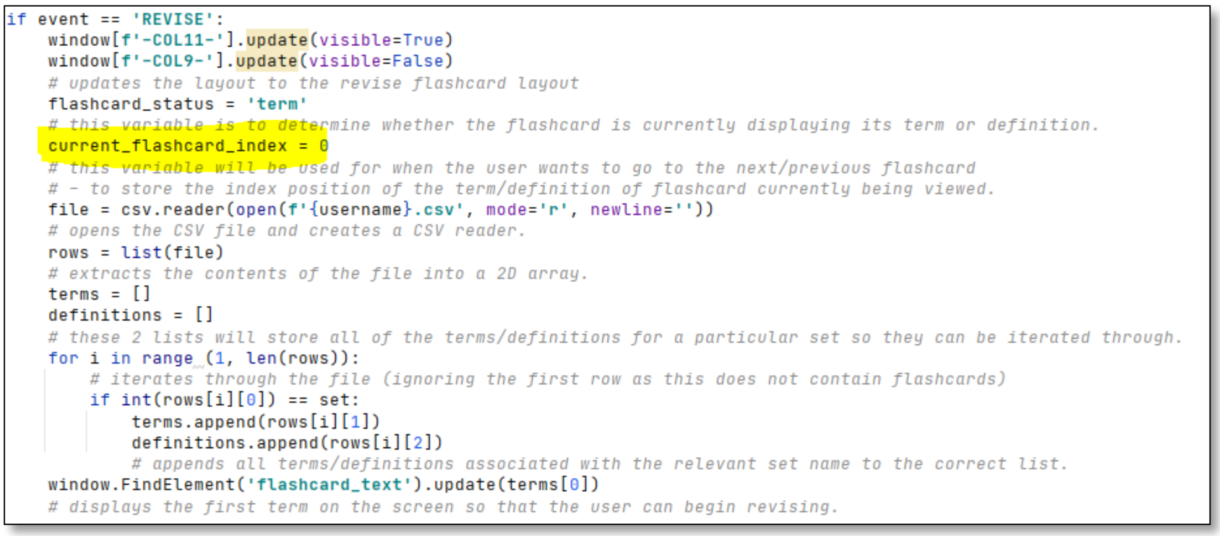
Now that I have successfully implemented a way to extract the relevant data from the file, I just need to work with the 2 lists (terms/definitions) to show the correct terms and definitions on the screen. I have started by initially displaying the first term from the list on the screen so the user can begin revising:



```python
if event == 'REVISE':
    window[f'-COL11-'].update(visible=True)
    window[f'-COL9-'].update(visible=False)
    # updates the layout to the revise flashcard layout
    flashcard_status = 'term'
    # this variable is to determine whether the flashcard is currently displaying its term or definition.
    file = csv.reader(open(f'{username}.csv', mode='r', newline=''))
    # opens the CSV file and creates a CSV reader.
    rows = list(file)
    # extracts the contents of the file into a 2D array.
    terms = []
    definitions = []
    # these 2 lists will store all of the terms/definitions for a particular set so they can be iterated through.
    for i in range (1, len(rows)):
        # iterates through the file (ignoring the first row as this does not contain flashcards)
        if int(rows[i][0]) == set:
            terms.append(rows[i][1])
            definitions.append(rows[i][2])
        # appends all terms/definitions associated with the relevant set name to the correct list.
    window.FindElement('flashcard_text').update(terms[0])
    # displays the first term on the screen so that the user can begin revising.
```

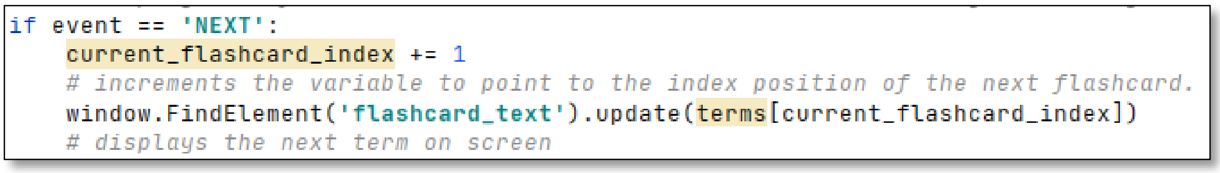


I now need to add the functionality of the previous, next and flip buttons. I will start with the next button.
To do this, I need a variable that will store the current index position of the term/definition being viewed in both lists (so that if a user wants to view a previous flashcard it can be decremented, and vice versa). This variable will then also be reset every time a user goes to revise a new set (as they will need to press the revise button).



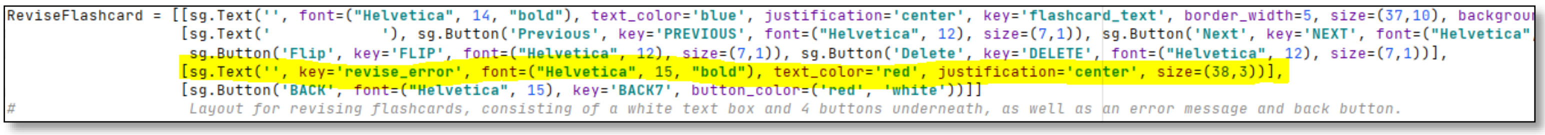
```
if event == 'REVISE':
    window[f'-COL11-'].update(visible=True)
    window[f'-COL9-'].update(visible=False)
    # updates the layout to the revise flashcard layout
    flashcard_status = 'term'
    # this variable is to determine whether the flashcard is currently displaying its term or definition.
    current_flashcard_index = 0
    # this variable will be used for when the user wants to go to the next/previous flashcard
    # - to store the index position of the term/definition of flashcard currently being viewed.
    file = csv.reader(open(f'{username}.csv', mode='r', newline=''))
    # opens the CSV file and creates a CSV reader.
    rows = list(file)
    # extracts the contents of the file into a 2D array.
    terms = []
    definitions = []
    # these 2 lists will store all of the terms/definitions for a particular set so they can be iterated through.
    for i in range (1, len(rows)):
        # iterates through the file (ignoring the first row as this does not contain flashcards)
        if int(rows[i][0]) == set:
            terms.append(rows[i][1])
            definitions.append(rows[i][2])
            # appends all terms/definitions associated with the relevant set name to the correct list.
    window.FindElement('flashcard_text').update(terms[0])
    # displays the first term on the screen so that the user can begin revising.
```

I now need to add the code to detect when the next button is pressed, and what to do when it is pressed:

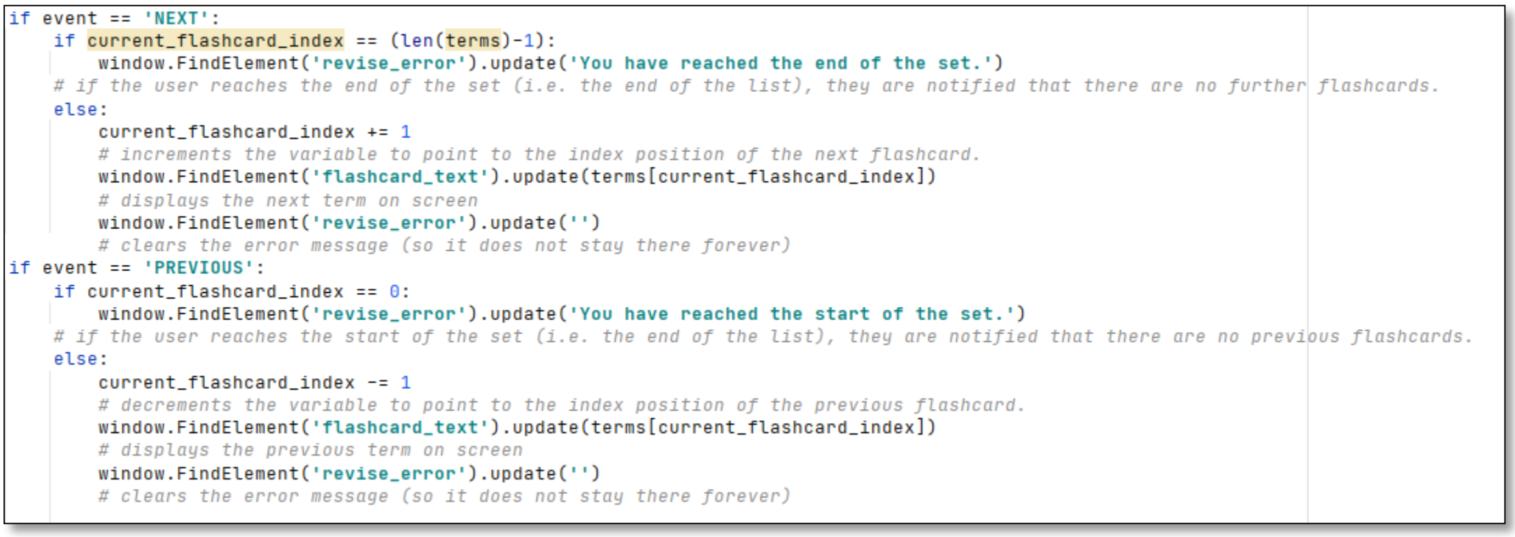


```
if event == 'NEXT':
    current_flashcard_index += 1
    # increments the variable to point to the index position of the next flashcard.
    window.FindElement('flashcard_text').update(terms[current_flashcard_index])
    # displays the next term on screen
```

However, I need to add something that tells the user when they have reached the end of the set (and so there is no 'next' flashcard in the set) to avoid errors occurring. It seems most logical to implement an if else statement checking whether the current flashcard index is equal to the maximum index in the list (because if it is, there can be no next flashcard). I will also need to add an error message to the display which will be updated to alert the user they have reached the end of the set. This will work in a similar way to alert the user to if they have reached the start of the set when using the 'previous' button. All of this being implemented is shown below:

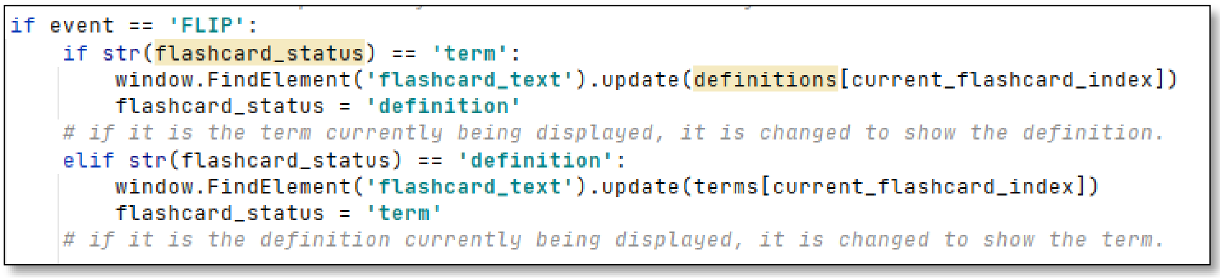


```
ReviseFlashcard = [[sg.Text('', font=("Helvetica", 14, "bold"), text_color='blue', justification='center', key='flashcard_text', border_width=5, size=(37,10), backgrou
                   [sg.Text('               '), sg.Button('Previous', key='PREVIOUS', font=("Helvetica", 12), size=(7,1)), sg.Button('Next', key='NEXT', font=("Helvetica",
                    sg.Button('Flip', key='FLIP', font=("Helvetica", 12), size=(7,1)), sg.Button('Delete', key='DELETE', font=("Helvetica", 12), size=(7,1))],
                   [sg.Text('', key='revise_error', font=("Helvetica", 15, "bold"), text_color='red', justification='center', size=(38,3))],
                   [sg.Button('BACK', font=("Helvetica", 15), key='BACK7', button_color=('red', 'white'))]]
#                   Layout for revising flashcards, consisting of a white text box and 4 buttons underneath, as well as an error message and back button.
```



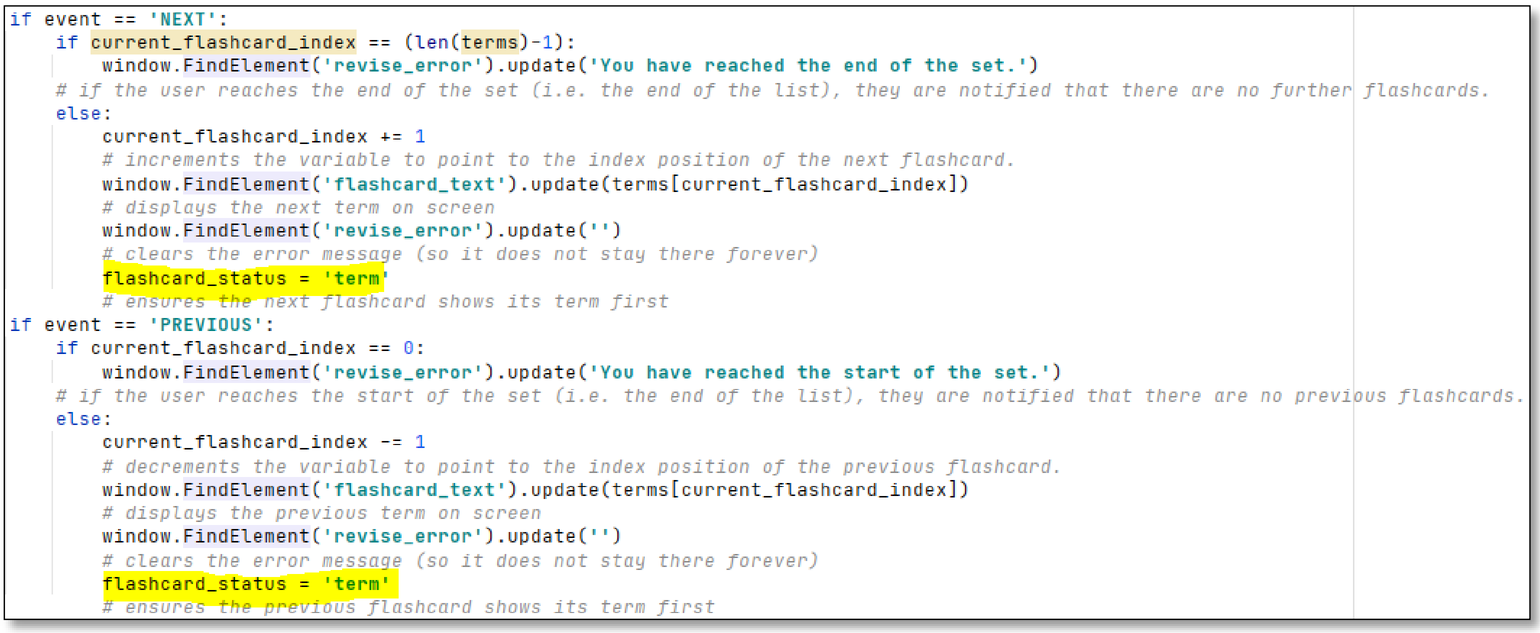
```
if event == 'NEXT':
    if current_flashcard_index == (len(terms)-1):
        window.FindElement('revise_error').update('You have reached the end of the set.')
    # if the user reaches the end of the set (i.e. the end of the list), they are notified that there are no further flashcards.
    else:
        current_flashcard_index += 1
        # increments the variable to point to the index position of the next flashcard.
        window.FindElement('flashcard_text').update(terms[current_flashcard_index])
        # displays the next term on screen
        window.FindElement('revise_error').update('')
        # clears the error message (so it does not stay there forever)
if event == 'PREVIOUS':
    if current_flashcard_index == 0:
        window.FindElement('revise_error').update('You have reached the start of the set.')
    # if the user reaches the start of the set (i.e. the end of the list), they are notified that there are no previous flashcards.
    else:
        current_flashcard_index -= 1
        # decrements the variable to point to the index position of the previous flashcard.
        window.FindElement('flashcard_text').update(terms[current_flashcard_index])
        # displays the previous term on screen
        window.FindElement('revise_error').update('')
        # clears the error message (so it does not stay there forever)
```

I have then implemented the functionality of the flip button, which simply changes what is displayed on the screen from the term to the definition or vice versa depending on the flashcard status.



```python
if event == 'FLIP':
    if str(flashcard_status) == 'term':
        window.FindElement('flashcard_text').update(definitions[current_flashcard_index])
        flashcard_status = 'definition'
    # if it is the term currently being displayed, it is changed to show the definition.
    elif str(flashcard_status) == 'definition':
        window.FindElement('flashcard_text').update(terms[current_flashcard_index])
        flashcard_status = 'term'
    # if it is the definition currently being displayed, it is changed to show the term.
```

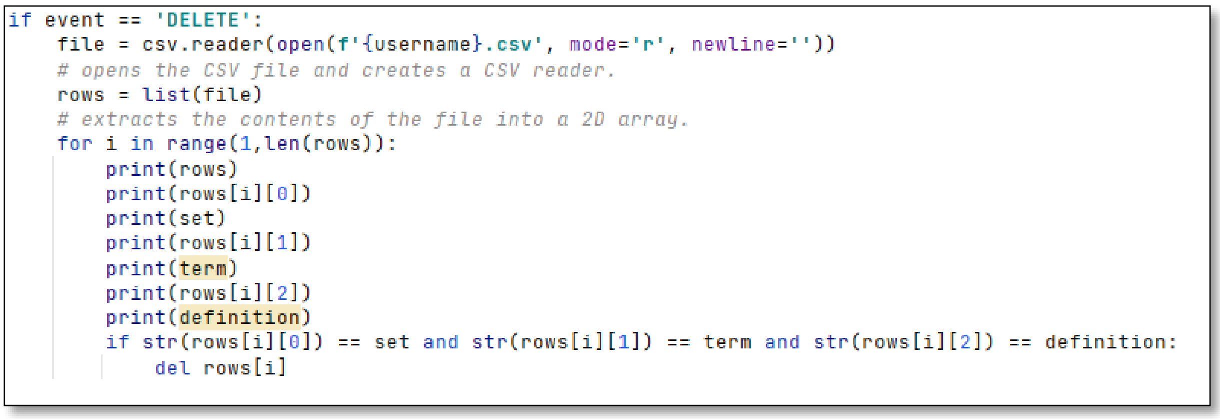
I have also added a piece of code to reset the flashcard status to term when the next/previous flashcard is viewed so that the term appears before the definition.



```python
if event == 'NEXT':
    if current_flashcard_index == (len(terms)-1):
        window.FindElement('revise_error').update('You have reached the end of the set.')
    # if the user reaches the end of the set (i.e. the end of the list), they are notified that there are no further flashcards.
    else:
        current_flashcard_index += 1
        # increments the variable to point to the index position of the next flashcard.
        window.FindElement('flashcard_text').update(terms[current_flashcard_index])
        # displays the next term on screen
        window.FindElement('revise_error').update('')
        # clears the error message (so it does not stay there forever)
        flashcard_status = 'term'
        # ensures the next flashcard shows its term first
if event == 'PREVIOUS':
    if current_flashcard_index == 0:
        window.FindElement('revise_error').update('You have reached the start of the set.')
    # if the user reaches the start of the set (i.e. the end of the list), they are notified that there are no previous flashcards.
    else:
        current_flashcard_index -= 1
        # decrements the variable to point to the index position of the previous flashcard.
        window.FindElement('flashcard_text').update(terms[current_flashcard_index])
        # displays the previous term on screen
        window.FindElement('revise_error').update('')
        # clears the error message (so it does not stay there forever)
        flashcard_status = 'term'
        # ensures the previous flashcard shows its term first
```

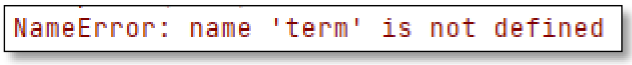
Finally, I need to implement the functionality of the delete button. The delete button will need to remove the flashcard the user is currently looking at from the set (i.e. by deleting it from the users CSV file), and then display the next flashcard on the screen. I also need to consider what to do when the user is deleting the last flashcard in the set (as there is no next flashcard to display, so in this case I will have to display the previous flashcard), and when the user deletes the last flashcard in the set (as there will be no other flashcards to display at this point).

I have first created the code to delete a flashcard from the users CSV file. This prototype contains some print statements for me to test it:



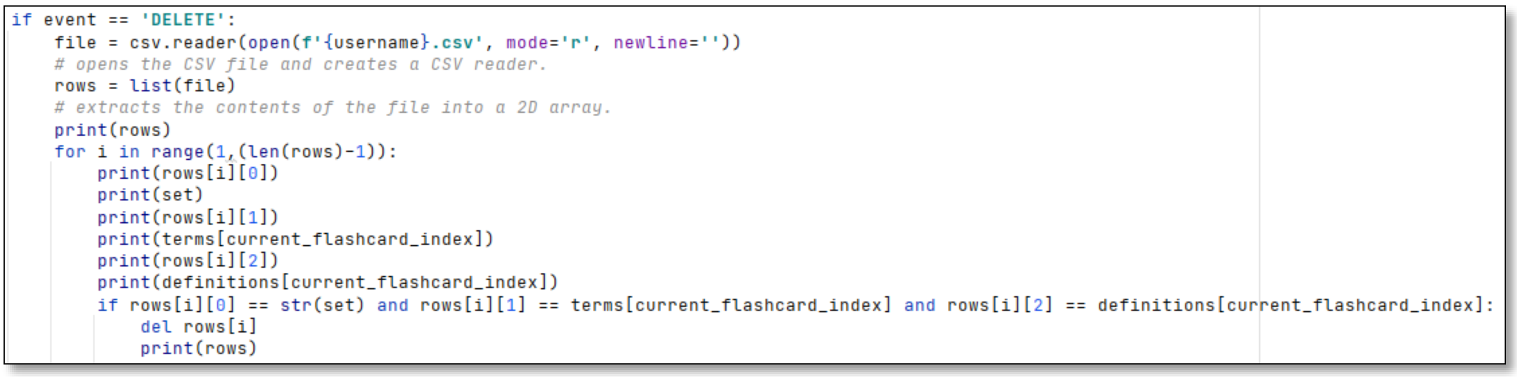
```
if event == 'DELETE':
    file = csv.reader(open(f'{username}.csv', mode='r', newline=''))
    # opens the CSV file and creates a CSV reader.
    rows = list(file)
    # extracts the contents of the file into a 2D array.
    for i in range(1,len(rows)):
        print(rows)
        print(rows[i][0])
        print(set)
        print(rows[i][1])
        print(term)
        print(rows[i][2])
        print(definition)
        if str(rows[i][0]) == set and str(rows[i][1]) == term and str(rows[i][2]) == definition:
            del rows[i]
```

However, when I run this piece of code, I receive the following error:



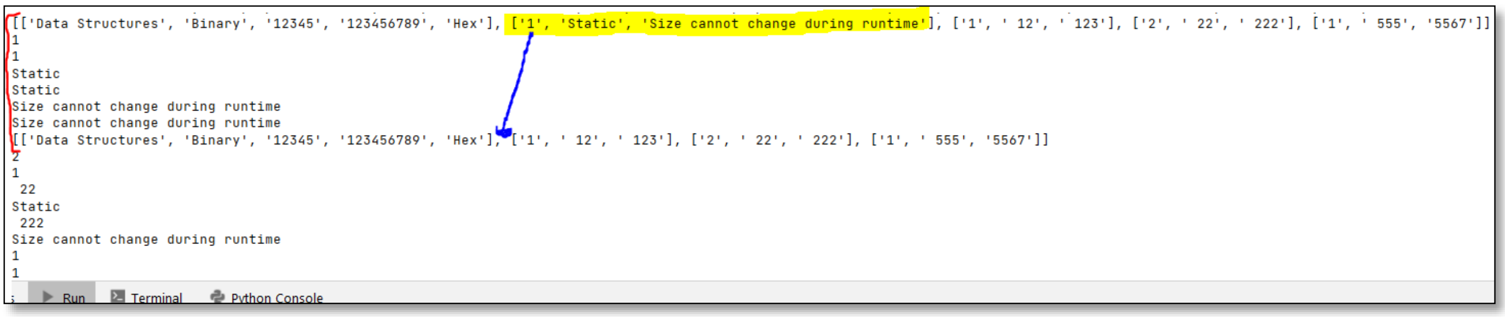
```
NameError: name 'term' is not defined
```

This is because term and definition haven't been defined. I need the if statement with the 3 conditions to ensure the flashcard being deleted is the correct one and not one from another set. To solve this, I realised I can just use the relevant elements in the terms and definitions arrays instead of storing them as separate variables:



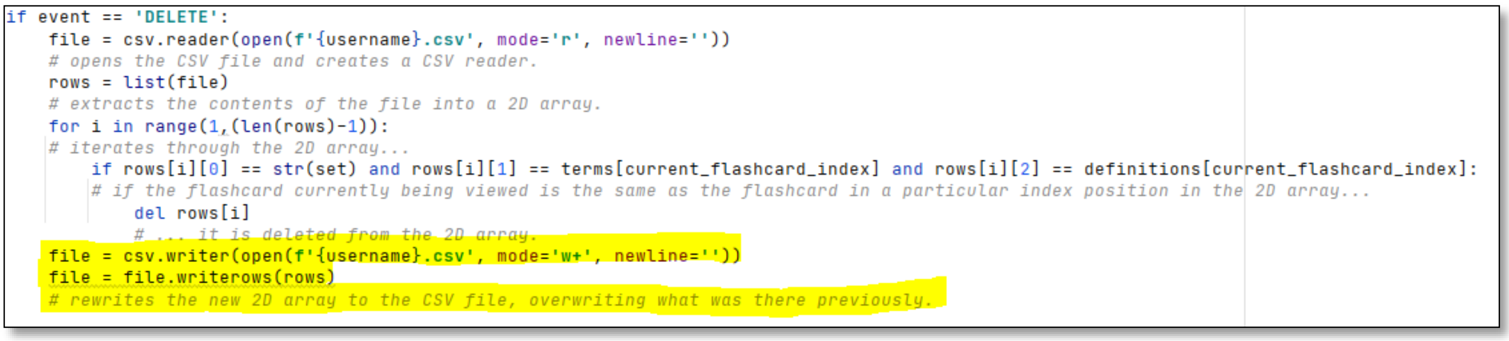
```
if event == 'DELETE':
    file = csv.reader(open(f'{username}.csv', mode='r', newline=''))
    # opens the CSV file and creates a CSV reader.
    rows = list(file)
    # extracts the contents of the file into a 2D array.
    print(rows)
    for i in range(1,(len(rows)-1)):
        print(rows[i][0])
        print(set)
        print(rows[i][1])
        print(terms[current_flashcard_index])
        print(rows[i][2])
        print(definitions[current_flashcard_index])
        if rows[i][0] == str(set) and rows[i][1] == terms[current_flashcard_index] and rows[i][2] == definitions[current_flashcard_index]:
            del rows[i]
            print(rows)
```

When I run this, the output is as follows:



```
[['Data Structures', 'Binary', '12345', '123456789', 'Hex'], ['1', 'Static', 'Size cannot change during runtime'], ['1', ' 12', ' 123'], ['2', ' 22', ' 222'], ['1', ' 555', '5567']]
1
1
Static
Static
Size cannot change during runtime
Size cannot change during runtime
[['Data Structures', 'Binary', '12345', '123456789', 'Hex'], ['1', ' 12', ' 123'], ['2', ' 22', ' 222'], ['1', ' 555', '5567']]
2
1
 22
Static
 222
Size cannot change during runtime
1
1
```

The bit within the red line shows that all 3 conditions are being met, and the correct sub-array is being deleted. I can now remove the print statements as I know this works and rewrite the new 2D array back to the file:



```
if event == 'DELETE':
    file = csv.reader(open(f'{username}.csv', mode='r', newline=''))
    # opens the CSV file and creates a CSV reader.
    rows = list(file)
    # extracts the contents of the file into a 2D array.
    for i in range(1,(len(rows)-1)):
    # iterates through the 2D array...
        if rows[i][0] == str(set) and rows[i][1] == terms[current_flashcard_index] and rows[i][2] == definitions[current_flashcard_index]:
        # if the flashcard currently being viewed is the same as the flashcard in a particular index position in the 2D array...
            del rows[i]
            # ... it is deleted from the 2D array.
    file = csv.writer(open(f'{username}.csv', mode='w+', newline=''))
    file = file.writerows(rows)
    # rewrites the new 2D array to the CSV file, overwriting what was there previously.
```

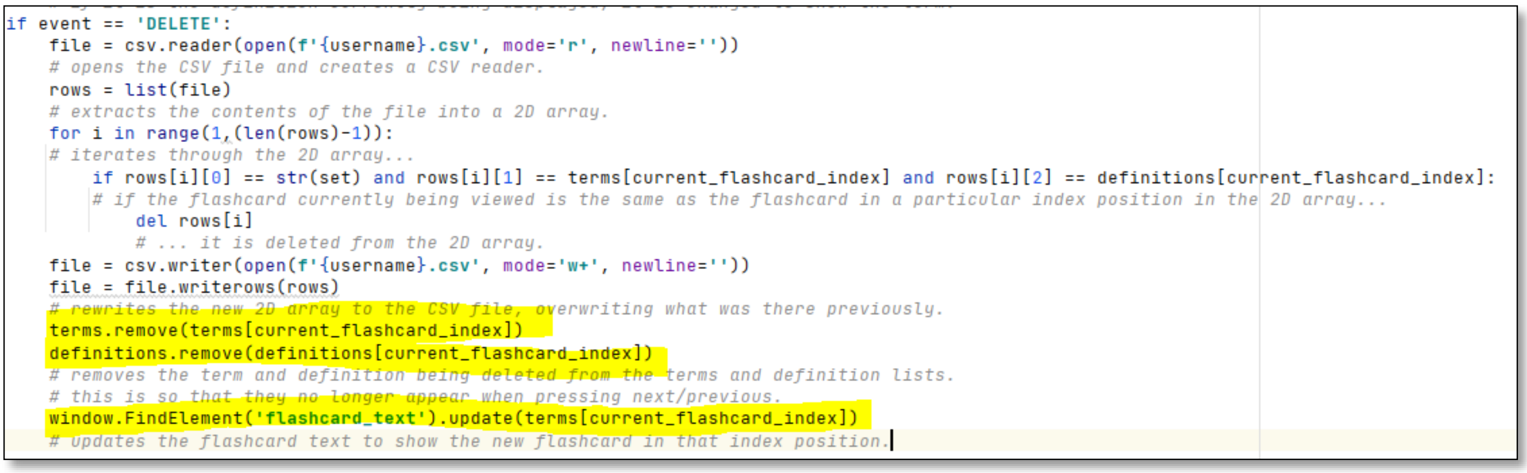
I am now going to carry out some testing to test this prototype of my solution (testing evidence for the following tests can be found in the testing evidence PowerPoint):

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
7.2.1	Flip flashcard via button press	Mouse click	The flashcard will be flipped (e.g. if term was on the screen, definition will now be on the screen and vice versa)	To check whether the correct corresponding definition to the term (or vice versa) is displayed on screen when the button is pressed.	The flashcard is successfully 'flipped', displaying a terms corresponding definition and vice versa
7.2.2	Viewing the next flashcard via a button press	Mouse click	The next flashcard read from the users .csv file will appear on the screen.	To check whether a new flashcard is displayed on screen from the correct set.	The next flashcard successfully shows up on screen, also alerting the user as to when they have reached the end of the set.
7.2.3	Viewing the previous flashcard via a button press	Mouse click	The previous flashcard read from the users .csv file will appear on the screen.	To check whether the previously displayed flashcard is displayed on screen when the button is pressed.	The previous flashcard successfully shows up on screen, also alerting the user as to when they have reached the start of the set.
7.2.4	Deleting flashcard via a button press	Mouse click	The flashcard will be deleted from the users .csv file, and the screen will turn blank as the procedure to search for the next flashcard has not been ran yet.	To check whether pressing the button to remove a flashcard removes it from the users .csv file.	The deleted flashcard is successfully removed from the users CSV file.

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
7.3.1	Flip flashcard via button press	Mouse click	The flashcard will be flipped (e.g. if term was on the screen, definition will now be on the screen and vice versa)	Checking this feature hasn't been affected by the change to deleting flashcards (test 7.3.4).	The flashcard is successfully 'flipped', displaying a terms corresponding definition and vice versa
7.3.2	Viewing the next flashcard via a button press	Mouse click	The next flashcard read from the users .csv file will appear on the screen.	Checking this feature hasn't been affected by the change to deleting flashcards (test 7.3.4).	The next flashcard successfully shows up on screen, also alerting the user as to when they have reached the end of the set.

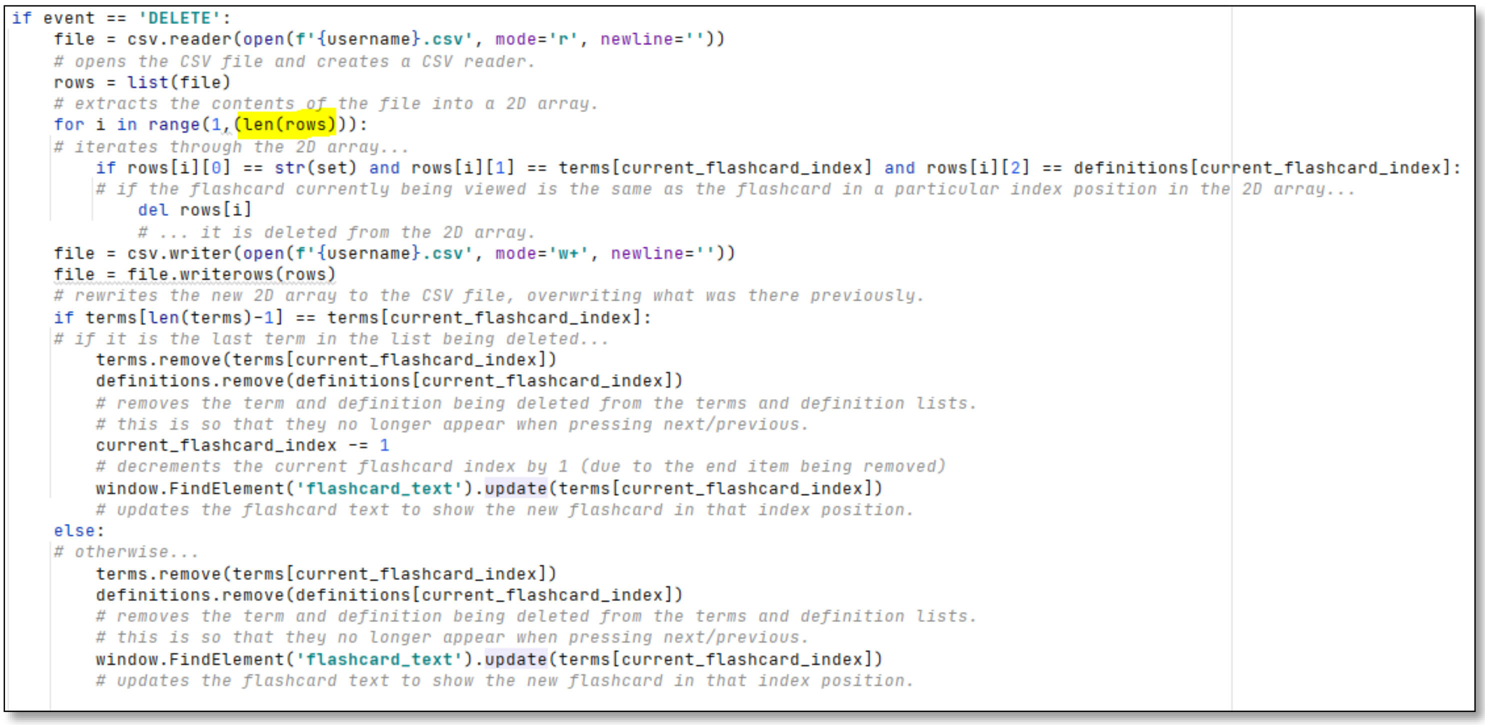
7.3.3	Viewing the previous flashcard via a button press	Mouse click	The previous flashcard read from the users .csv file will appear on the screen.	Checking this feature hasn't been affected by the change to deleting flashcards (test 7.3.4).	The previous flashcard successfully shows up on screen, also alerting the user as to when they have reached the start of the set.
7.3.4	Deleting flashcard via a button press	Mouse click	The flashcard will be deleted from the users .csv file and the next flashcard procedure is then ran to make the next term appear on the screen.	To check whether the next flashcard automatically appears on screen when the user deletes a flashcard.	The flashcard is deleted from the users CSV file but the next flashcard is not displayed.

To successfully complete test 7.3.4, I need to create a way for the next flashcard to appear on screen when a flashcard is deleted, but also if the last flashcard in the list is deleted, move to the previous flashcard. Now, when an item is deleted from an array, the next item will automatically fall back into its place, thus I do not need to increment any variables, just update the text element to display the next flashcard:



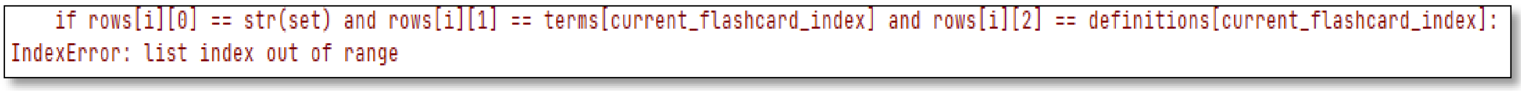
```
if event == 'DELETE':
    file = csv.reader(open(f'{username}.csv', mode='r', newline=''))
    # opens the CSV file and creates a CSV reader.
    rows = list(file)
    # extracts the contents of the file into a 2D array.
    for i in range(1,(len(rows)-1)):
    # iterates through the 2D array...
        if rows[i][0] == str(set) and rows[i][1] == terms[current_flashcard_index] and rows[i][2] == definitions[current_flashcard_index]:
        # if the flashcard currently being viewed is the same as the flashcard in a particular index position in the 2D array...
            del rows[i]
            # ... it is deleted from the 2D array.
    file = csv.writer(open(f'{username}.csv', mode='w+', newline=''))
    file = file.writerows(rows)
    # rewrites the new 2D array to the CSV file, overwriting what was there previously.
    terms.remove(terms[current_flashcard_index])
    definitions.remove(definitions[current_flashcard_index])
    # removes the term and definition being deleted from the terms and definition lists.
    # this is so that they no longer appear when pressing next/previous.
    window.FindElement('flashcard_text').update(terms[current_flashcard_index])
    # updates the flashcard text to show the new flashcard in that index position.
```

I also need to implement the code for if it is the last item in the array being deleted (as there will be no further items in the array to fall into its place). All I need to do is decrement the current flashcard index if this is the case (I have also removed the -1 from the len() function in for loop (highlighted) as this should not have been there:



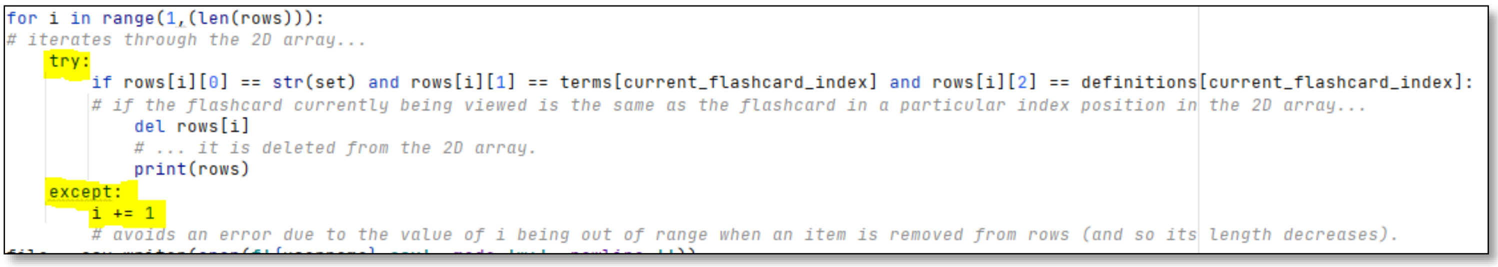
```python
if event == 'DELETE':
    file = csv.reader(open(f'{username}.csv', mode='r', newline=''))
    # opens the CSV file and creates a CSV reader.
    rows = list(file)
    # extracts the contents of the file into a 2D array.
    for i in range(1,(len(rows))):
    # iterates through the 2D array...
        if rows[i][0] == str(set) and rows[i][1] == terms[current_flashcard_index] and rows[i][2] == definitions[current_flashcard_index]:
        # if the flashcard currently being viewed is the same as the flashcard in a particular index position in the 2D array...
            del rows[i]
            # ... it is deleted from the 2D array.
    file = csv.writer(open(f'{username}.csv', mode='w+', newline=''))
    file = file.writerows(rows)
    # rewrites the new 2D array to the CSV file, overwriting what was there previously.
    if terms[len(terms)-1] == terms[current_flashcard_index]:
    # if it is the last term in the list being deleted...
        terms.remove(terms[current_flashcard_index])
        definitions.remove(definitions[current_flashcard_index])
        # removes the term and definition being deleted from the terms and definition lists.
        # this is so that they no longer appear when pressing next/previous.
        current_flashcard_index -= 1
        # decrements the current flashcard index by 1 (due to the end item being removed)
        window.FindElement('flashcard_text').update(terms[current_flashcard_index])
        # updates the flashcard text to show the new flashcard in that index position.
    else:
    # otherwise...
        terms.remove(terms[current_flashcard_index])
        definitions.remove(definitions[current_flashcard_index])
        # removes the term and definition being deleted from the terms and definition lists.
        # this is so that they no longer appear when pressing next/previous.
        window.FindElement('flashcard_text').update(terms[current_flashcard_index])
        # updates the flashcard text to show the new flashcard in that index position.
```

However, when testing this I received the following error:



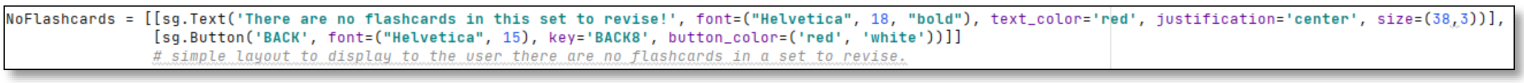
```
    if rows[i][0] == str(set) and rows[i][1] == terms[current_flashcard_index] and rows[i][2] == definitions[current_flashcard_index]:
IndexError: list index out of range
```

This is because when an item gets removed from rows, its length decreases by 1 but the for loop still goes to the original length (which is now out of range). To fix this, I inserted a simple try and except loop, which just increments i whenever this happens to avoid the error:

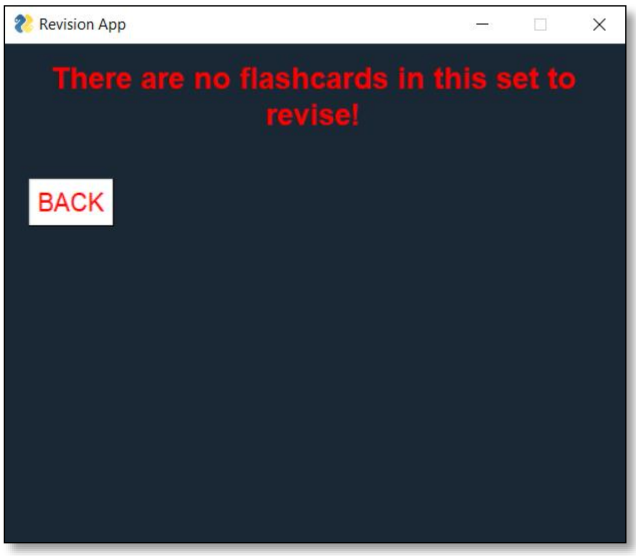


```python
for i in range(1,(len(rows))):
# iterates through the 2D array...
    try:
        if rows[i][0] == str(set) and rows[i][1] == terms[current_flashcard_index] and rows[i][2] == definitions[current_flashcard_index]:
        # if the flashcard currently being viewed is the same as the flashcard in a particular index position in the 2D array...
            del rows[i]
            # ... it is deleted from the 2D array.
            print(rows)
    except:
        i += 1
        # avoids an error due to the value of i being out of range when an item is removed from rows (and so its length decreases).
```

I now also need to implement a method of displaying to the user if the set has become empty and thus there is nothing in it to revise. The way I have decided to do this (which will apply to all sets at all times) is if the user ever decides to ‘revise’ a set which has no flashcards in it, they will be taken to a separate layout which tells them this and then has a back button taking them back to the set menu. This stops users revising sets which have no flashcards. This screen will also appear when the user deletes the last flashcard in a set.
This is the layout:



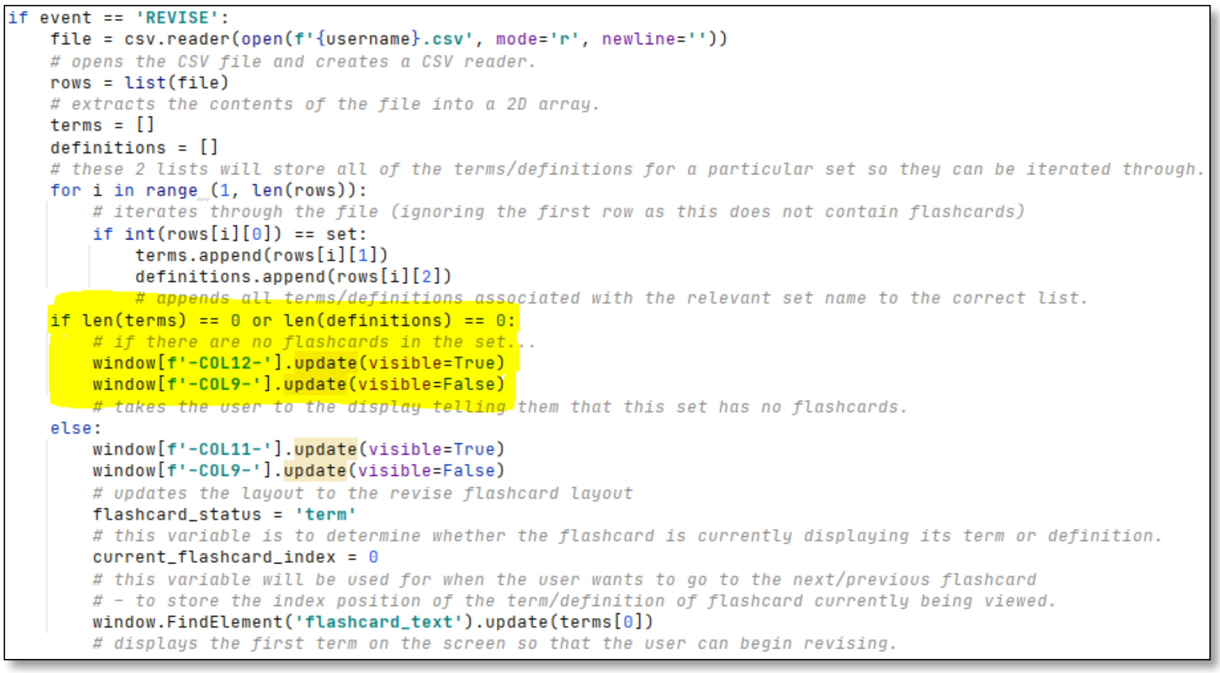
```python
NoFlashcards = [[sg.Text('There are no flashcards in this set to revise!', font=("Helvetica", 18, "bold"), text_color='red', justification='center', size=(38,3))],
                [sg.Button('BACK', font=("Helvetica", 15), key='BACK8', button_color=('red', 'white'))]]
                # simple layout to display to the user there are no flashcards in a set to revise.
```



I have also added this in the column layout.

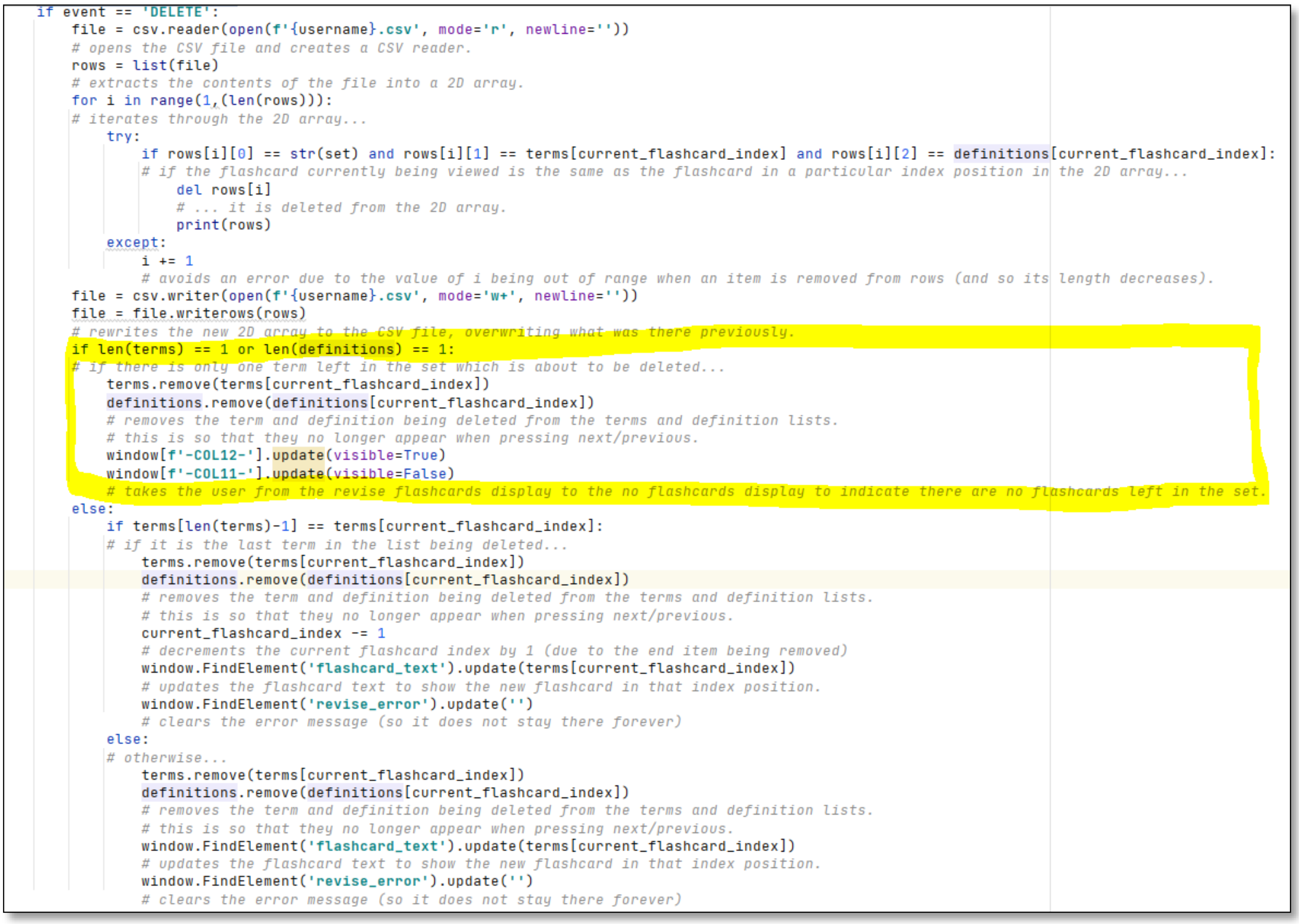
I now need to implement this into my code so that it is made visible when the user tries to revise a set with no flashcards in it. I have started by implementing the functionality of the back button, which will take the user back to the set menu:

I have then made it so that the user is taken to this screen if they initially choose to revise a set that has nothing in it:



```
if event == 'REVISE':
    file = csv.reader(open(f'{username}.csv', mode='r', newline=''))
    # opens the CSV file and creates a CSV reader.
    rows = list(file)
    # extracts the contents of the file into a 2D array.
    terms = []
    definitions = []
    # these 2 lists will store all of the terms/definitions for a particular set so they can be iterated through.
    for i in range (1, len(rows)):
        # iterates through the file (ignoring the first row as this does not contain flashcards)
        if int(rows[i][0]) == set:
            terms.append(rows[i][1])
            definitions.append(rows[i][2])
            # appends all terms/definitions associated with the relevant set name to the correct list.
    if len(terms) == 0 or len(definitions) == 0:
        # if there are no flashcards in the set...
        window[f'-COL12-'].update(visible=True)
        window[f'-COL9-'].update(visible=False)
        # takes the user to the display telling them that this set has no flashcards.
    else:
        window[f'-COL11-'].update(visible=True)
        window[f'-COL9-'].update(visible=False)
        # updates the layout to the revise flashcard layout
        flashcard_status = 'term'
        # this variable is to determine whether the flashcard is currently displaying its term or definition.
        current_flashcard_index = 0
        # this variable will be used for when the user wants to go to the next/previous flashcard
        # - to store the index position of the term/definition of flashcard currently being viewed.
        window.FindElement('flashcard_text').update(terms[0])
        # displays the first term on the screen so that the user can begin revising.
```

I have then added a piece of code which also takes the user to this screen if they delete all flashcards in a set (and thus there are none left to revise):



```
if event == 'DELETE':
    file = csv.reader(open(f'{username}.csv', mode='r', newline=''))
    # opens the CSV file and creates a CSV reader.
    rows = list(file)
    # extracts the contents of the file into a 2D array.
    for i in range(1,(len(rows))):
    # iterates through the 2D array...
        try:
            if rows[i][0] == str(set) and rows[i][1] == terms[current_flashcard_index] and rows[i][2] == definitions[current_flashcard_index]:
            # if the flashcard currently being viewed is the same as the flashcard in a particular index position in the 2D array...
                del rows[i]
                # ... it is deleted from the 2D array.
                print(rows)
        except:
            i += 1
            # avoids an error due to the value of i being out of range when an item is removed from rows (and so its length decreases).
    file = csv.writer(open(f'{username}.csv', mode='w+', newline=''))
    file = file.writerows(rows)
    # rewrites the new 2D array to the CSV file, overwriting what was there previously.
    if len(terms) == 1 or len(definitions) == 1:
    # if there is only one term left in the set which is about to be deleted...
        terms.remove(terms[current_flashcard_index])
        definitions.remove(definitions[current_flashcard_index])
        # removes the term and definition being deleted from the terms and definition lists.
        # this is so that they no longer appear when pressing next/previous.
        window[f'-COL12-'].update(visible=True)
        window[f'-COL11-'].update(visible=False)
        # takes the user from the revise flashcards display to the no flashcards display to indicate there are no flashcards left in the set.
    else:
        if terms[len(terms)-1] == terms[current_flashcard_index]:
        # if it is the last term in the list being deleted...
            terms.remove(terms[current_flashcard_index])
            definitions.remove(definitions[current_flashcard_index])
            # removes the term and definition being deleted from the terms and definition lists.
            # this is so that they no longer appear when pressing next/previous.
            current_flashcard_index -= 1
            # decrements the current flashcard index by 1 (due to the end item being removed)
            window.FindElement('flashcard_text').update(terms[current_flashcard_index])
            # updates the flashcard text to show the new flashcard in that index position.
            window.FindElement('revise_error').update('')
            # clears the error message (so it does not stay there forever)
        else:
        # otherwise...
            terms.remove(terms[current_flashcard_index])
            definitions.remove(definitions[current_flashcard_index])
            # removes the term and definition being deleted from the terms and definition lists.
            # this is so that they no longer appear when pressing next/previous.
            window.FindElement('flashcard_text').update(terms[current_flashcard_index])
            # updates the flashcard text to show the new flashcard in that index position.
            window.FindElement('revise_error').update('')
            # clears the error message (so it does not stay there forever)
```

All of this can be seen working in the following tests, testing evidence for which are available in the testing evidence PowerPoint.

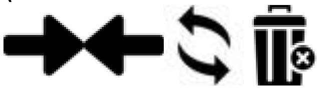
Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
7.3.1	Flip flashcard via button press	Mouse click	The flashcard will be flipped (e.g. if term was on the screen, definition will now be on the screen and vice versa)	Checking this feature hasn't been affected by the change to deleting flashcards (test 7.3.4).	The flashcard is successfully 'flipped', displaying a terms corresponding definition and vice versa
7.3.2	Viewing the next flashcard via a button press	Mouse click	The next flashcard read from the users .csv file will appear on the screen.	Checking this feature hasn't been affected by the change to deleting flashcards (test 7.3.4).	The next flashcard successfully shows up on screen, also alerting the user as to when they have reached the end of the set.
7.3.3	Viewing the previous flashcard via a button press	Mouse click	The previous flashcard read from the users .csv file will appear on the screen.	Checking this feature hasn't been affected by the change to deleting flashcards (test 7.3.4).	The previous flashcard successfully shows up on screen, also alerting the user as to when they have reached the start of the set.

7.3.4	Deleting flashcard via a button press	Mouse click	The flashcard will be deleted from the users .csv file and the next flashcard procedure is then ran to make the next term appear on the screen.	To check whether the next flashcard automatically appears on screen when the user deletes a flashcard.	The flashcard is deleted from the users CSV file and the next/previous flashcard is displayed.

Revising flashcards – Review

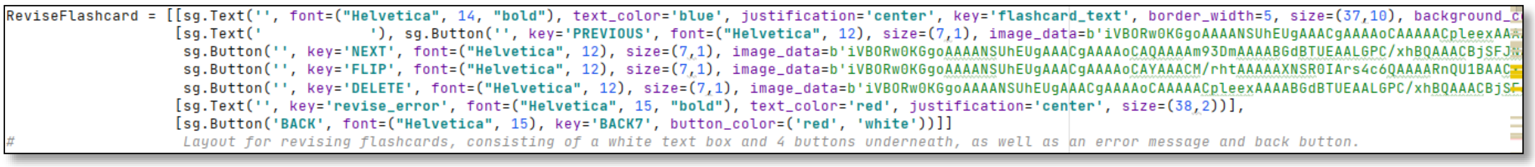
Stakeholder feedback

I sent this prototype of my solution to all of my stakeholders as I feel this is an integral part of my app, and they all said they were happy with the functionality but would like the interface to be a little less bland. All stakeholders said the layout was very clear and the flashcard was big enough. Marcus suggested putting shapes instead of words on the next, previous, flip and delete buttons (such as arrows or crosses). I feel this is a good idea to make the interface look more user friendly, whilst still maintaining its functionality. However, I will need to get some images off of Google and make them the correct size for this to work. I have found some images that I will be using, and I have used the website http://www.simpleimageresizer.com/upload#.YDpr3Gj7RPZ to make all of my images 40x40 (as I feel this is a suitable size for the display).



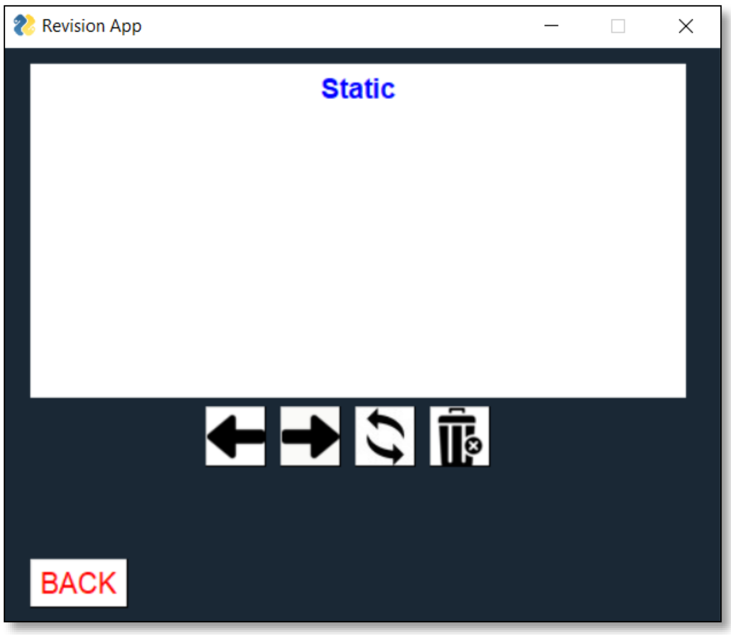
I think these images are suitable and self-explanatory. I asked my stakeholders and they agreed that they would know what these images mean in the context of revising flashcards.

As this is something I have never implemented before, I looked online at the website https://pysimplegui.readthedocs.io/en/latest/cookbook/#step-4-use-base64-variable-to-make-your-button for help. Following the advice on the website, I first had to convert the image to base64. I used the website https://base64.guru/converter/encode/image to convert all of my images to base64. I then had to insert this image data as a parameter in the button element to make the button display the image, which I then did as shown below:



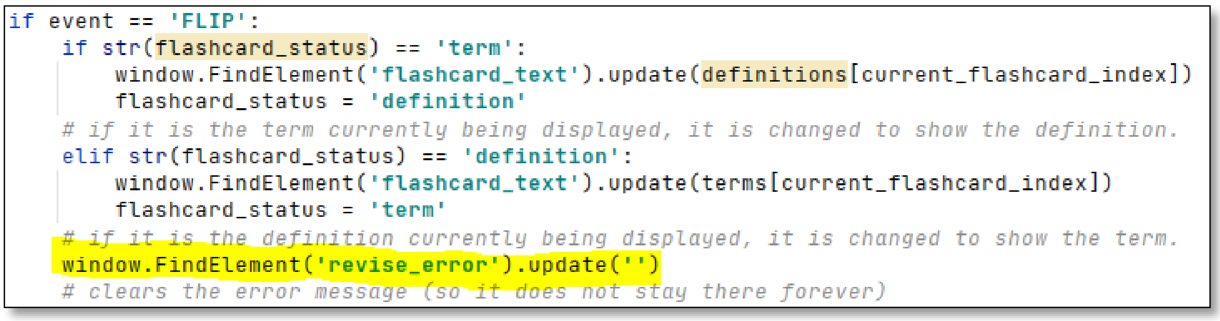
```
ReviseFlashcard = [[sg.Text('', font=("Helvetica", 14, "bold"), text_color='blue', justification='center', key='flashcard_text', border_width=5, size=(37,10), background_c
                   [sg.Text('             '), sg.Button('', key='PREVIOUS', font=("Helvetica", 12), size=(7,1), image_data=b'iVBORw0KGgoAAAANSUhEUgAAACgAAAAoCAAAAACpleexAA
                    sg.Button('', key='NEXT', font=("Helvetica", 12), size=(7,1), image_data=b'iVBORw0KGgoAAAANSUhEUgAAACgAAAAoCAQAAAAm93DmAAAABGdBTUEAALGPC/xhBQAAACBjSFJ
                    sg.Button('', key='FLIP', font=("Helvetica", 12), size=(7,1), image_data=b'iVBORw0KGgoAAAANSUhEUgAAACgAAAAoCAYAAACM/rhtAAAAAXNSR0IArs4c6QAAAARnQU1BAAC
                    sg.Button('', key='DELETE', font=("Helvetica", 12), size=(7,1), image_data=b'iVBORw0KGgoAAAANSUhEUgAAACgAAAAoCAAAAACpleexAAAABGdBTUEAALGPC/xhBQAAACBjSF
                   [sg.Text('', key='revise_error', font=("Helvetica", 15, "bold"), text_color='red', justification='center', size=(38,2))],
                   [sg.Button('BACK', font=("Helvetica", 15), key='BACK7', button_color=('red', 'white'))]]
#                   Layout for revising flashcards, consisting of a white text box and 4 buttons underneath, as well as an error message and back button.
```

The image data strings are very long and so I cannot get all of it on one screenshot. This is how the layout now looks:



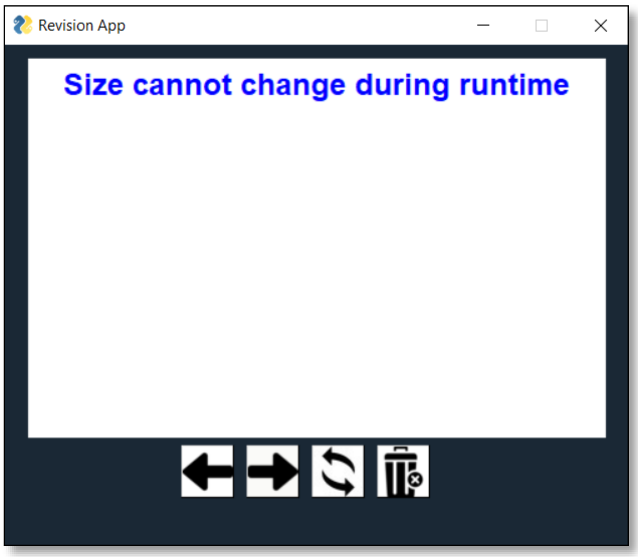
I sent this new layout to my stakeholders and they were much happier with it and said it looks a lot nicer. As a result of this feedback, in the future, I may use images instead of words in other areas of my program, such as the main menu or quiz.

Vivek also noted than when an error message appears, and then the flashcard is flipped, the error message remains there and it would be better if it went hidden when the user flipped the flashcard. I have fixed this by adding a single line of code:



```
if event == 'FLIP':
    if str(flashcard_status) == 'term':
        window.FindElement('flashcard_text').update(definitions[current_flashcard_index])
        flashcard_status = 'definition'
    # if it is the term currently being displayed, it is changed to show the definition.
    elif str(flashcard_status) == 'definition':
        window.FindElement('flashcard_text').update(terms[current_flashcard_index])
        flashcard_status = 'term'
    # if it is the definition currently being displayed, it is changed to show the term.
    window.FindElement('revise_error').update('')
    # clears the error message (so it does not stay there forever)
```

They also said it would be better if the text size on the flashcards was increased slightly, so I increased the text size slightly so it now appears like this on the screen:



Success Criteria

Now I have fully implemented the ability for users to revise their sets of flashcards, as well as the ability to fully edit them by adding/deleting flashcards to sets. So far I have met:

- SC10 - The stakeholder would like the system to be user-friendly and easy to understand.
- SC1 - The stakeholder would like the system to allow them to create an account if they haven't already got one.
- SC2 - The stakeholder would like the system to allow them to log in to their account if they have already created one.
- SC9 - The stakeholder would like the system to have a clear main menu.
- SC3 - The stakeholder would like the system to allow them to create their own sets of flashcards.
- SC4 - The stakeholder would like the system to view sets of flashcards they have already created.
- SC5 - The stakeholder would like the system to allow them to revise sets of flashcards they have created.
- SC11 - The stakeholder would like the system to allow them to edit already existing sets of flashcards.

Changed made to original plan

Not many massive changes were made in this section, apart from the changes in interface by using images on buttons instead of text.

Evaluation Section

In this section of my report, I will be testing and evaluating my final solution, with input from my stakeholders on how well success criteria/usability features have been met. Criteria that have been met will be tested, and the criteria that have not been met will be justified and I will explain how these features could be met in future development.

Success Criteria post-development testing

I will start by evaluating each of my success criteria lined out in the analysis section and justifying how well they have been met. Testing evidence for all of the following tests can be found in the '**Post development** testing evidence' PowerPoint.

Success Criteria 1: The stakeholder would like the system to allow them to create an account if they haven't already got one.

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
10.1.1	Attempting to submit valid account details	String 'username123' String 'password123' String 'password123' String 'bradleymak2003 @gmail.com' **VALID DATA**	The users account will be created, their details will be saved to their .csv file, and they will be taken to the main menu.	Ensures the user can successfully create an account using valid account details.	**The users details were successfully written to the text file (with the password hashed) since this is valid data, and they are taken to the main menu – their account is now successfully created.**
10.1.2	Attempting to submit invalid account details	String 'username123' String 'password123' String 'password123' Integer 123456789 **INVALID DATA**	An error message will appear at the bottom of the screen notifying the user of the problem (invalid email in this case).	Checks whether the system detects when an invalid email has been entered (and does not crash).	**An error message appears alerting the user to the fact that the email they have entered is not valid.**
10.1.3	Attempting to submit incorrect account details	String 'username123' String 'password123' String 'password' String 'bradleymak2003 @gmail.com' **INVALID DATA**	An error message will appear at the bottom of the screen notifying the user that the 2 passwords they entered do not match (i.e. they mistyped their password).	Checks whether the system detects when the two password fields do not match.	**An error message appears alerting the user to the fact that the two passwords they have entered do not match.**
10.1.4	Attempting to submit empty fields	String '' String '' String '' String 'bradleymak2003 @gmail.com' – I have still entered an email to ensure the	An error message will appear at the bottom of the screen notifying the user that they have left fields blank and	Checks that the user cannot enter empty fields for their username/password.	**An error message appears alerting the user to the fact that they have left fields blank and must fill them all with valid**

		error does not appear just because the email is 'invalid', but rather because the user hasn't filled the username and password fields. **INVALID DATA**	they must all be filled.		**details to create an account.**
10.1.5	Attempting to create two accounts with the same username	Create one account with the username 'username123', and then attempt to create another account with the same username.	An error message will appear saying that an account already exists with that username.	Checks the user cannot create an account with the same username as an already created account, as this will cause issues with other areas of the program.	**An error message appears alerting the user to the fact that an account already exists with that username.**

I have added two new tests to this success criteria than I originally planned in the design section as some key robustness/features were not originally covered. Firstly, I have added test 10.1.4 as I feel like testing that the user cannot create an account without entering any details is a necessary requirement. I have also added test 10.1.5 to ensure that the user cannot create 2 accounts with the same username (as this will cause issues with other areas of the program as the user will no longer have unique identification).

All of these tests being completed shows that this feature functions correctly (i.e. when the user enters valid details – an unused username, 2 passwords that match and a gmail address – an account is successfully created and the details are written to userinfo.txt, but if the details are invalid – either a username that has already been used, 2 passwords that *don't* match, or an invalid email address – the relevant error message is correctly displayed on screen), and is robust (i.e. the program does not crash no matter what the inputs are). This success criteria has been fully met as it has passed all of the tests I have carried out. As such, there are no further improvements that would need to be made in later development.

Success Criteria 2: The stakeholder would like the system to allow them to log in to their account if they have already created one.

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
11.1.1	Entering correct username and password	String 'username123' String 'password123' **VALID DATA**	App will verify their account details and take the user to the main menu.	Ensures that the user can log back into the system at a different time using their already existing log-in details.	**The program successfully allows the user to log into their already existing account.**
11.1.2	Entering incorrect password.	String 'username123' String 'password1' **INVALID DATA**	An error message will appear at the bottom of the screen alerting the user to the problem	Ensures the app is secure by checking whether the system detects when the user has entered a password that is not associated with the entered username.	**The program does not allow the user to enter their account as password is not correct, and an error message appears alerting the user to the issue.**

11.1.3	Entering a username that does not exist.	String ‘username1’ String ‘password123’ **INVALID DATA**	An error message will appear at the bottom of the screen alerting the user to the problem. The error message should be the same as the error message for 11.1.2.	Ensures the app recognises when a username that does not exist has been entered.	**The program does not allow the user to enter their account as the username is not correct, and an error message appears alerting the user to the issue.**
11.1.4	Entering random symbols/integers	String ‘!”£$%^&*()’ Integer 123456789	An error message will appear on screen telling the user that these inputs are invalid.	This is to test for robustness – checking whatever is inputted does not crash the program.	**The program successfully displays the error message on screen saying that the inputs are invalid.**

I have added test 11.1.3 to ensure that both elements of the log in verification work – the password and username verification. I have also added test 11.1.4 to check for extra robustness within this feature – ensuring that the program does not crash even when the inputs consists of random symbols/integers.

As my solution has passed all tests I have conducted on the log in feature, success criteria 2 has been fully met. The program allows the user to enter their account successfully when the correct details are entered, and when they are not, an error message is displayed on screen. Again, this element of my program is robust as it does not crash no matter what the user inputs are, as shown by test 11.1.4.

Success Criteria 3: The stakeholder would like the system to allow them to create their own sets of flashcards.

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
13.1.1	Entering name of set into textual input boxes	String ‘Data Structures’ **VALID DATA**	Name of set will be written to the users personal .csv file, and the relevant button on the choose set layout will be updated to show the new set name.	Ensures the user can create a new set of flashcards under whatever name they want, and that this name is written to the users .csv file and the buttons on the ‘MYSETS’ menu are updated to include this new set.	**The program successfully writes the entered set name to the users csv file and takes the user to the add flashcards layout so they can begin adding flashcards to the set.**
13.1.2	Attempting to submit empty input box	String ‘’ **INVALID DATA**	An error message should appear telling the user that they must input something to be able to create a set with that given name.	This is to test for additional robustness within this feature – it would be wrong for the user to be able to create a set with no name.	**The program successfully tells the user that they must enter a set name to be able to create a set.**
13.1.3	Attempting to create a set once the user has	String ‘LMC’ **VALID DATA**	An error message should appear telling the user that they have reached their limit of 5 sets,	This again tests for robustness by ensuring the user does not go over the 5 set limit, because if	**The program informs the user that they have reached their set limit**

	reached the set limit of 5.		and so cannot create another set.	they did the new set they create would not be visible and would cause issues in the solution.	
13.1.4	Deleting a set	Mouse click	The set should be deleted – the set name, and all of the flashcards associated with the set should be removed from the users CSV file.	Due to the set limit, it is important users can delete sets once they are finished with them to free up a slot.	**The program does not allow the user to delete a set as this feature has not been implemented.**

I have added test 13.1.2 to test for extra robustness within this feature as the user should not be able to create a set with no name. There is no further tests I can add for robustness as technically the user can create any set name of any length/contents as long as they enter something in the set name textual input field.

As this feature passed most of the tests I have conducted on it, I have mostly met, but not fully met success criteria 3. The user is able to create a set with any set name, which is then written to the users CSV file. If the user enters nothing and attempts to create the set, an error message appears alerting the user to this issue. However, the user is not able to completely delete sets of flashcards they have created (once a set has been created with a certain name, the name cannot be changed/removed). This has not been implemented as I did not realise this was a key functionality during the design/development section and I have only realised in the post development testing that this is necessary. Implementing this feature in future development would be relatively simple – the first line of the CSV file would be iterated through to find and delete the set name that the user wants to delete from the CSV file, and then all of the flashcard lines underneath would be iterated through and every time a line in the CSV file is found that contains the relevant set number (as each line in the CSV file takes on the structure 'set number, term, definition'), that whole line would be deleted from the file and the next time a flashcard is added to any set it can fill that slot to ensure that there are no empty spaces (and so extra space being taken up) in the CSV file.

Success Criteria 4: The stakeholder would like the system to view sets of flashcards they have already created.

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
14.1.1	Viewing names of sets of flashcards the user has already created	Clicking the 'My Sets' button from the main menu.	The names of the users sets should be displayed on the buttons on the MYSETS display. If they have not used all 5 of their slots yet, the unnamed sets will show up without a name.	To ensure that the set names displayed are correct and up-to-date.	**The names of all of the sets the user has created are successfully displayed on screen. If they have not used all 5 of their slots yet, the unnamed sets show up without a name.**

I have not added any tests to my original plan here as I did not feel it was necessary – this is only a very small success criteria.

Since I have met the single test here, I have fully met success criteria 4 – the names of the users sets are correctly displayed on screen. However, in future development, the program could be amended to only show the buttons of the sets that the user has created a name for – this stops users being able to effectively edit and revise sets that haven't been created yet (i.e. the sets that don't have a name given to them), as this doesn't make much sense and is an inconsistency with the creating a set process (as creating the set is effectively just naming the set in the current prototype).

Success Criteria 11: The stakeholder would like the system to allow them to edit already existing sets of flashcards.
I have done success criteria 11 before success criteria 5 as it makes more sense to test my program in this order (i.e. testing the user can add to/delete from sets of flashcards before I test whether they can revise from them).

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
16.1.1	Submitting a term and definition to be created as a flashcard via 2 textual inputs and a button press.	String 'Static' String 'Size cannot change during runtime' **VALID DATA**	Term and definition will be written to the relative positions in the users personal .csv file, and the term and definition textual input boxes will be cleared.	Ensures that any term/definition can be entered and written to the .csv file without an error occurring.	**Allows the user to create the flashcard and the term/definition along with the set number they belong to are written to the users unique CSV file.**
16.1.2	Submitting a term and definition to be created as a flashcard via 2 textual inputs and a button press.	String '12*4' String '48' **VALID DATA**	Term and definition will be written to the relative positions in the users personal .csv file, and the term and definition textual input boxes will be cleared.	Ensures that any term/definition can be entered and written to the .csv file without an error occurring.	**Allows the user to create the flashcard and the term/definition along with the set number they belong to are written to the users unique CSV file.**
16.1.3	Deleting flashcard via a button press	Mouse click	The flashcard will be deleted from the users .csv file and the next flashcard procedure is then ran to make the next term appear on the screen.	Ensures that the user can successfully delete a flashcard from a set (by checking whether it has been removed from the users .csv file).	**This has to be done from the revise flashcard screen – but when the delete button is pressed the program successfully removes the term/definition row from the users CSV file.**
16.1.4	Attempting to delete a flashcard when there are no flashcards left in the set.	Mouse click	This should not be possible as the delete flashcard button is on the revise flashcard screen, and when there are no flashcards left in a set the user is automatically took to a screen informing them that	This ensures the user cannot 'delete a flashcard' from a set which has no flashcards, as this would cause an error – it is testing the robustness of my solution.	**The user is taken to a screen telling them that there are no flashcards remaining in the set.**

			there are no flashcards left in the set.		
16.1.5	Attempting to add a flashcard with nothing entered in the term/definition boxes	String ‘’ String ‘’	The user should be notified of when one of the fields has not been filled in.	This ensures the user cannot create a flashcard that does not have a term/definition, and also ensures the program is robust and cannot be crashed in this way.	**An error message is displayed on screen alerting the user to the issue, and nothing is written to the CSV file.**

I have added test 16.1.4 as I feel it is necessary to ensure that the user can only revise a set which has flashcards in it, and once the final flashcard has been deleted, the user is not left looking at an empty screen. This is also a test of robustness to ensure that the program does not crash when this happens. I have also added test 16.1.5 to ensure the user cannot crash the program by attempting to submit a flashcard with no term/definition – this is another test of robustness. There is no limit to the number of flashcards in a set, and so there is no test I need to conduct regarding a maximum number of flashcards in a set.

Since all of the functionality tests have been passed here, I have fully met success criteria 11 – the user can successfully add and delete flashcards to and from a set, and the users CSV file is manipulated accordingly. This feature is also robust, as the user cannot accidently crash the program by trying to submit an empty flashcard/deleting a flashcard from a set with no flashcards.

Success Criteria 5: The stakeholder would like the system to allow them to revise sets of flashcards they have created.

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
15.1.1	Flip flashcard via button press	Mouse click	The flashcard will be flipped (e.g. if term was on the screen, definition will now be on the screen and vice versa)	Ensures that the user can ‘flip’ flashcards (i.e. alternate between the term and definition) via a button press.	**The user can successfully ‘flip’ a flashcard, allowing them to alternate between the term and definition (as this is how flashcards work to help you revise).**
15.1.2	Viewing the next flashcard via a button press	Mouse click	The next flashcard read from the users .csv file will appear on the screen.	Ensures that the next flashcard displayed is from the correct set and is not repeated.	**The next flashcard in the set is successfully displayed.**
15.1.3	Viewing the previous flashcard via a button press	Mouse click	The previous flashcard read from the users .csv file will appear on the screen.	Ensures that when tis button is pressed, the previous flashcard is successfully displayed to the user.	**The previous flashcard in the set is correctly displayed.**
15.1.4	Pressing ‘next flashcard’ when the	Mouse click	An error message should appear on screen telling the user they have	This ensures the solution is robust and does not crash when the user reached the	**An error message appears on screen telling the user they have**

	user has reached the end of the set.		reached the end of the set.	end of the set they are revising.	**reached the end of the set.**
15.1.5	Pressing 'previous flashcard' when the user has reached the start of the set.	Mouse click	An error message should appear on screen telling the user they have reached the start of the set.	This ensures the solution is robust and does not crash when the user reached the start of the set they are revising.	**An error message appears on screen telling the user they have reached the start of the set.**

I have added tests 15.1.4 and 15.1.5 to test for additional robustness within this feature of my solution by ensuring that the user cannot break the program by attempting to go beyond the limits of the set they are revising.

This feature of my solution has passed all of the tests I have conducted on it, so I have fully met success criteria 5. This part of my solution is also very robust as shown in the new tests I have added.

Success Criteria 9 and 10: The stakeholder would like the system to have a clear main menu and should be easy to navigate around/user friendly.

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
12.1.1	Selection of option via 'Sign_up' button	Mouse click	Layout will be updated to 'SIGNUP'	Ensures the user can choose to sign up easily via a button press.	**Correct layout is shown upon button press.**
12.1.2	Selection of option via 'log_in' button	Mouse click	Layout will be updated to 'LOGIN'	Ensures the user can choose to log in easily via a button press.	**Correct layout is shown upon button press.**
12.1.3	Selection of option via 'new_set' button	Mouse click	Layout will be updated to SETNAME, so the user will be taken to a different screen.	Ensures the user can choose to create a new set easily via a button press.	**Correct layout is shown upon button press.**
12.1.4	Selection of option via 'my_sets' button	Mouse click	Layout will be updated to SETS, so the user will be taken to a different screen.	Ensures the user can choose to view their sets easily via a button press.	**Correct layout is shown upon button press.**
12.1.5	Selection of option via 'progress_tracker' button	Mouse click	Layout will be updated to PROGRESS, so the user will be taken to a different screen.	Ensures the user can choose to view their progress easily via a button press.	**Correct layout is shown upon button press. However, nothing is created on this layout as the progress tracker feature has not been implemented yet.**
12.1.6	Selection of option via 'manage_account' button	Mouse click	Layout will be updated to ACCOUNT, so the user will be taken to a different screen.	Ensures the user can choose to manage their account easily via a button press.	**Correct layout is shown upon button press. However, nothing is created on this layout as the manage account feature has not been implemented yet.**

12.1.7	Selection of option via 'log_out' button	Mouse click	User will be logged out and taken back to the start menu (layout is updated to STARTMENU)	Ensures the user can choose to log out easily via a button press.	**Correct layout is shown upon button press and the correct username is shown on the main menu.**
12.1.8	Selection of option via 'Exit' button	Mouse click	App will close as event loop is broken.	Ensures the user can choose to exit the app easily via a button press.	**Exits app.**
12.1.9	General navigation around the app	Mouse clicks	The user should be able to easily navigate around the app and the buttons should all carry out the purpose their labels imply.	This is to ensure that success criteria 10 has been met – ensuring that the program is user friendly.	**The general navigation around the app is simple, easy and user-friendly. All navigation features (buttons) work correctly.**

I have added test 12.1.9 to my original test plan as I feel it is essential to include a clip of general navigation outside of the main menu to show that success criteria 10 has been met.

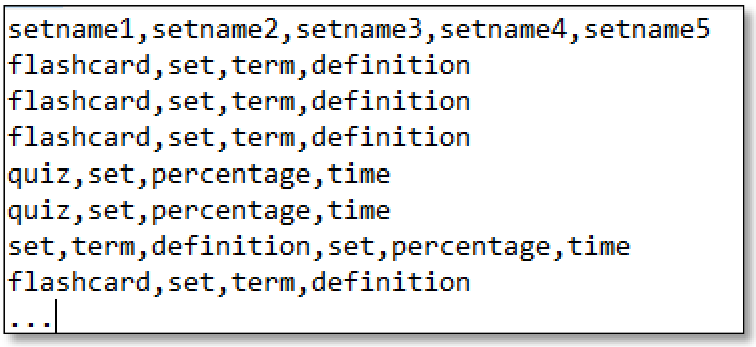
The solution successfully passed all of these tests hence success criteria 9 and 10 have been fully met. I will also consult my stakeholders on the user-friendliness of my solution in the usability testing section later in the evaluation section to gain more feedback from my stakeholders on the general usability and intentional simplicity of my solution.

Success Criteria 6: The stakeholder would like the system to allow them to test their knowledge on a particular set of flashcards.

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
17.1.1	Question and 4 multiple choice answers should appear on screen. User should be able to choose an answer.	Clicking the 'Quiz' button from the set menu. Mouse click.	A question should appear on the screen, with 4 multiple choice answers in the form of buttons. When an answer is chosen, a new question should appear with 4 randomly placed multiple-choice answers. This process will repeat until the quiz is complete, when a summary screen should appear with no statistics on it as the score, percentage and time_taken variables have not been implemented yet.	To ensure that the quiz feature works correctly regardless of which set the user chooses, and that the questions and answers are random and not repeated. To ensure that the quiz ends after all of the questions have been answered.	**The quiz button is present on the set menu and takes the user to a new layout designed to be for the quiz, but the quiz feature has not been implemented yet so nothing is present on this layout.**
17.1.2	Quiz Summary	Visual output when quiz is complete via multiple button presses.	A summary screen should appear with the correct score, percentage and time taken. These should also be written to the users csv file.	To ensure that the correct statistics are displayed on screen when the quiz ends.	**Feature not yet implemented.**

This success criterion has not been met at all yet as it has not been implemented in my partially complete solution due to time constraints. As such, it is not possible for me to conduct any of the post development tests that I had planned for this feature.

To implement this in future development, a quiz feature would need to be added, which displays the term of each flashcard in the chosen set as the 'question' and then the correct answer along with 3 other incorrect answers are shown on the screen in random positions as 4 possible multiple choice answers. The user would have to choose one of these and their final score, percentage and time taken to complete would all be recorded and displayed once they have completed the quiz. Every time a new quiz is taken, the multiple choice answers would be different and appear in different places. This would be achievable by writing all of the terms/definitions from a set to separate lists, displaying the string in the corresponding index position in the definition list for each term randomly in one of the 4 multiple choice slots, and then adding the number of this slot to another list so that another answer cannot be placed in this slot and override the correct answer, and then randomly choosing 3 other definitions from the list and displaying them in the other 3 slots in a similar way. The results would also be written to the users CSV file in the following format:



```
setname1,setname2,setname3,setname4,setname5
flashcard,set,term,definition
flashcard,set,term,definition
flashcard,set,term,definition
quiz,set,percentage,time
quiz,set,percentage,time
set,term,definition,set,percentage,time
flashcard,set,term,definition
...
```

Each row after the set names contains either a flashcard or quiz result, indicated by the first element in that row. I would have to edit the flashcard feature to now check for rows with the string 'flashcard' in the first element as well as searching for the set number. This would allow the results to be displayed on the progress tracker by reading from the file.

Psuedocode for how I originally planned this can be found in the design section.

Success Criteria 7: The stakeholder would like the system to keep track of their progress and show it in a clear and concise way.
Success Criteria 8: The stakeholder would like the system to allow them to share their results and progress on social media.
Success Criteria 13: The stakeholder would like the system to send them email notifications when prompted to.

Test Number	Test Item	Test Data	Expected Result	Justification	Actual Result
18.1.1	Raw numbers representation of users data	Clicking the 'Progress Tracker' button from	The users scores on quizzes should appear on the screen, with the	To ensure that the statistics displayed on screen are correct.	**Button takes user to a separate layout with no contents**

		the main menu.	corresponding set name.		**as this feature has not been implemented.**
18.1.2	Visual representation of users data (e.g. graph)	Clicking the 'Progress Tracker' button from the main menu.	The graph should appear on the left with the correct data plotted.	To ensure that the graph is plotted correctly and is easy for the user to see and deduce their progress from.	**Button takes user to a separate layout with no contents as this feature has not been implemented.**
18.1.3	Choosing a function via a button press (post to Twitter)	Mouse click	A tweet should be sent out containing the users percentage, time taken and set name in a logical sentence.	To ensure that the user can successfully tweet their progress onto the applications Twitter page via a button press.	**Feature not implemented.**
18.1.4	Choosing a function via a button press (email reminder)	Mouse click	An email should be sent to the users email address with the correct subject and message.	To ensure that the user can successfully send themselves an email reminder to their email address stored by the app.	**Feature not implemented.**

Success criteria 7, 8 and 13 have not been met at all as these features have not yet been implemented in my partially complete solution due to time constraints.

If the progress tracker feature were to be implemented in future development, it would require a graph to be plotted of all of the users test scores in each set. There would be a button for each set which would take the user to another screen where the graph of their results (percentages) on the y axis and the attempt number on the x axis. There would be another similar graph of time to complete against attempt number. Attempt number does not need to be stored in the users CSV file as this would be a waste of space as the quiz results are stored in linear order anyway. On this screen, there would also be an email reminder button and a share results button. The email reminder button would send an email from the apps email address to the users email address they entered when signing up using the Gmail API – I would have to do extra research on this as I have never used the Gmail API before. The share results button would share a tweet on the applications twitter page containing the users username, their set name and their 5 most recent scores to illustrate the progress they have been making – this would be done using the Twitter API which I have had experience using before.

Success Criteria 12: The stakeholder would like the system to allow them to set exam dates/deadlines.

I decided not to include this success criteria during the design stage as I realised it would be pretty useless as the app cannot send automated emails/reminders when it is not running, so I decided to focus my time elsewhere. I also consulted my stakeholders on this decision and they agreed that this is not a particularly key feature for the revision app and they would much rather more time be spent improving the revision/quiz features. As such, this success criterion has intentionally not been met at all.

Other features

The manage account feature has also not been implemented yet due to time constraints, and whilst this is only a minor feature I felt it would have been necessary to allow the user to amend their account details either for security reasons (e.g. changing their username/password), or if they have changed their email address etc. To implement this, similar validation would have taken part to as in the sign up feature (hence why I made separate subroutines for the validation procedures), and the userinfo.txt file would have been rewritten to include the new info if the details entered were valid. There would also be an account deletion button (which would ask the user for confirmation of this decision before the account was deleted), and once the decision was confirmed, the userinfo.txt file would have been rewritten without the deleted account details.

Summary of success criteria

Success Criterion	Successfully Created? (partially/fully/not)	Why?	How to fully meet this criterion.
The stakeholder would like the system to allow them to create an account if they don't already have one.	Fully	User can successfully create an account with valid details, and inputs are always validated by the program.	N/A
The stakeholder would like the system to allow them to log in to their account if they have already created one.	Fully	User can successfully login to an already existing account, and incorrect usernames/passwords are always flagged up.	N/A
The stakeholder would like the system to allow them to create their own sets of flashcards.	Partially	User can create a set with any given name, which is then written to the users CSV file. However, they cannot delete sets once they are created.	Add a way to delete sets (i.e. delete a given set name and all of the flashcards associated with that set from the users CSV file.

The stakeholder would like the system to view sets of flashcards they have already created.	Fully	User can view all of their sets on the MySets layout.	N/A
The stakeholder would like the system to allow them to revise sets of flashcards they have created.	Fully	User can successfully navigate their way through a set of flashcards and flip the flashcards. This feature is robust by now allowing the user to go beyond the set limits.	N/A
The stakeholder would like the system to allow them to test their knowledge on a particular set of flashcards.	Not	The quiz feature has not been implemented.	Add in a quiz feature which displays all of the terms from a given set along with 4 random multiple choice answers, one of which will be the correct answer. Record the users percentage and time taken and write these to their CSV file.
The stakeholder would like the system to keep track of their progress and show it in a clear and concise way.	Not	Progress tracker has not been implemented.	Add in a progress tracker which reads the users quiz results from their CSV file and displays them graphically on the screen for every set.
The stakeholder would like the system to allow them to share their results and progress on social media.	Not	Social media sharing has not yet been implemented.	Add in a button on the progress tracker which posts a tweet to the apps Twitter page displaying the users username along with their results for a given set.

The stakeholder would like the system to have a clear main menu.	Fully	The main menu is simple and easy to navigate through. The buttons all clearly display their purpose and leave no ambiguity.	N/A
The stakeholder would like the system to be user-friendly and easy to understand.	Fully	The program is very easy to navigate through via buttons. All buttons clearly display their purpose.	N/A
The stakeholder would like the system to allow them to edit already existing sets of flashcards.	Fully	Users can add flashcards to a set via the 'add flashcards' button, and delete flashcards from a set on the revision screen.	N/A
The stakeholder would like the system to allow them to set exam dates/deadlines.	Not	This feature has not been implemented.	I have decided not to implement this feature as it would be pointless as the app cannot send reminders whilst it is not running (as it is a locally ran app).
The stakeholder would like the system to send them email notifications when prompted to.	Not	This feature has not been implemented.	Emails could be sent from the apps email address (Gmail) to the users email address (Gmail) they entered when signing up via a button press. This could be done using the Google API.

Usability features

In order to test the usability features of my program, I have contacted all three of my stakeholders and given them my solution to test and use. I have asked each of them to fill out the questionnaire I set out in the analysis section. There is also a recording of one of my stakeholders, Marcus, using the program and its features in the post-development testing evidence PowerPoint.

Questionnaire 1: Sign-up/Log-in

This questionnaire relates to success criteria 1 and 2.

1.) Can you successfully create an account on the app?
Vivek: *Yes*
Marcus: *Yes, but I did not have a Gmail account.*
Sam: *Yes*

These responses show that the sign up feature works and the users details are written to a file if they are valid.

2.) Can you successfully log back into this account using the same details?
Vivek: *Yes*
Marcus: *Yes*
Sam: *Yes*

These responses show that the log in feature works and that the details were correctly written to the file when the user created their account and are also successfully read from the file and validated when the user attempts to log in.

3.) Was this an easy process to understand and carry out?
Vivek: *Yes, the process is very similar/identical to other apps/sites where you have to create an account.*
Marcus: *Yes*
Sam: *Yes*

These responses show that the process of creating an account/logging in is simple as intended and so the user will not need to spend lots of time figuring out how to do it.

4.) Do error messages appear when incorrect details are entered?
Vivek: *Yes*
Marcus: *Yes, although it would be more practical if more than just Gmail accounts were allowed to be used as I do not have a Gmail account.*
Sam: *Yes*

These responses show that this element of the program is robust as the validation works as intended.

Questionnaire 1 shows that the signup/login feature works perfectly, the stakeholders were happy with the functionality, validation and simplicity of the process. When I asked my stakeholders whether the passwords being hashed before they are stored to the text file made them feel more secure about their details, all of them responded 'yes'. As such, the ability for the user to create an account or login to an already existing account has been a success.

Questionnaire 2: Creating/editing/revising a set of flashcards
This questionnaire relates to success criteria 3, 5 and 11.

1.) Can you successfully create a set of flashcards with a given name?

Vivek: *Yes but not beyond 5 sets.*
Marcus: *I can only create 5 sets before it says there is a set limit and then theres no option to delete the sets so I'm stuck with the same set names forever once they are created.*
Sam: *Yes, but the 5 set limit is annoying.*

It is clear from this feedback that the users did find the 5 set limit annoying and inconvenient, so in future development I would remove this limit and whilst this would potentially increase the storage requirements of my solution, the benefits to the stakeholders here seem to outweigh this. To do this, I would have to store a value in the users CSV file that indicates how many sets they have created so that the program would know which index position in the first row to insert a new set name to. However, this could also create an issue with the buttons on the 'MySets' layout as there is only 5 buttons but the user would now have the ability to create more than 5 sets. I am not immediately sure how to get around this issue but it could involve having a 'next page' button which updates the 5 buttons to the names of the next 5 sets the user has created.

2.) Can you then add flashcards to this set?
Vivek: *Yes, and the amount I can add is seemingly unlimited which is convenient.*
Marcus: *Yes*
Sam: *Yes*

All stakeholders were happy with the ability to add flashcards to a set they have created and were pleased with the unlimited nature of this.

3.) Can you remove flashcards?
Vivek: *Yes*
Marcus: *Yes*
Sam: *Yes, but it would be better if there was a confirmation screen that ensures the user definitely wants to delete that flashcard in case the delete button is accidently pressed.*

Sam has raised a very important point here. I agree it would be beneficial to include a confirmation screen in case the delete button is accidently pressed, especially considering that it is located right next to the next, previous and flip buttons. This would ensure that the user does not have to waste valuable revision time going back and recreating flashcards they have accidently deleted. This could be implemented in future development by simply updating the error message whenever the delete button is pressed to display a message telling the user to press the delete button again if they wish to delete that flashcard. If they press the delete button again the flashcard will be deleted, and if they do not, the confirmation message will disappear and the flashcard will not be deleted.

4.) Can you 'flip' flashcards for revision purposes?
Vivek: *Yes*
Marcus: *Yes*
Sam: *Yes, however sometimes the text goes slightly off the screen when it is a longer term/definition.*

Sam could have experienced this issue because of the border I placed on the flashcard textbox in order to make it look more aesthetically pleasing. In future development, this border could easily be made smaller or even entirely removed to get around this issue if it becomes too much of an issue.

5.) Is the process easy to understand and carry out?
Vivek: *Yes, very simple and effective.*
Marcus: *Yes similar to real-life flashcards and websites such as Quizlet.*
Sam: *Yes*

Again, the stakeholders were happy with the simplicity of this feature of the app.

6.) Do you feel the revision is efficient?
Vivek: *Yes even though I do not use flashcards that often*
Marcus: *Yes*
Sam: *Yes*

All stakeholders were happy with the way the revision on the app worked even if flashcards were not their preferred method of revision.

7.) I have also decided to add a question regarding the robustness of this feature (i.e. can you break the program by deleting all of the flashcards in a set or attempting to go to the 'next flashcard' at the end of the set or vice versa? Etc.)
Vivek: *I have tried to break the program by deleting all of the flashcards and attempting to go back and revise the set but a screen is shown telling me there is no flashcards in the set. The feature seems very robust.*
Marcus: *Very robust*
Sam: *Works perfectly and is unbreakable.*

The stakeholders have confirmed that the revision feature is robust and errors will not interrupt a users revision.

The responses to questionnaire 2 show that the process of creating revision material and revising it is very efficient, simple and robust. The few complaints received could be sorted in future development, with the biggest issue being the 5 set limit. However, overall the stakeholders were very happy with the functionality and usability of this feature so this element of my solution has been a success.

Questionnaire 3: Quizzes
This questionnaire relates to success criteria 6.

1.) Does the program allow you to take a quiz on a set of flashcards you have created?
2.) Was the quiz thorough enough?
3.) Were the questions in a random order?
4.) Did the multiple choice/written quiz work correctly and were you given a valid score, percentage and time taken upon completion?

As explained earlier in the report, the quiz feature has not been implemented and so I will not be giving this questionnaire to my stakeholders. They all said it would be massively beneficial for this feature to be implemented in future development as testing your knowledge is an important part of revision. As such, the quiz feature of my solution has been a failure. The ways this would be implemented in future development were outlined earlier in the evaluation section.

Questionnaire 4: Progress Tracker
This questionnaire relates to success criteria 7, 8 and 13.

1.) Can you successfully access and view the progress tracker feature?
2.) Are the statistics shown correct?
3.) Do you feel this feature is helping you?
4.) Can you successfully set exam dates/deadlines, and do these show on the main menu?
5.) Are email notifications successfully sent and received when you prompt the program to send one?
6.) Can you successfully share your results on the application's Twitter page?

Similar to questionnaire 3, the progress tracker feature has not been implemented as explained earlier in the report so it would be pointless giving this questionnaire to my stakeholders. I did however ask whether they thought it would be beneficial to implement this feature in future development and they all said yes. As such, the progress tracker feature of my solution has been a failure. The ways this would be implemented in future development were outlined earlier in the evaluation section.

Questionnaire 5: General
This questionnaire relates to success criteria 4, 9 and 10.

1.) Is the app easy to use and navigate?
Vivek: *Yes, the functionality of the buttons is clear and they work as intended.*
Marcus: *Yes, although I feel it would be beneficial to have a 'how to use' page on the start menu informing users of the features of the app, as well as things like the 5 set limit.*
Sam: *Yes, navigation is easy.*

In future development, a 'how to use' feature could easily be added as Marcus as suggested. I feel this would be beneficial to ensure that users know about how to use the app, its features and its limitations before creating an account. It could be added by simply adding a button on the start menu which takes the user to a new layout containing a block of text explaining these things, along with a back button to take the user back to the start menu.

2.) Is the GUI layout simple and not confusing?
Vivek: *Yes, although it would be better if the GUI was more modern as it looks slightly outdated.*
Marcus: *Yes, usability is easy.*
Sam: *Yes, easy to use but looks a bit old.*

Vivek and Sam raised the point of making the layout more modern and more like other revision apps on the market at the moment. I agree with this, and the reason more time was not dedicated to this during development is because I decided to focus more of my time on the functionality once the GUI was sufficient for use. Making the GUI more modern in PySimpleGUI is possible, but difficult. It could be easier to use a different framework to make the GUI look more modern in future development.

3.) Are there any errors/problems you encountered whilst using the app? If so, what were these?
Vivek: *None apart from the 5 set limit, which is more of an inconvenience rather than an error.*
Marcus: *Its not an error with the program but it is annoying that only Gmail accounts are allowed.*
Sam: *No errors.*

From the stakeholder feedback, it is clear that there are very little to no 'errors' in the program, rather just intentional features such as the 5 set limit that some stakeholders find inconvenient.

4.) Did the sound effects work correctly?
Vivek: *There were no sound effects.*
Marcus: *No*
Sam: *I did not hear any sound effects*

When programming my solution, I did not include any sound effects so I did not expect the user to hear anything while using the app. This could be easily implemented using mixer in future development, but it is not a key feature of the app.

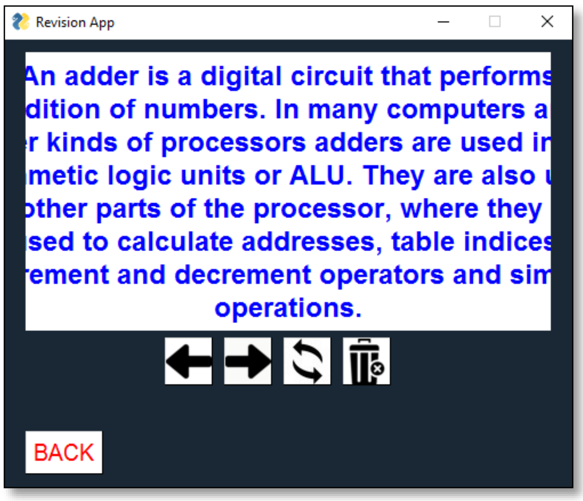
5.) Is the colour scheme easy on the eye?
Vivek: *Yes, the colour scheme is fine.*
Marcus: *Yes*
Sam: *Yes*

All stakeholders were happy with the colour scheme used and did not think it was too overwhelming and would be suitable for long runs of revision looking at the screen.

6.) I added a new question to my original questionnaire asking the stakeholders whether they had anything to add/any advice for future development:
Vivek: *The app is good and the features that have been implemented work correctly, but the solution would be massively improved by adding a progress tracker and quiz feature in future development.*
Marcus: *The only advice I have is the feedback I have given to previous questions. Apart from that, the app is good. I also noticed when testing the program that when a new set has been made, and you are taken to add flashcards to the set, when you press back to go to the MySets layout, the new set has not appeared on the button – it only appears when you go back to the main menu and then back onto the MySets layout. Longer definitions also appear to go significantly off the screen on the flashcard screen.*

Sam: *A more modern interface and the features that have not yet been implemented would make the solution better. A way to edit my details once I have create my account would also be beneficial.*

The responses here show that apart from the concerns mentioned earlier, the stakeholders are mostly happy with the solution. However, when testing the program (the evidence video for which can be found in the post-development testing evidence PowerPoint), Marcus noticed that the set names were not being updated on the MySets layout when the user goes back to it from the flashcard screen when they have made a new set. This means that success criteria 4 has only been partially met. This could easily be solved in future development by running the set_names function whenever a new set is made. He also noticed that longer definitions are significantly cut off the screen when revising flashcards. This again could be because the text element has been made too large or the border is obscuring some of the text. Both of these things could be investigated in future development. However, I do not think it is because of the text element being too large because when I use a large piece of text as the definition, it does go onto a new line so it appears that the border is the issue:



An adder is a digital circuit that performs
dition of numbers. In many computers a
r kinds of processors adders are used in
metic logic units or ALU. They are also u
other parts of the processor, where they
sed to calculate addresses, table indices
ement and decrement operators and sim
operations.

This screenshot shows that either the textbox is slightly too large or the border is obscuring the text.

Sam wanted a way to edit your details after account creation, which links to the ‘manage account’ feature that I mentioned earlier in the evaluation section that I was unable to implement for reasons listed earlier. This is a feature I would definitely implement in future development.

Questionnaire 5 has shown that the users are happy with the functionality of the program, but think it would be improved by making the interface more modern and implementing the progress tracker/quiz features.

Overall, the stakeholder feedback here has outlined that the features that have been implemented mostly work well apart from a couple of small, easily fixable issues. The general usability and user-friendliness of my solution has been a success.

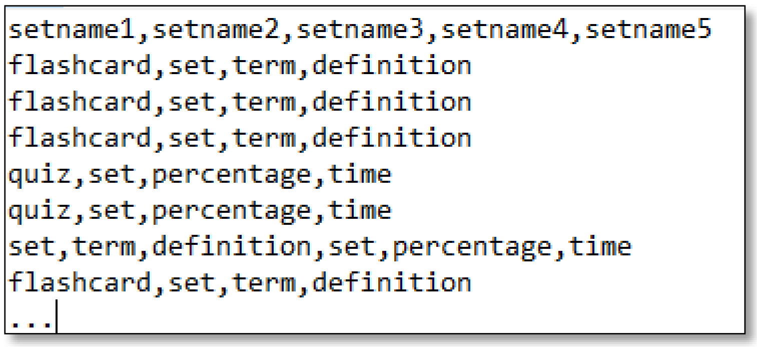
Summary of usability features

Usability Feature	**Successfully Created? (partial/fully/not)**	**Why?**	**How to fully meet this feature.**
Instructions on use	Not	There is nowhere in the program where the user is given any instructions on how to use the program or its features.	Add a button on the start menu taking the user to a screen containing instructions for use as well as the features of the app.
Buttons	Fully	All buttons have a clear functionality, work as intended and are big enough for the user to easily see, making the interface much easier to navigate.	N/A
GUI	Partially	Users said the GUI was simple and easy to navigate through and understand, but looked outdated and would be better if it was more modern like many already existing revision apps.	Use images on buttons to make the layout look more modernised, use a different colour scheme, or even use an entirely different framework for the GUI.
Window size	Partially	The window size is good and is fixed so it appears the same on all devices, but the window itself cannot be reshaped, so on desktops the layout only covers a small portion of the screen and so may appear a bit small.	Make device-dependent layouts rather than a fixed layout.

Limitations of my solution and how they could be addressed

One limitation of my partially complete solution is the 5 set limit. This is a limitation because it effectively prevents the user from using the app for a wide range of subjects as it limits them to 5 sets of flashcards. This was an issue mentioned by all my stakeholders as something they would like to be removed in future development. The main issue with having more than 5 sets is not a storage related issue, but rather the fact that the set menu only has 5 buttons, and so can only display 5 sets for the user to interact with. There is no way I am aware of in PySimpleGUI to automatically create buttons and their functionality whilst the program is running (the layouts have to be predetermined), so I could work around this by having a 'next page'/'previous page' button on the MySets layout which updates the buttons on the screen to display the next/previous 5 sets, effectively allowing for unlimited sets to be displayed. There would also be robustness features added to this to ensure that the user cannot press 'next page'/'previous page' when there are no more sets to be displayed.

Another limitation of my solution is the lack of ability for the user to test their own knowledge on a set of flashcards (i.e. the quiz feature). This is a limitation because testing your knowledge is a key part of revision to check whether you fully understand and remember what you have revised. My stakeholders also agree that not having this feature is a big limitation to the solution. To address this in future development, as laid out earlier in the report, a quiz feature would need to be added, which displays the term of each flashcard in the chosen set as the 'question' and then the correct answer along with 3 other incorrect answers are shown on the screen in random positions as 4 possible multiple choice answers. The user would have to choose one of these and their final score, percentage and time taken to complete would all be recorded and displayed once they have completed the quiz. Every time a new quiz is taken, the multiple choice answers would be different and appear in different places. This would be achievable by writing all of the terms/definitions from a set to separate lists, displaying the string in the corresponding index position in the definition list for each term randomly in one of the 4 multiple choice slots, and then adding the number of this slot to another list so that another answer cannot be placed in this slot and override the correct answer, and then randomly choosing 3 other definitions from the list and displaying them in the other 3 slots in a similar way. However, this would require the layout of the CSV file to change so that it now includes spaces for the quiz results to be stored, and there is a way to differentiate between them and the flashcards themselves in the CSV file:



```
setname1,setname2,setname3,setname4,setname5
flashcard,set,term,definition
flashcard,set,term,definition
flashcard,set,term,definition
quiz,set,percentage,time
quiz,set,percentage,time
set,term,definition,set,percentage,time
flashcard,set,term,definition
...
```

Each row after the set names contains either a flashcard or quiz result, indicated by the first element in that row. I would have to edit the flashcard feature to now check for rows with the string 'flashcard' in the first element as well as searching for the set number.

Another limitation to my solution is the fact that the progress tracker has not been implemented either due to time constraints. This is a limitation because tracking your progress is key when doing revision to see where you need to focus your time and which subjects you are not making as much progress in. To implement this in future development, as mentioned earlier in the evaluation section, there would have to be graphs displayed on the screen which display the percentages and time taken to complete quizzes on a particular set (I would have to do extra research on this as graphing is not a concept I am familiar with in python). The data for the graphs would be read from the users CSV file.

Another limitation of my solution is the fact that the program only allows Gmail accounts, as mentioned by one of my stakeholders. This is a limitation because it prevents people who do not have a Gmail account from creating an account on the app with their email address, as an error message would appear telling them that their email address is not valid (as it is not Gmail). This could be changed by removing the gmail validation from the sign up feature and sending emails from the apps Gmail account to non-gmail addresses.

Finally, the last limitation of my solution is the fact that the user cannot change their account details/delete their account (which creates security issues as users cannot change their passwords or email addresses) as the manage account feature has not been implemented. This could be solved in future development by adding a screen that allows a user to enter their current password (for added security), and if it is correct, allows them to change the password/email address to something else. The userinfo.txt file would then be rewritten with this new data in place of the old data. Adding a delete account feature would be relatively simple – it would just require the users CSV file to be deleted, and all of their information from the userinfo.txt file to be removed (most likely by rewriting the file without that information).

Maintenance issues

For users to be able to access their accounts on any device, the userinfo.txt file would have to be constantly updated, as well as the users CSV file. This maintenance issue could be solved by storing the users details in a server that uses a database that can save all data. Then use a python SQL library to get data from this database. I would have to do further research on finding a python library that handle server traffic in order to choose a suitable one and learn how to use this library before implementing this change.

In the long term, if the app ever became large-scale, I would have to hire a maintenance team to deal with the user feedback/support and further updates. I would also have to implement a user support feature in the app if it were to be rolled out on a large scale so there is a clear way to ask for support so that the support team can handle the request. This could be expensive and time-consuming to interview and hire a support team, but it would be necessary as it would be impossible for mee to single-handedly run all aspects of the app if it was being used on a large scale.

Furthermore, any updates made to the app in the future would require users to re-download the app, and if any changes were made to the way the text/CSV files/databases were laid out, the already existing files/databases would have to be entirely reconfigured to match the way they are being read. This would be very time consuming and meticulous, and emphasises the need for a large support team.

End of report summary

Now that I have reached the end of my NEA report and I have analysed, designed, developed and evaluated my solution, the app can be rolled out for my stakeholders to use.

The elements of my solution that were implemented mostly followed how I planned their design, and the elements that were not implemented were justified and evaluated in terms of their importance and how they could be implemented in future development.

Any long-term maintenance issues have been highlighted and I have offered suggestions to how these could be solved or how the solution could be amended to deal with them.