A Level Programming Project By Mayur Shankar

H446-03

Contents:

Analysis:

- Overview of the project
- Description of stakeholders
- Justification of computational methods
  - Thinking Abstractly and Visualisation
  - Thinking Ahead
  - Thinking Procedurally and decomposition
  - Thinking Logically
  - Thinking Concurrently
- Primary Research:
  - Questionnaire and Feedback
- Secondary Research:
  - Focus Education Learning
  - Memrise
- Hardware Requirements
- Software Requirements
- Features of my proposed solution
- Success Criteria

Design:

- Structure of the solution
- Key Variables and Structure
- Screen Designs
- Algorithms
- Testing for development
- Final test data

Implementation:

- Login screen development
- Practical animation development
- Quiz screen development

Evaluation:

- Post-development testing
- Meeting the requirements

Code Listings

Bibliography

ANALYSIS:

Overview of the project:

My program is aimed at helping Physics students with their practical skills which are required in their exams. Students will log into the program using their credentials and will be able to choose a required Physics practical to view. They will be able to view a labelled, detailed animation of their chosen practical as many times as they want. Once finished, they can answer questions about the viewed practical to test their knowledge. The answers will be marked, a score will be given and a graph will be produced to help the student gauge their performance.

Stakeholders:

This program will be mainly targeted at high school or college students as a valuable learning tool as my program will contain animations of practicals that are required by the exam.

Currently, these practicals can only be performed in school labs. Since they can only be performed once or twice and sometimes do not work as intended, it can be difficult for students to understand the process. To compensate, often in textbooks, there are printed diagrams however I feel that they are not useful in a student's understanding.

My solution will involve animations of practicals that will be much more informative. A student can use this to cover any questions they have, either after having performed the actual experiment, or to revise or to teach themselves without any prior knowledge. Since students will be able to test themselves, this program will be able to reinforce practical knowledge more effectively than current paper methods. Another reason my solution is appropriate to students is that my program does not need access to a lab so students are able to learn wherever they want. This is particularly useful in schools that may not be fortunate enough to have the correct or enough resources for students, or even in light of this current Covid-19 pandemic.

I have chosen Physics students Joe and Abbas to personify my target audience. They both have learned Physics from years 7-12 and have had plenty of experience with performing school practicals and therefore this will help me create a program that is appropriate to the end-user. Since I share my Physics class with Joe and Abbas, I will have regular contact with them.

JUSTIFICATION OF COMPUTATIONAL METHODS

Thinking Abstractly and Visualisation:

In general, my program is an abstraction of Physics practicals and digital quizzes. These abstractions allow me to take the best parts of what current methods teach therefore ensuring my program is relevant and helpful to a student.

- General information about the student will not be needed in order to keep the program efficient. All that will be needed is a username, a password and a current date so the student's progress can be saved and tracked over time.
- Since my program will be an abstraction of Physics practicals that are performed live in front of students, to keep it relevant to the real-world, it will be important to add items such as the apparatus required. However, I can ignore any discrepancies caused by the physical environment or any human error to keep the practical streamlined. This is because it will be hard to simulate these factors through a computer program. Though, students will still be made aware of these potential errors as required by the exam.
- The program will need to ask questions which are based on the Physics subject where I will only need to focus on topics that relate to the practical section. Abstraction is important as I will need to include information that is necessary for a student to learn. Adding too much or unnecessary information - such as other Physics topics - could overwhelm the student.

Thinking Ahead:

For the program to work as intended, I have included what should be inputted and outputted.

- My program will have an account-based system. Therefore, a username and password will need to be inputted to secure the account. This will be stored in a list or database. Also, a date can be inputted when the student logs in. This is so the student's progress can be tracked over time.
- Additionally, the student should have the ability to either create a new account or change their username or password if they desire to.
- The animation may contain various buttons for a user to press to simulate the interaction they would have in real-life when performing the experiment.
- Sound effects could be played if a user presses a button.
- The program will need to store a list of questions for the integrated quiz to work.
- The user will need to input their answers for the questions they are asked, using a keyboard and mouse. This could either be multiple-choice questions comprised of buttons or a text box where students can type in a longer answer.
- A score will be given to the student which will then be stored alongside the student's details and the date which they inputted.
- A list of the student's past and current scores can be used to output a graph of the student's progress over time.

Thinking Procedurally & Decomposition:

To explain my program, I have broken it down into specific parts.

Student login:

- The student will be able to input their credentials or can create a new account to log in with.

Practical animation:

- The student can choose from a list of Physics practicals.
- A new window will be opened, containing an animation with buttons a user can press to interact with it. For example, the user can choose to repeat the animation or exit it and return to the login screen.

Integrated Quiz:

- When the student is ready, a timer starts and questions are displayed one after each other. The user will input their answer.
- Once the timer finishes or the user finishes the quiz, the answers are marked against a list of correct answers and a score is outputted and then stored.

Graphing:

- A graph of the student's score against the date which they completed the quiz is plotted on Matplotlib, displaying progress.
- The student can exit or restart the program.

Mechanical:

- The input system must cope with anything the user may enter. A validation system could be implemented in the login system if a wrong data type is entered.
- Use of files to correctly store the student's details and questions. This is vital as lots of information will need to be written to and read from these files.
- The timer system must be robust. There could be a pause button if a student cannot continue for any reason.
- Sound effects or music during the animation will have to be implemented. This also needs to be turned off if a user does not want it.

Thinking Logically:

This is where I can include branching or decision points.

- A validation system can ask for the correct credentials repeatedly until they are entered.
- The student can choose to repeat the animation as many times as they want.
- The program will be checking when the timer reaches zero as the student answers questions.

Thinking Concurrently:

- The program will have to check whether the timer has finished and if the student has finished the questions at the same time to determine if they have completed the quiz in the allocated time.
- The program will have to write the student's answer to a file while preparing to display the next question.

RESEARCH

Primary Research:

Below I have listed questions that I feel are important to be asked to my stakeholder and these are their answers:

1. *Do you think more importance should be placed on the quality of the animation of the practical or a more thorough bank of questions to be tested on? Or would a balance of both be ideal?*

   Joe: "While I do think a range of questions is important, it is much easier for students to find practice questions online than quality animations so I would say the animation is more important."

   Abbas: "A 75% bank of questions and 25% animation focus would be good. This is because the theory questions are important as they come up the most on exams, but there are many 6 markers that focus on the set up of the practical visually too."

2. *Would there be times when you would just want to view the animation and not do the quiz?*

   Joe: "Yes, but there could be an option to choose how long the quiz should be"

   Abbas: "No, the quiz should be complementary to the practical."

3. *Approximately, how many questions would you like there to be?*

   ☐ *1-5* ☐ *6-10* ☐ *11-15* ☐ *More than 16*

   Joe: "11-15"

   Abbas: "More than 16. If done thoroughly, this app should be supplied with many questions that one can rely upon to get the utmost practice."

4. *What style of questions would you prefer – multiple-choice questions or ones that require written answers or a mixture of both?*

   Joe: "I think a mixture of both would be good."

   Abbas: "50% multiple-choice, 50% written. This is because both are quite good, multiple-choice is engaging and written is better for memory."

5. *Do you think a timer would be useful when answering to maintain concentration? Could you explain why or why not?*

   Joe: "For me personally, a timer counting down would just stress me out but a stopwatch counting upwards might make me more conscious of the time I'm wasting and compel me to focus more."

   Abbas: "Yes, but not straight away. It should start relaxed and have an option for a timer, then one can get more comfortable in the app then time themselves to maintain concentration."

6. *If you are stuck when answering a question, would hints be useful? Could you explain why or why not?*

Joe: “Yeah, I think hints would be useful. It's annoying when you have a question where you have to explain why something occurs and you know the answer, but word it wrong or don't hit all the mark points.”

Abbas: “Yes, hints would be useful at the start. It would help trigger memory but an option could be put on to turn off hints to make it harder. The only reason hints would be useful is for one to be comfortable when beginning revision.”

7. *When answering questions, would you want to see whether you got it correct or incorrect straight away or wait until you've answered all of them?*

   Joe: “Yes please, I think instant feedback would encourage me to improve my answers straight away because sometimes if I'm revising and leave all the checking till the end, it becomes a chore. Breaking that up makes the experience more enjoyable.”

   Abbas: “Straight away, then possibly bank the wrong questions for a re-attempt on those wrong answers”

8. *Is it worth having explanations of where you went wrong if you got a question incorrect?*

   Joe: “Explanations or model answers would be really useful.”

   Abbas: “Yes.”

9. *For the animation, would you rather just watch it or interact with it using buttons and sliders?*

   Joe: “Yeah I want buttons and sliders! I definitely want to interact with the animation.”

   Abbas: “Interact with the animation and then just watch the replay if possible. If that's not possible then just the interaction will suffice.”

10. *Would sound effects and/or music be something you would like to have, particularly for the animation and quiz?*

    Joe: “I don't think it's essential. I wouldn't like to have music on while I'm answering questions but some sound effects during the animations might be nice. Just as long as there's a mute button so that people can revise with Spotify on in the background.”

    Abbas: “Sound effects only, music would be off-putting in my opinion.”

11. *How important would you say it is to be able to change your login credentials – username and password.*

    Joe: “I'm not all that fussed about changing my details just as long as I have a way to remember my password or something.”

    Abbas: “Changing username is not particularly important compared to changing the password.”

12. *Would you want the ability to have your previous scores saved and be able to view them?*

    Joe: “Yes please, that would make me want to try and compete against myself.”

    Abbas: “Yes, would be useful to see progress.”

13. *Would you suggest any feature that would make this app more useful to your learning – for example, more helpful to the lessons you normally do with your teacher?*

Joe: “A league table, such that of Memrise.”

Abbas: “More focus on questions and usage of the app is more useful in my opinion.”

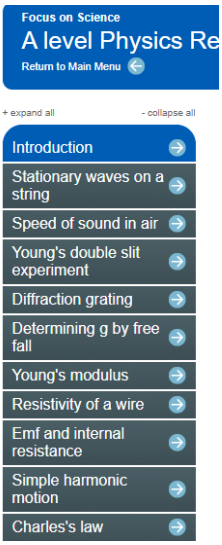
Analysis:

1. *Both my stakeholders had opposing views when it came to whether the quality of the animation or the questions were more important. This shows that different students will have different needs when using the program so therefore I think a balance of both would be ideal.*
2. *Again, there were different views towards this. I think that the default option should be for the program to move straight onto the questions when the animation has finished, but there could be an option if a student wants to only view the animation or wants to do a shorter quiz.*
3. *Based on feedback, a number between 11 and 16 would be ideal. However, a student could have an option to do a shorter quiz based on the analysis above.*
4. *Both stakeholders agreed that a mixture of both would be best. For example, I could implement a keyword-based search when marking the answers as this is what examiners commonly look for when marking exam questions.*
5. *Both stakeholders agreed that some sort of timer would be useful to keep themselves engaged. However, their concern was that a typical countdown timer may have negative results since it could make the quiz too difficult and stressful. Therefore, I will give more flexibility to the user where they could choose their time limit or even have a timer counting upwards, so they are aware of how much time they are taking. This could even be saved alongside the score and date, so the student has another parameter to rate themselves against.*
6. *Both agreed that hints would be useful. An implementation of this could be saving the hints alongside the question and displaying the hint when a button is pressed. There could also be an option to turn off hints to make the quiz more challenging.*
7. *Yes, having the answers instantly displayed after they have answered a question was agreed on.*
8. *Explanations and model answers were considered useful.*
9. *Interacting with the animation was favourable as it would be more engaging.*
10. *Sound effects were considered a nice feature to have so I will implement them in my program as it adds life to it. However, music was seen to be more annoying so I will not include it.*
11. *Changing the username and password was not seen to be important so I will not include this feature in my program.*
12. *Having previous scores saved was thought to be useful, as it would make the student want to compete against themselves to improve. This will also be reflected in the progress graph that will be plotted using these scores.*
13. *Having a league table is a nice feature to implement. I could have a file where I append names of various students with their scores and then display it at the end.*

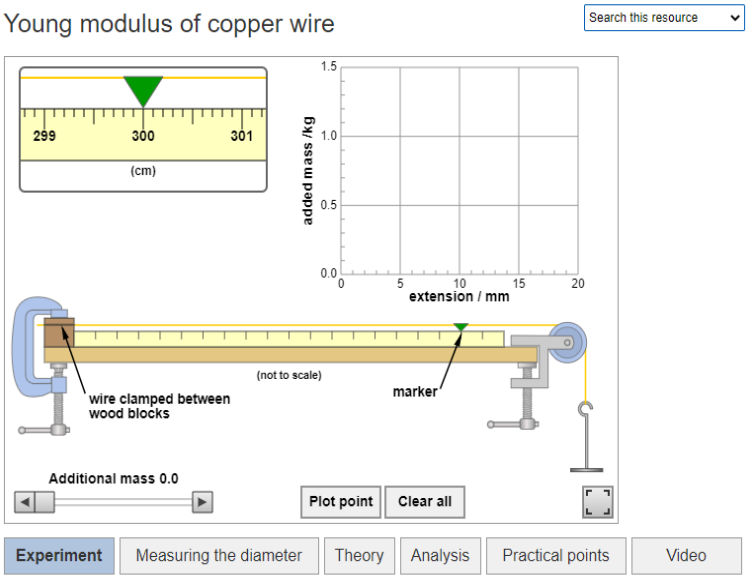
SECONDARY RESEARCH:

Focus Educational Software:

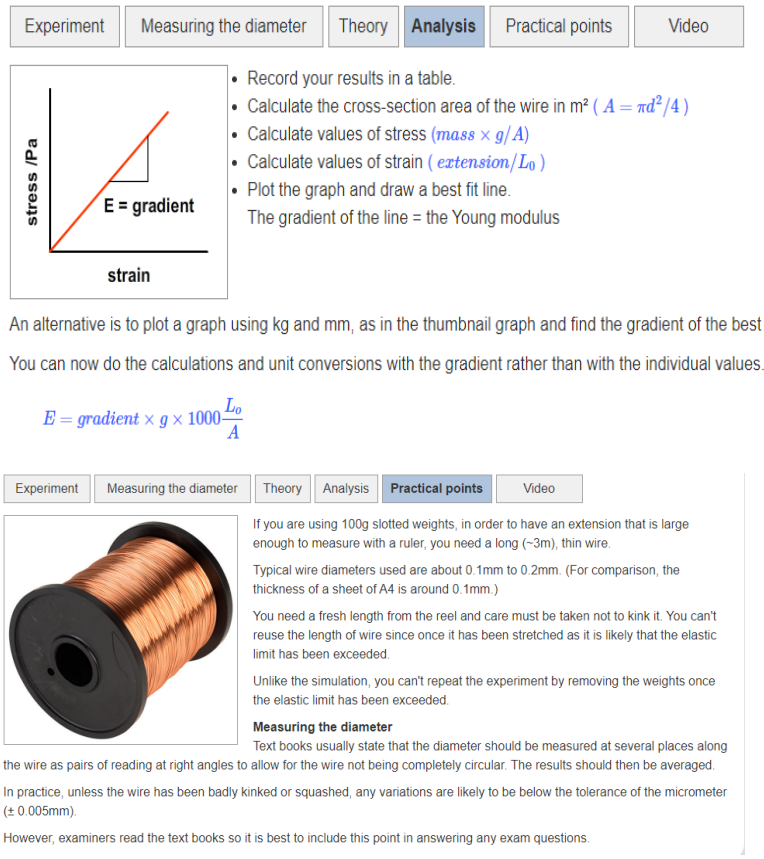
This company offers an online science resource pack which contains animations of science practicals. As it is meant to be paid for and targeted towards school organisations, it will contain a wide range of practicals.



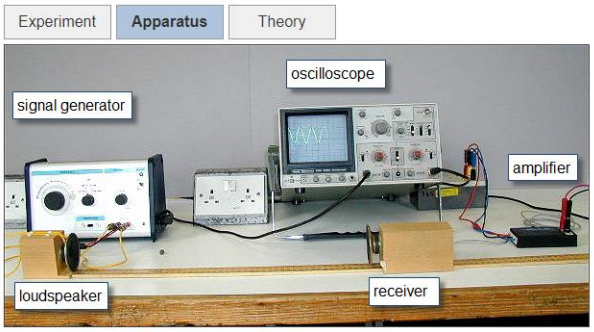
Above, is the start screen for Focus eLearning once the student has logged in. They offer a large variety of animations and learning tools that cater to students across multiple ages and different exam boards. The menu is well laid out and clear for a user to navigate and there is an option to search. The use of pictures is a feature I may implement as it makes the menu more user-friendly.



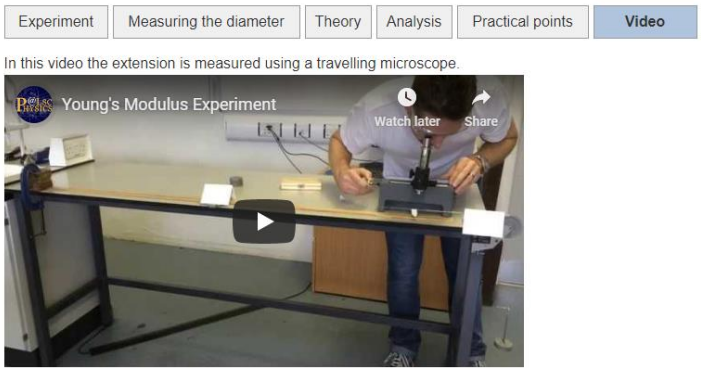
This is an example of a Physics practical animation. A student can change the slider 'additional mass' which corresponds to an action onscreen. Once finished, they can press a button to have a graph of the results drawn. Overall, it is clear and concise and makes it easy for student to follow and learn. For example, there is not too much information being displayed at one time and the use of bold text and labels highlights what students should be aware of – just like a teacher may point things out in a live demonstration. I do like this idea of the interactive buttons and sliders and I want to include this in my program as it was requested in my primary research.



There are a variety of buttons a student can press on. For example, Focus eLearning has also provided an analysis page which contains additional information for the student such as using the on-screen results to plot a graph and how to derive information from it. Users can press on the 'Practical Points' button which gives additional information about how to conduct the experiment. If I have enough time, I could incorporate a page like this since if a student is unsure or stuck, this could help. This coincides with how my stakeholders considered hints to be valuable when stuck on a question.



Having a labelled demonstration is a great idea here. To replicate the apparatus that is needed, having a real-life picture that the student can relate to is important.



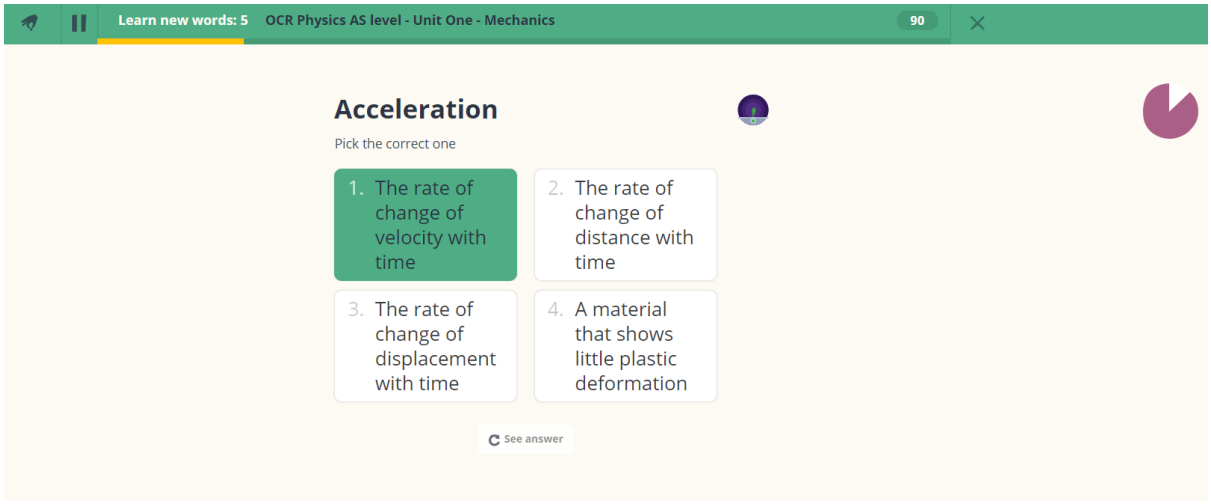
Lastly, Focus have included a video that shows someone performing the practical. While I do appreciate that this may be very useful, it may be hard for me to gain access to such content and to embed it into my program so I will not consider it.

Memrise:

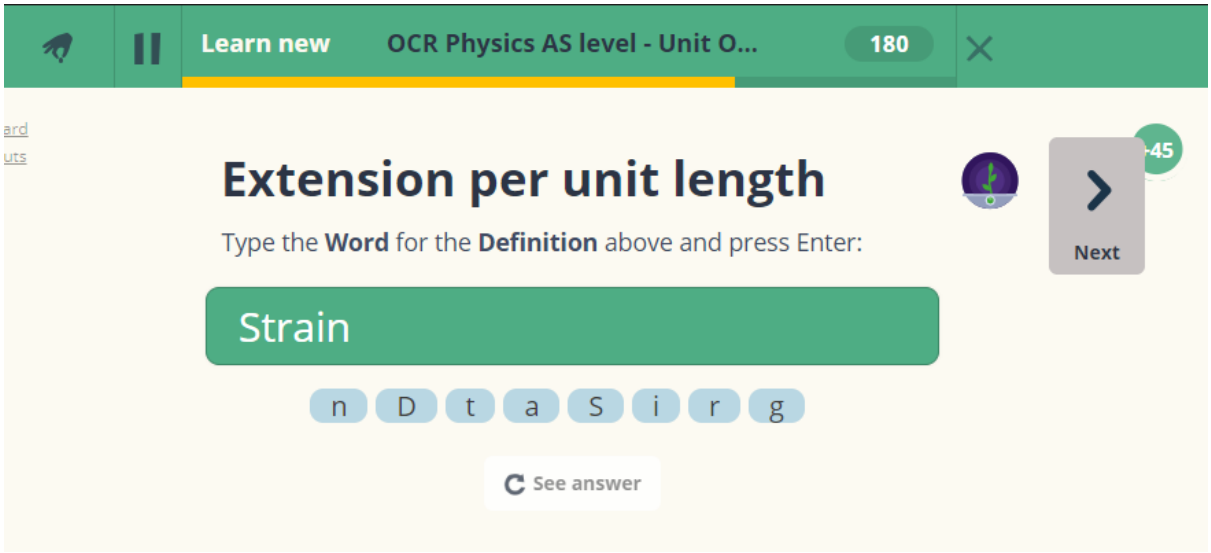
Another piece of software that I have researched is Memrise, which is targeted at people who want to learn new subjects covering a vast range from languages to science. Here, people can improve and practice their knowledge by testing themselves on questions.

The first screenshot depicts the profile page for Memrise. There is access to all the courses that a user can take along with their total accumulated points. At the top are buttons that take the user to different parts of the program – for example, an option to log out and exit Memrise. The window is well laid out and makes it easy to have access to any part of the program. I could incorporate this into my program as a drop-down menu where the students have access to their profile, listing previous scores or wrongly answered questions. These were requested features in my primary research.

I like the use of bright colours and the option of adding a profile picture makes it more personalised for a student and can make it more inspiring to learn. However, too much personalisation could be distracting.

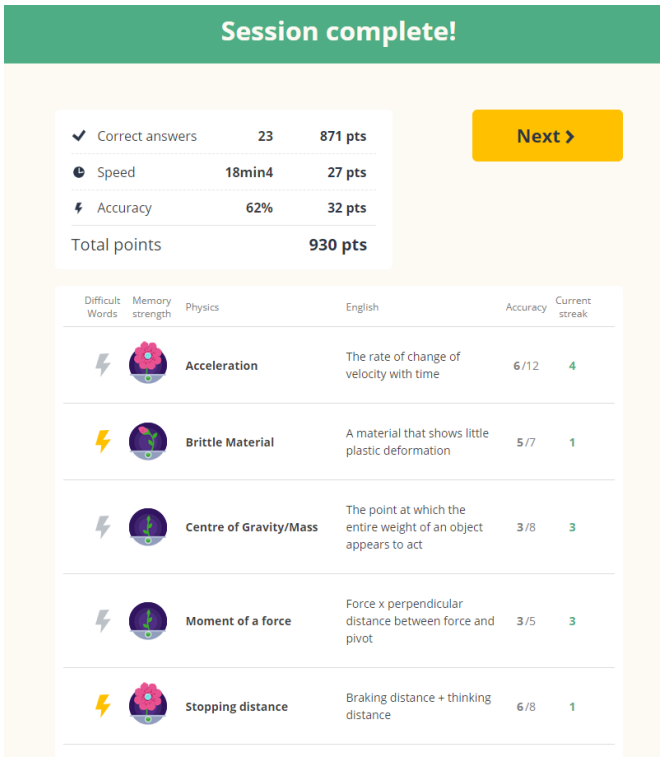


The next screenshot is an example of the multiple-choice questions in Memrise. Again, the use of colours is effective where green denotes a correct answer and red denotes a wrong one. In my opinion, having colours is more encouraging than just having a message telling whether you are correct or not. Memrise has also implemented a timer on the right-hand side. Linking to my primary research, this may seem a little stressful for students so I could have a timer that counts down very slowly or even upwards so that the student must stay on task while having enough time to think about the question. However, Memrise has also included a 'See answer' button which I do not like as it doesn't force the student to think about a question even if they are unsure which isn't effective revision.

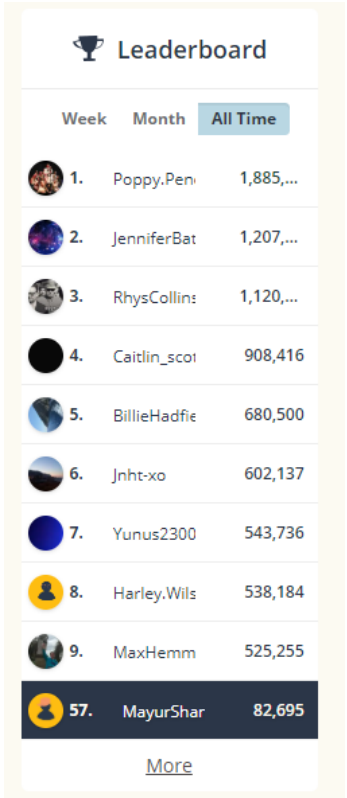


Based on my primary research, a combination of multiple-choice and written answers were considered helpful. I could incorporate this, like the above picture, where the student must pick letters that correspond to the correct answer. Since between 11 and 16 questions were wanted, having ones that require long written answers would be too tedious. Therefore, I could have a

feature where the student can type in a word or two instead to offer some range as opposed to just multiple-choice questions.



This is the review screen where a student can see which questions they have gotten right or wrong. Memrise is a little different from typical online quizzes as it repeats questions if a student gets it wrong. This is shown where each definition has a 'Memory Strength' indicator. While my program won't be repeating questions during the quiz, I think this page is very useful as it gives a clear breakdown of where the student can improve which is important when I plan for this program to be incorporated into a student's revision time, as referenced by my stakeholder page. Therefore, this could be included in the student's profile so they have easy access to it.



This screenshot is of the leaderboard that Memrise implements. It is calculated by total points a student has accumulated and can be sorted for different periods of times such as weekly or monthly. While one of my stakeholders wanted a feature like this, I feel it less important than the other features that I have mentioned.

Summary of what I will take forward into my program

Focus Education Software has some great features that if implemented, would greatly improve my program.

- Including pictures. It adds to the aesthetic of the program and makes the animation easier to visualise.
- Having labels attached to the various parts of the animation will make it a lot easier for a student to follow.
- Incorporating buttons and sliders for the student to interact with – this builds upon my primary research as interaction was important.
- Having additional information about the practical, such as the apparatus or the graph being plotted - I could incorporate this into the actual animation or include this as hints if the student is stuck on a question.

Memrise

- Having a drop-down screen where a student has easy access to details such as their previous scores or wrongly answered questions. There could be an option to exit the program or have the graph of their progress drawn.
- Including a review screen. This doesn't have to be complicated as I could just display the correct answers next to the wrongly answered answers that a student gave.
- Have questions where students must pick letters or words to make up a correct definition. This would provide variety but I feel that multiple-choice questions should still take precedence.
- A leader board would be a fun feature to include however I don't think it is a necessity as the focus should be on the student's own progress.
- Adding colours would make my quiz more engaging. For example, when a student gets a question right, green would be displayed, and if they get it wrong, red would be displayed.

HARDWARE REQUIREMENTS:

Item	Justification
Monitor	The user needs to see the program
Mouse	The user will need to navigate the graphical user interface, such as clicking on various buttons
Keyboard	For example, a user's details such as username and password can be entered
Speaker or Headphones	This is so sound effects can be played
1 GHz or better processor	This so the program meets the requirements of Pygame and Tkinter
1-2Gb of RAM	While the program will not be memory intensive, it will enable the user to run other processes in the background

200 Mb of space	Not much will have to be stored. Only images needed for the animation and files containing student details. This storage requirement covers the fact that my program may be compiled to an exe file.

SOFTWARE REQUIREMENTS

Item	Justification
An OS capable of running Python or exe files	Most operating systems are compatible with Python so this shouldn't be a problem. If not, then my Python script could be converted into an exe file.
Any Python version, version 3 or above	The code will be written in the Python language
Pygame	This is so I can include an animation of the Physics practical
Tkinter	This will make up the main user interface that the user will navigate through
Matplotlib	This is so I can draw graphs, whether it be for the animation or the student's progress

FEATURES OF MY PROPOSED SOLUTION:

A login system:

I want my solution to have a comprehensive and robust login system. A user would type in their username and password which would be compared to a file with the correct username and password. If correct, they are allowed access, however, if wrong, they will be allowed another go. There will also be a create account button which allows a new user to create a new username and password which will be appended to a file. They can then login with those credentials. Initially, in my analysis, I wanted the student to also input the current date so that the graph of their progress could be displayed, however, I can find out the current date in Python itself so there is no need.

A problem I might face would be if a user has forgotten their username or password. For their progress to be saved, these login details will be needed. The program could provide a hint, but this wouldn't be as secure. Therefore, this will be a limitation of my proposed solution since implementing a system to recover their login details will be too complicated. The best option would be to create a new account.

Also, a nice feature to include would be a counter which counts how many times a user has tried to log in if their credentials are wrong. After a certain amount of tries, the program could lock the user out for 5 minutes before trying again.

Practical Animation:

This feature is fundamental to the way this solution will work. To start with, I will create a labelled animation of the Determination of 'g' by a freefall method practical which will simulate the motion of a falling ball bearing. I want to include sound effects and images as this was requested by my primary and secondary research.

Along with this, the student can click a button to show a corresponding graph in a separate window. However, the accuracy of the recorded results may prove to be problematic. While my program is an abstraction of Physics practicals, I still need to make sure that the graph shows correct results. Therefore, a limitation of my proposed solution would be the inability to show results that reflect the real-world.

It would be nice to include some sliders and buttons that can change what is onscreen to make it interactive. Once the program has finished, there will be two buttons, one to exit and close the window, and one to repeat the animation. Since users may only want to view the animation, there should be an option to view it without doing the questions.

Integrated quiz:

This feature will allow students to test their knowledge. I want the program to display a bank of around 11 to 16 multiple-choice questions, one at a time. The student can press on buttons that are labelled with various answers. There will be a hint button if a student is struggling and the correct answer will be displayed after they have submitted their answer. I want to include a timer, however, I will need this timer to give adequate time as demonstrated by my primary research. Additionally, if I have time a pause button could be implemented if the student cannot continue. Moreover, I will need to append to another file the questions that the student got right or wrong so they can review it later. As part of this, there should be functionality which allows the user another go at the quiz so that they can improve their score.

Graphing Student Progress:

Once the student has completed the test, they will be given a score that will be written to a file alongside the date. From this file, I can use Matplotlib to draw a graph of the student's scores against time so the student can see how well they are doing.

Student Review:

This could be a drop-down menu where a student can access their information. For example, they could view their past scores or wrongly answered questions. Furthermore, this could include a feedback page where students can write what could be improved but this will likely be an optional extra. A nice feature to include is for the program to show more information about what the student got wrong such as explanations or worked examples.

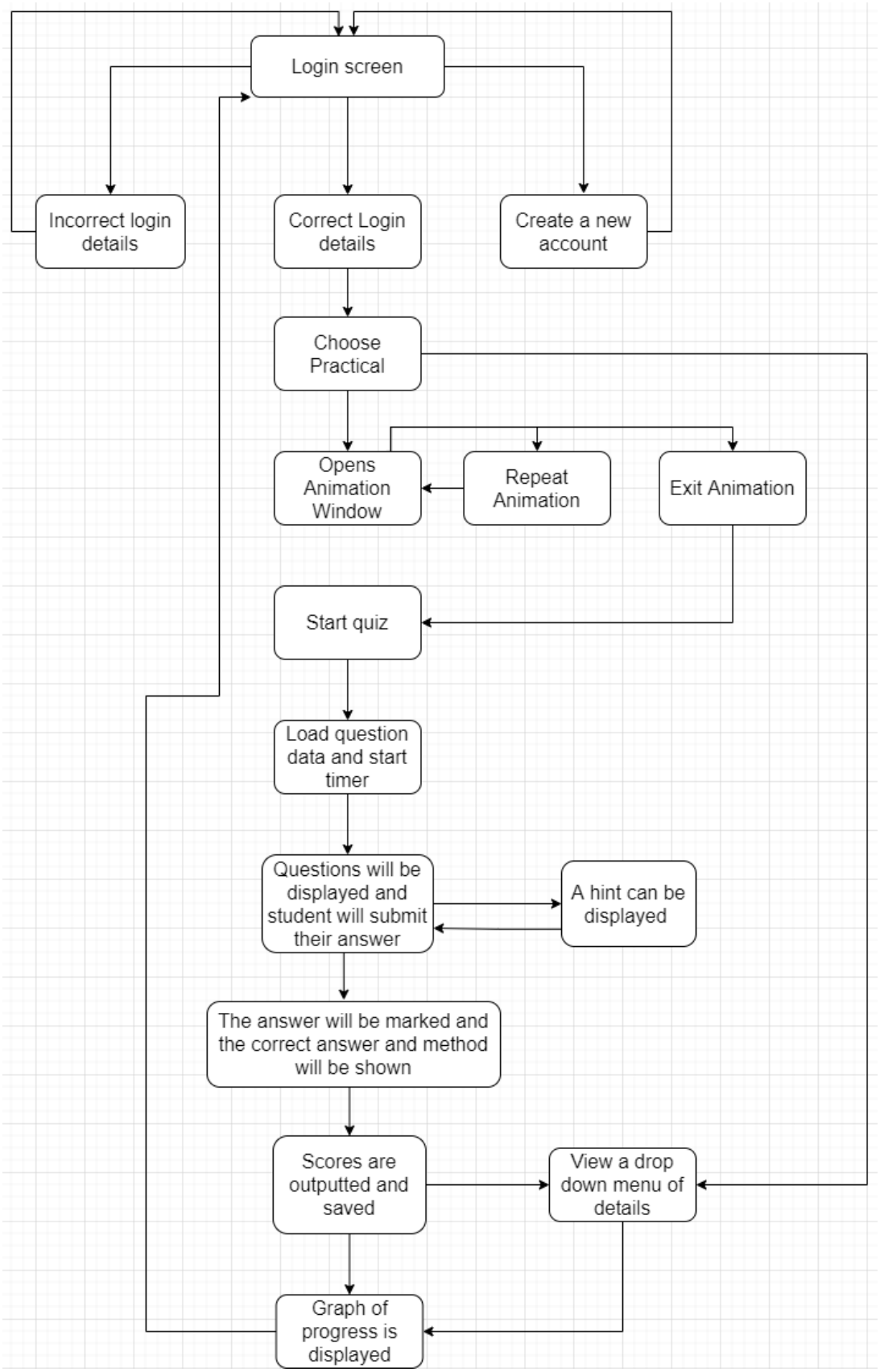
SUCCESS CRITERIA:

1. The program must have a robust and functional login system that allows students to sign in.
2. There should be a validation system so the student can try again but there should a limit to this of around 5 attempts.
3. The student must be able to create a new account with which they can log in.

4. The program must use CSV files to store student data, so it is well organised and easy to manipulate.
5. The practical animation should have interactive buttons and sliders. This is demonstrated in my primary research as my stakeholders were keen on having this ability.
6. The animation must have labels, images and sound effects.
7. The user should be able to view a graph within the animation linking to my secondary research.
8. The user should be able to view the animation as many times as they want.
9. The student should have the option to only watch the animation and skip the quiz.
10. The quiz should ask no more than 16 multiple choice questions to keep it from becoming too tedious.
11. The questions must be of reasonable difficulty and be based on aspects of the practical process.
12. There should be a working timer, so the students must complete questions in an allocated time. However, I must ensure that there is ample time to reduce stress.
13. The correct answer that should have been pressed should be displayed if the student gets the question wrong.
14. Once they have finished the questions, a graph of a student's progress must be plotted for them to easily visualise their progress.
15. There should be an option for the user to have another go at the quiz.
16. Having a drop-down menu with the student's details such as past scores or wrongly answered questions.
17. Once finished, the student can have the option to log out and return to the main menu.

DESIGN:

System Diagram:



KEY VARIABLES AND STRUCTURE:

Since I will be using classes, I will be using attributes mainly.

Attributes	**Description**	**Data type**
Login.Username_entry	The student's username that is inputted into the program	String
Login.Password_entry	The student's password that is inputted into the program	String
Login.Correct_username	The correct username that is compared to what the user enters	String
Login.Correct_password	The correct password that is compared to what the user enters	String
Login.New_username	The new username entered by the student when creating an account	String
Login.New_password	The new password entered by the student when creating an account	String
Login.StudentCredentials	A file of all the student's details	CSV file
Login.Button	Button used to check credentials or to create a new account	Boolean
Animation.Colours	The various colours that will be used	Integer
Animation.Shapes	The different shapes that will be used	Integer
Animation.Image	Images that will be displayed	File
Animation.Positions	The coordinates of all the objects in the window	Integers
Animation.Graph	Graph of practical results can be drawn	File
Animation.Button	Buttons that will allow the user to interact with the animation	Boolean
Quiz.Questions	List of questions to be asked to the user	CSV File
Quiz.Answers	List of correct answers	CSV File
Quiz.StudentAnswers	List of answers that a student gives	List
Quiz.StudentAnswersCorrect	List of correct answers that a student gives.	List
Quiz.StudentAnswerIncorrect	List of incorrect answers that a student gives	List
Quiz.StudentAnswerCheck	Checks whether the student's answer is correct	Boolean
Quiz.Button	The button the student presses when answering questions	Boolean
Quiz.Scores	The student's total score	Integer
Graph.Date	The date when the user completes the quiz	String
Graph.Plot	The graph of the student's scores against time is plotted	File

Student.ReviewButton	The button that the user presses to access their profile	Boolean

Variables:

Name	Description	Datatype
LoginCount	Count of the tries the student has had when logging in	Integer
TimeCount	Count of how long it takes for the user to complete the quiz	Integer

Validation:
I need to make sure that correct and valid data is entered into the login attributes when signing in or creating an account. I can do this by checking that the right data type is entered. For example, inputs with numbers, symbols or no characters could be rejected.

Validation will need to be incorporated into the animation to make sure that all possible inputs are dealt with. An example is if a user wants to change an aspect of the animation which would not be feasible in real-life.

Classes:

There will be five main classes which I will be using. Within these classes will be additional methods

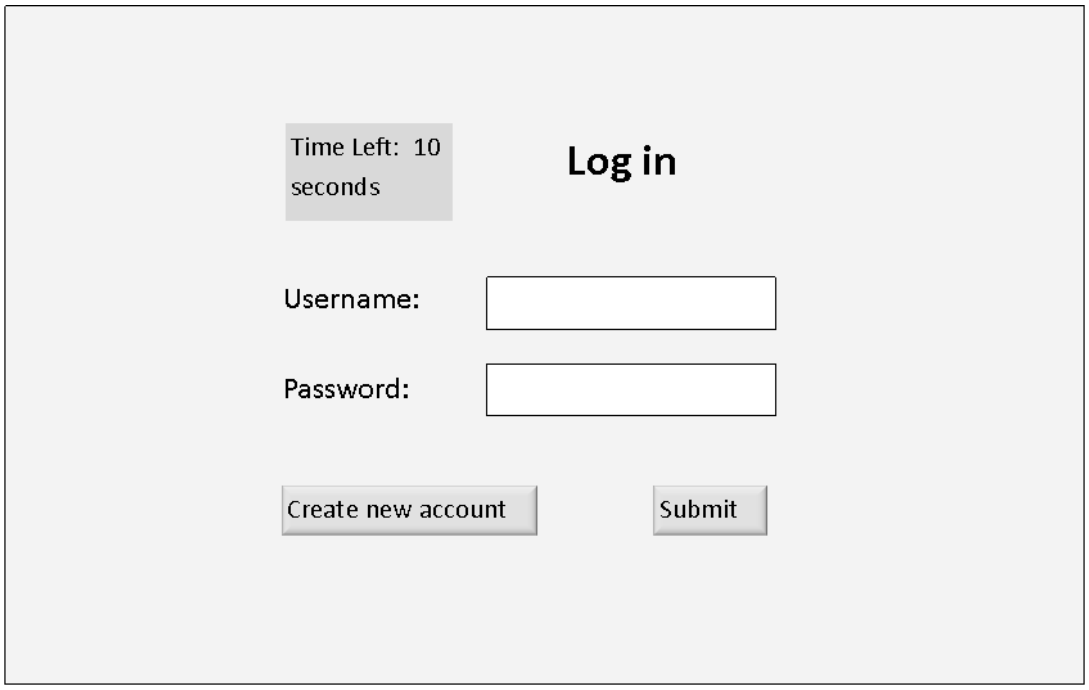
Class	**Attributes**	**Methods**
Login Class	Login.Username_entry	Checking Credentials
	Login.Password_entry	Creating a new account
	Login.Correct_username	Limiting login tries to 5
	Login.Correct_password	
	Login.New_username	
	Login.New_password	
	Login.StudentCredentials	
	Login.Button	
Animation class	Animation.Colours	Drawing images
	Animation.Shapes	Drawing objects
	Animation.Image	Drawing graphs
	Animation.Positions	Button inputs
	Animation.Graph	
	Animation.Button	
Quiz class	Quiz.Questions	Displaying questions and answers
	Quiz.Answers	Receiving a student's answer
	Quiz.StudentAnswers	Changing score
	Quiz.StudentAnswersCorrect	
	Quiz.StudentAnswerIncorrect	
	Quiz.StudentAnswerCheck	
	Quiz.Button	
	Quiz.Scores	
Graph class	Graph.Date	Displaying graph
	Graph.Plot	

Student class	Quiz.StudentAnswersIncorrect	Displaying past scores or wrongly answered questions
	Quiz.Scores	
	Student.ReviewButton	

SCREEN DESIGN:

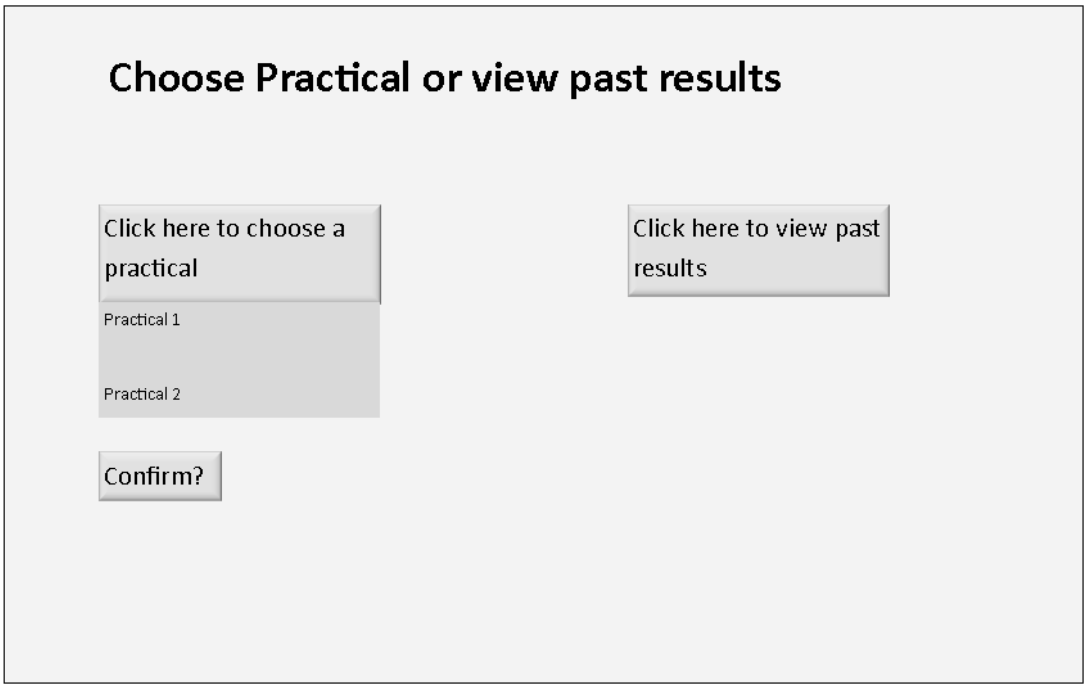
I will now talk about my proposed screens and their usability features. Usability will be a theme that will continue throughout the rest of the project.

Screen 1:
The first screen will be the login screen where the user can log in with their credentials using the entry fields, “Username” and “Password”. As part of usability, anything written in the password entry boxes will be displayed as asterisks. This improves the program’s security as it prevents other people from seeing the user’s password, especially when logging in in public places. There is also a button called “Create new account” which transitions the user to a new screen where they can create a new account. Since validation will be used, the user will be prompted to enter their details again if they enter them wrong on their first try. As part of my success criteria, the user will have five attempts to enter the correct details. After that, there will be a timer as shown where any new logins will not be accepted to prevent brute force attacks. Once the time has passed, the user will be returned to the original login screen.



Screen 2:
The second screen shows where the user can input their new username and password when creating a new account. The submit button will save these new credentials to a file. It is most likely that the user will want to login with these credentials – to make the usability of the program easier, the submit button takes them back to the original login screen. Again, anything written in the password entry boxes will be displayed as asterisks to improve security. Validation could be implemented where inputs with no characters will be rejected.

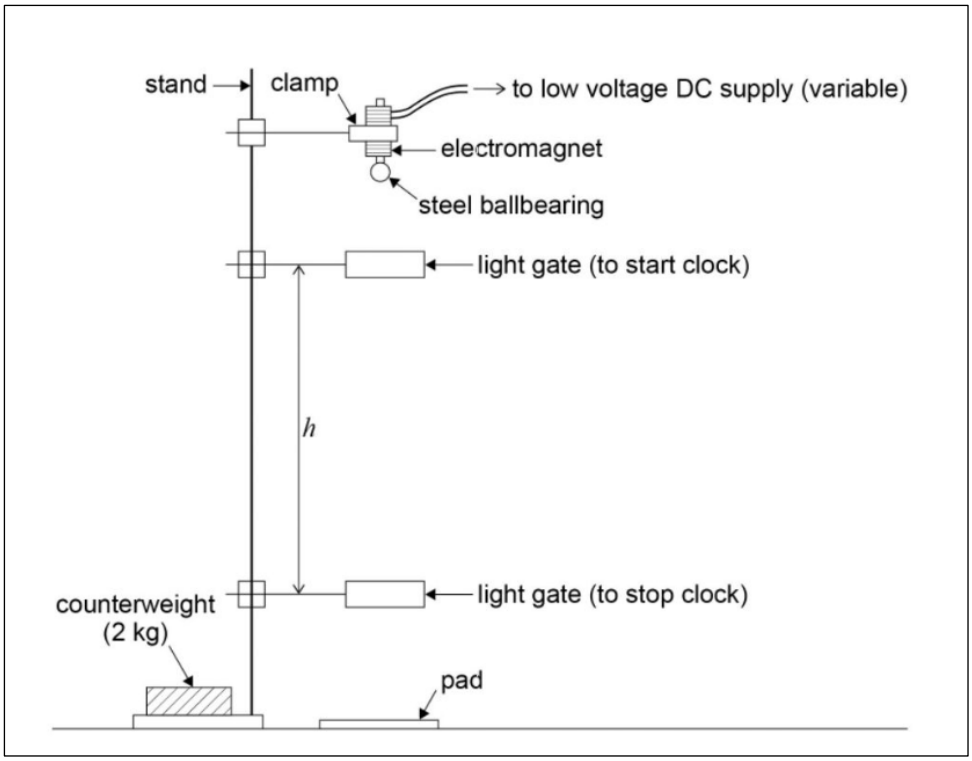
Screen 3
The next screen is where the student can choose from a list of practicals. I am using a drop-down menu as it will be easier for a user to navigate, improving usability. I can have much more information to display without the screen being too cluttered. For example, in future development, it will be easier to add more practicals. Additionally, I have included a 'Confirm?' button which the user has to press. The user may accidentally click an option in the dropdown menu without intending to which could start the practical that they don't want to watch– the 'Confirm' button is an example of usability since it prevents this. Also, there is a button to the side of the dropdown menu where the user can access their past results.

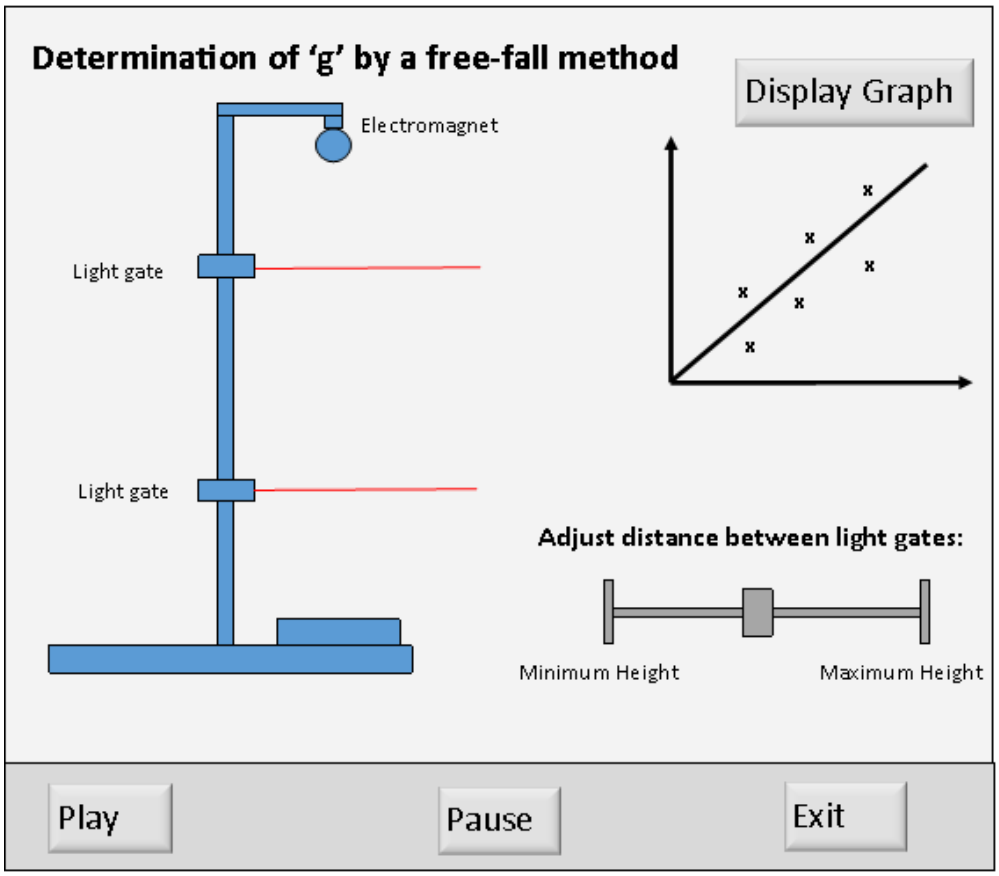


Screen 4

This is the animation screen. I have included a reference diagram which is from a textbook to give me an idea of how this practical should look. This animation will be in a separate window as I will be using a different programming module for it (Pygame).

In the screen design below, I will have a diagram of the experiment initially displayed. The user can use a slider which will change aspects of the practical such as height. The use of a slider is an example of a usability feature since it is easy to use and allows a range of heights to be chosen. I was going to use buttons with each one labelled with a height however this would limit how many heights the user could try out since there might have not been enough space. I have included a 'Display Graph' button which will display a graph of the results of the animation – for example, the height and time it takes for the ball to drop. At the bottom of the screen are 3 buttons allowing the user to play, repeat the animation or to quit and close the window. These buttons will be permanent so the user will always have access to them allowing them complete control over the animation.

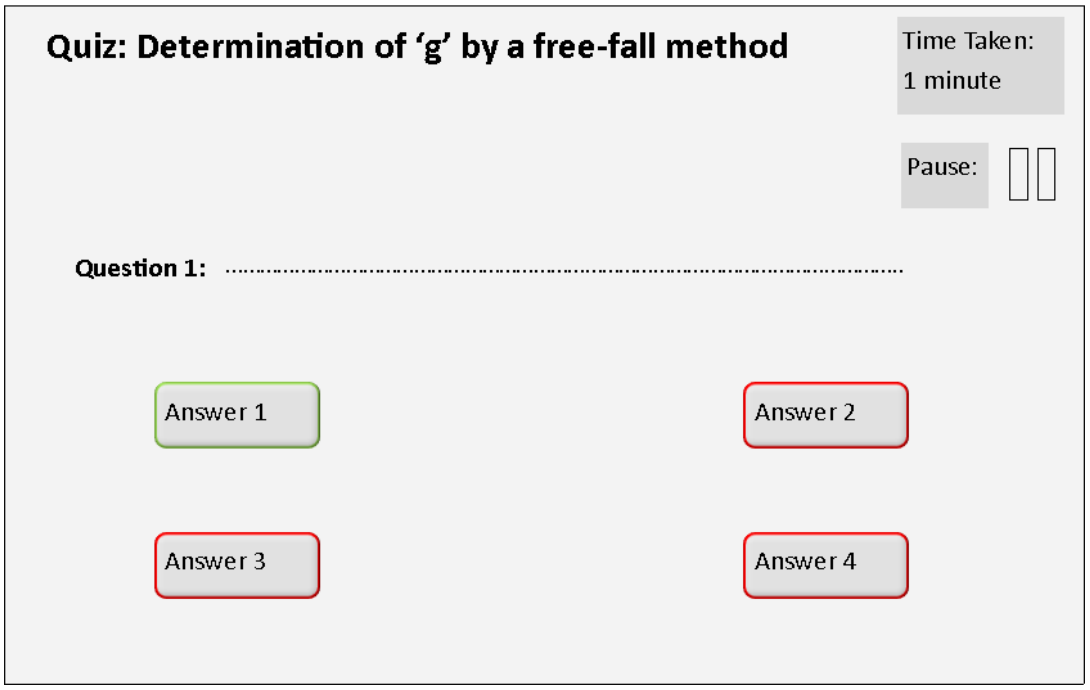

stand
clamp
to low voltage DC supply (variable)
electromagnet
steel ballbearing
light gate (to start clock)
h
counterweight
(2 kg)
light gate (to stop clock)
pad


Determination of 'g' by a free-fall method
Display Graph
Electromagnet
Light gate
Light gate
Adjust distance between light gates:
Minimum Height
Maximum Height
Play
Pause
Exit

Screen 5:

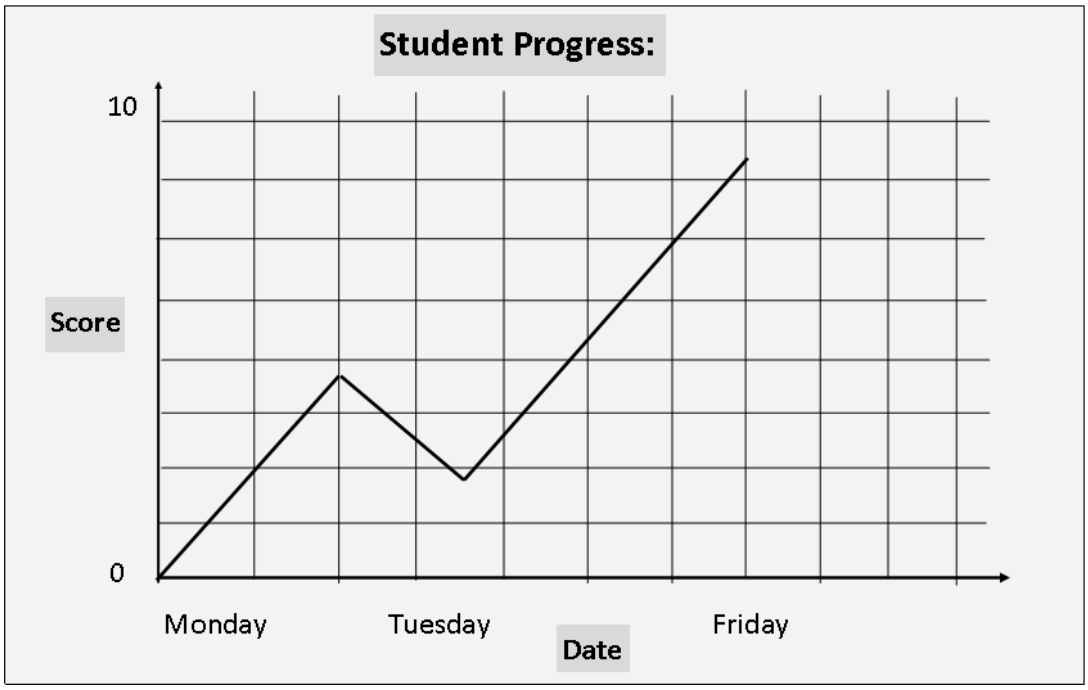
This is the integrated quiz screen where questions are asked one by one and the student has the option of four answers in the form of buttons. Once the buttons are pressed, the answer will be checked and the program will display whether this answer is correct or not. I will show whether the user answered correctly using colours as inspired by my secondary research. This links to usability as it gives life to my quiz and makes it nicer to use.

At the top there is a pause button with the amount of time remaining. This is an important usability feature since if a student must leave for any reason, they do not have to forfeit the quiz.



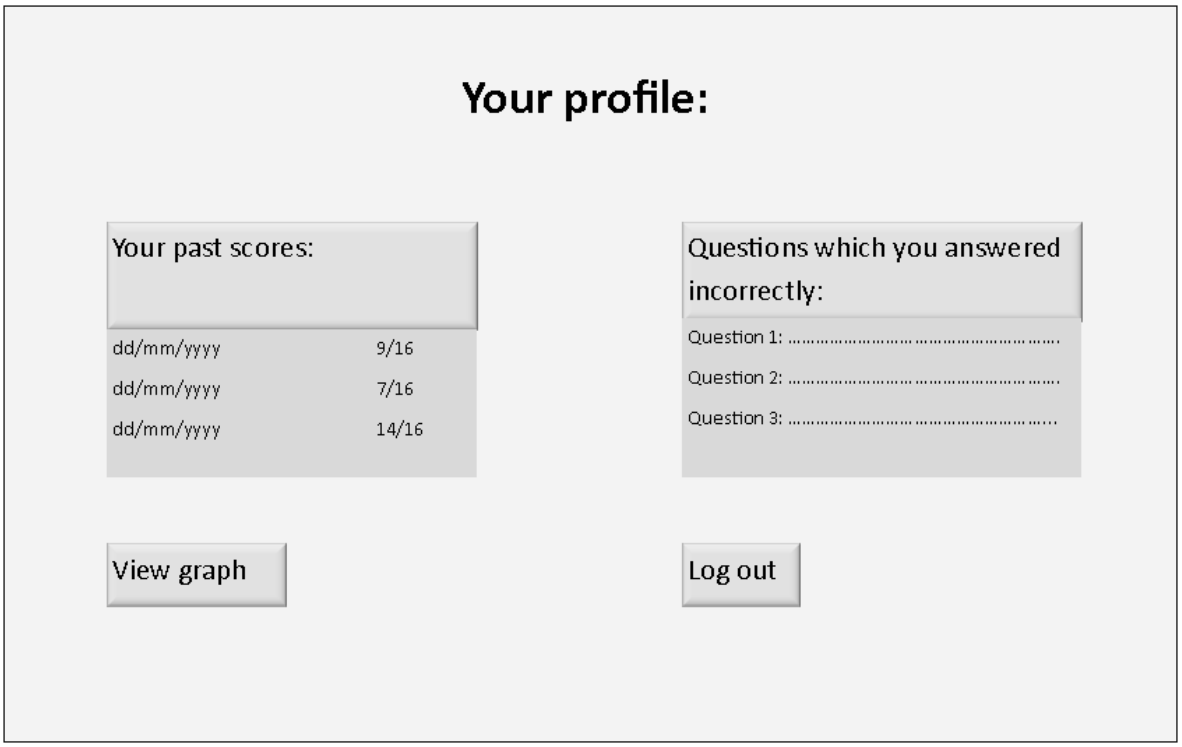
Screen 6:

This screen will be the graph of the student's progress. Using the list of the student's scores, the graph will display score against time. I will have this graph be displayed in a separate Matplotlib window. The use of a grid is a usability feature that I have implemented since it makes it easier for the user to assess their progress over smaller periods of time.



Screen 7:

The final screen that I will incorporate will be the student's profile. Here, they can view their past scores and see which questions they answered incorrectly. I have used drop-down menus as they facilitate usability when displaying lots of information. For example, the student could scroll through their scores which could be ordered by date. As part of my success criteria, there is an option for the user to log out and return to the login screen. Additionally, they can view the graph of their progress again to see how their past scores match up to the graph. This improves the program's usability as the user has more control over what they can do.



PSEUDOCODE

Login screen:

```
CLASS LoginScreen:
        IMPORT Login.Correct_username as string
        IMPORT Login.Correct_password as string
        LoginCount = 0
        DRAW LoginScreen
        Login.Username_entry = INPUT 'Enter Username'
        Login.Password_entry = INPUT 'Enter Password'
        WHEN Key_Enter PRESSED:
                IF Login.Correct_username == Login.Username_entry AND Login.Correct_password
                == Login.Password_entry THEN
                        CLOSE LoginScreen
                        RUN CLASS Animation
                ELSE IF LoginCount == 5 THEN
                        WAIT 30 seconds
                        LoginCount = 0
                        RUN CLASS LoginScreen
                ELSE:
                        RUN CLASS LoginScreen
                        LoginCount += 1
                ENDIF
        ENDWHEN
ENDCLASS
```

The above class will be used so that the student can log in into the program. After receiving their input, it will be compared to the correct username and password that will be stored in a CSV file. If valid, the user can progress onto the animation otherwise they will have an opportunity to re-enter their details as part of validation. I have included the variable called 'LoginCount' to keep track of how many tries the user has. When 5 is reached, they must wait 30 seconds before they can enter their details. If the user fails to enter their details after this, I have reset this variable to 0 so this process can be repeated.

Creating a new account:

```
METHOD CreateAccount
        DRAW CreateAccount screen
        Login.New_username = INPUT 'Enter new username'
        Login.New_password = INPUT 'Enter new password'
        WHEN Key_Enter PRESSED
                IF LEN (Login.New_username) or LEN (Login.New_password) == 0 THEN
                        RUN METHOD CreateAccount
                ELSE
                        Login.StudentCredentials APPEND Login.New_username
                        Login.StudentCredentials APPEND Login.New_password
                        RUN CLASS LoginScreen
                ENDIF
        ENDWHEN
ENDMETHOD
```

The above method is used so that the student can create a new account. Once they have inputted their details, it is appended to a CSV file which contains the student details. After, they must enter their details again via the login screen. For validation, I have used an if statement which will reject any inputs with no characters entered.

Animation:

```
CLASS Animation
        IMPORT Animation.Colours as integer
        IMPORT Animation.Shapes as integer
        IMPORT Animation.Image from file
        IMPORT Animation.Positions as integer
        IMPORT Animation.Graph from file
        DRAW InitialAnimation Screen
        IF Animation.Button PRESSED THEN
                RUN METHOD NewAnimationScreen
        ELSEIF Animation.RepeatButton PRESSED THEN
                RUN CLASS Animation
        ELSEIF Animation.GraphButton PRESSED THEN
                RUN METHOD AnimationGraph
        ELSEIF Animation.ExitButton PRESSED THEN
                EXIT Animation
```

```
        ENDIF
ENDCLASS
```

First, after importing the attributes into the class, I have set a series of IF and ELSEIF statements that respond to the user's inputs. For example, when the Animation.Button is pressed, a corresponding change is made to the animation such as increasing distance between objects. Using iteration, the user has the option to re-watch the animation or exit it entirely.

Animation Screens:

```
METHOD InitialAnimationScreen
        DRAW InitialObject
        DRAW InitialText
        WHILE TRUE
                REDRAW InitialObject
        ENDWHILE
        RUN CLASS Animation
ENDMETHOD

METHOD NewAnimationScreen
        DRAW NewObject
        DRAW NewText
        WHILE TRUE
                REDRAW NewObject
        ENDWHILE
        RUN CLASS Animation
ENDMETHOD


METHOD AnimationGraph
        DRAW GraphObject
        DRAW GraphText
        WHILE TRUE
                REDRAW GraphObject
        ENDWHILE
        RUN CLASS Animation
ENDMETHOD
```

These methods illustrate how the screens will change according to the input from the user. The methods will inherit all the attributes from the Animation Class, therefore, all that needs to be done is drawing them onscreen. There will be some sort of iteration to allow the objects in the screen to move around. After, the Animation class will be run again so the program can check for additional user inputs – for example, if a user decides to repeat the program but then wants to exit it.

Quiz:

```
CLASS Quiz:
        IMPORT Quiz.Questions as array
```

```
        IMPORT Quiz.Answers as array
        IMPORT Quiz.Scores as integer
        TimeCount = 0 SECONDS
        WHILE TRUE
                TimeCount += 1 SECOND
                DRAW Quiz Screen
                DRAW Quiz.Questions [0]
                IF Quiz.Button PRESSED THEN
                        IF Quiz.StudentAnswers == Quiz.Answers[0] THEN
                                Quiz.StudentAnswerCheck == TRUE
                                Quiz.StudentAnswerCorrect.APPEND (Quiz.StudentAnswers)
                                Quiz.Score += 1
                                IF Quiz.Question == LAST Quiz.Question THEN
                                        DRAW Score Screen
                                        DISPLAY TimeCount
                                        RUN CLASS Graph
                                ELSE
                                        NEXT Quiz.Question
                        ELSE
                                Quiz.StudentAnswerCheck = False
                                DRAW Quiz.Answer[0]
                                Quiz.StudentAnswerIncorrect.APPEND (Quiz.StudentAnswers)
                                IF Quiz.Question == LAST Quiz.Question THEN
                                        DRAW Score Screen
                                        DISPLAY TimeCount
                                        RUN CLASS Graph
                                ELSE
                                        NEXT Quiz.Question
                                ENDIF
                        ENDIF
                ENDIF
        ENDWHILE
ENDCLASS
```

This is the pseudocode for the Quiz class. After initializing all attributes, the program starts a while loop that increases the 'TimeCount' variable. The second IF/ELSE statement decides whether the student's answer is correct. If it is, the score is incremented. If not, the correct answer that should have been clicked will be displayed. The program also checks if the student is on the last question. If this is true, the Graph Class is then run. If not, the next question will be loaded.

Graph:

```
CLASS GRAPH
        IMPORT Graph.Date as integer
        IMPORT Quiz.Scores as integer
        DRAW Graph.Plot
        IF Logout.Button PRESSED THEN
                RUN CLASS Login
```

```
        ELSEIF Profile.Button PRESSED THEN
                RUN CLASS Profile
        ELSEIF Exit.Button PRESSED THEN
                EXIT Program
        ENDIF
ENDCLASS
```

After receiving the current date when the student took the quiz and their score, this can be plotted in Matplotlib to produce a graph. After, the user can have the option of returning to the login screen or can view their profile or can exit the program entirely.

Student Profile:

```
CLASS PROFILE
        IMPORT Graph.Date as integer
        IMPORT Quiz.Scores as integer
        IMPORT Quiz.StudentAnswersIncorrect as string
        IMPORT TimeCount
        Student Drop-down menu = [Graph.Date, Quiz.Scores, Quiz.StudentAnswersIncorrect,
        TimeCount]

        IF Student.ReviewButton PRESSED
                DISPLAY Student Drop-down menu
        ELSEIF Graph.Button PRESSED THEN
                RUN CLASS Graph
        ELSEIF Logout.Button PRESSED THEN
                RUN CLASS Login
        ELSEIF Exit.Button PRESSED THEN
                EXIT Program
        ENDIF
ENDCLASS
```

The final class will allow students to view their past scores and wrongly answered questions. After they have viewed them, they can exit to the login screen or can view the graph of their progress again which corresponds to my screen designs.

TESTING.

Below is the test data that I will use for development.

Test Table for development:

Test	Test Type and Data	Expected Outcome	Justification	Test Number
Login:				
Username	Invalid input – no characters or wrong username entered	Username does not match the stated value in the CSV file so must	To ensure authorised access	1.0

		be rejected and re-entered		
	Invalid input – and the maximum number of tries has been reached	User will be unable to enter their credentials for some time	To prevent brute-force attacks	1.1
	Valid input – correct username	Username does match the stated value in the CSV file so is accepted	To ensure authorised access	1.2
Password	Invalid input– no characters or wrong password entered	Password does not match the stated value in the CSV file so must be rejected and re-entered	To ensure authorised access	1.3
	Invalid input – and the maximum number of tries has been reached	User will be unable to enter their credentials for some time	To prevent brute-force attacks	1.4
	Valid input – correct password	Password does match the stated value in the CSV file so is accepted	To ensure authorised access	1.5
Submit Button	Valid input – button is pressed	Allows the user to successfully log in or else they have to re-enter their details	To ensure that the student can log in successfully	1.6
Create Account Button	Valid input – button is pressed	Allows the student to move to a new screen where they can create a new account	The student can create a new account	1.7
Creating a new account				
Username	Invalid input– no characters are entered	Username must be rejected and re-entered	To ensure a suitable username is saved	2.0
	Valid input – valid username entered	Username is accepted and will be written to a file	To ensure a suitable username is saved	2.1
Password	Invalid input– no characters entered	Password must be rejected and re-entered	To ensure a suitable password is saved	2.2

	Valid input – valid password entered	Password is accepted and will be written to file	To ensure a suitable password is saved	2.3
Submit Button	Valid input – button is pressed	Username and password will be successfully written to file and then the user will be returned to the login screen.	Allows the user to log in with these credentials	2.4
Choosing Practical				
Choose Practical Button	Valid input – button is pressed	Open a new screen containing the animations	Allows the user to choose an animation to watch	3.0
Drop-down button	Valid input – button is pressed	Allows access to the drop-down menu	Allows the user to bypass the quiz	3.1
Animation				
Images button	Valid input – button is pressed	Shows images of the practical equipment that will be used	Adds more realism to the animation	4.0
Play button	Valid input – button is pressed	The animation is played – e.g. the ball is dropped	User can watch the animation	4.1
Slider	Valid input – slider is moved Invalid input – slider is moved beyond its limits	Allows the user to change an aspect of the practical such as height	Allows the user to interact with the practical	4.2
Any other buttons or key presses	Valid input – button is pressed	There is a change onscreen – e.g. displaying a new object	Allows the user to interact with the practical	4.3
Repeat button	Valid input – button is pressed	Repeats the animation from the start	Allows the user to watch the animation as many times as they want	4.4
If any buttons or sliders are not pressed	Valid input – no buttons are pressed	The animation will either stay static or will carry on without the prompted action that would have been initiated by the button	Allows the user the choice of whether to interact	4.6
Graph button	Valid input – button is pressed	Displays graph of recorded results	Allows the user to view a graph	4.7

Exit button	Valid input – button is pressed	Closes the animation window	Allows the user to quit the animation and return to the main window	4.8
Quiz				
Quiz Button	Valid input – button is pressed	Finishes the animation and allows the user to move onto the quiz	Allows the user to complete the quiz	5.0
Start timer button	Valid input – button is pressed	Starts the timer as the quiz begins	So the student can complete the questions in a timed environment	5.1
Displaying questions from CSV file	Valid input – button to start the quiz is pressed	Displays the questions	So the user can attempt the quiz	5.2
Answer buttons (Will consist of 4 multiple choice answers)	Valid input – button is pressed	If the answer is correct, the user will gain a mark and the next question should be displayed	To check if the student's answer is correct and to allow them to move onto the next question	5.3
		If the answer is incorrect, the user will get to see the correct answer. Then the next question will be displayed.	To check if the student's answer is incorrect and to allow them to move onto the next question	5.4
	Valid input – button is pressed and the student is on the last question	Will give a score and allow the student to repeat the quiz or see their progress on a graph	To allow the student to view their score and progress	5.5
	Valid input – button is not pressed	The quiz will end when the timer finishes	If the user does not want to carry on	5.6
Pause Button	Valid input – button is pressed	Will pause the quiz (including timer)	If the student is unable to continue for any reason	5.7
Repeat Button	Valid input – button is pressed	Will repeat the quiz	If the student wants another attempt at the quiz	5.8
Graphing student's progress				
Draw Graph Button	Valid input – button is pressed	Opens a new Matplotlib window with a graph drawn	So the student can view their progress	6.0

Exit button	Valid input – button is pressed	Exits window and return to the main menu	When the student has finished viewing the graph	6.1
Student Drop Down Menu				
Drop-down menu	Valid input – button is pressed, (when scores have been recorded)	Shows a list of the student's past scores and wrongly answered questions with options to review them.	To allow the student to view their progress and take action on what they have struggled with	7.0
	Valid input – button is pressed, (but when no scores have been recorded yet)	Shows message saying student needs to complete a set of questions first	Prompts the user to complete the quiz	7.1
Exit				
Final Exit button	Valid input – button is pressed	Returns the user to the login screen	So the user can properly exit the program	8.0

POST DEVELOPMENT TESTING

These are the questions that will be asked to the user. There will be a total of four answers to choose from.

Question Number	Question	Correct Answer	Incorrect Answer 1	Incorrect Answer 2	Incorrect Answer 3
1.	Describe the relationship of the drawn graph	Directly Proportional	Proportional	Inversely Proportional	Indirectly Proportional
2.	Click on the closest value to the gradient of the graph	4.9	3	6	9.8
3.	What is the physical significance of the gradient?	½ Acceleration due to gravity	Acceleration due to gravity	Speed of the ball bearing	Gravitational Field Strength
4.	What is an independent variable in this experiment?	Height between light gates	Speed of the ball	Time it takes for the ball to fall	Acceleration of the ball
5.	What is a dependant variable in this experiment?	Time it takes for the ball to fall	Height between light gates	Acceleration of the ball	Speed of the ball

6.	State a control variable	Same surface area of the ball	Time it takes for the ball to fall	Height between light gates	Acceleration of the ball
7.	What is an advantage of using an automatic timer compared to a stopwatch?	More accurate results due to the elimination of human reaction time	More precise results due to the elimination of zero errors	Less likely to fail compared to a stopwatch	Less work needed to operate it
8.	Which of the following do **not** influence the ball bearing's acceleration?	Mass	Air Resistance	Both	Neither
9.	Why is the pad used?	To prevent a trip accident if the ball falls onto the floor	To prevent it from falling on the floor	So the ball bearing doesn't bounce on impact	To quieten the impact of the ball
10.	What is the disadvantage of using an electromagnet?	There could be a small delay between it becoming demagnetised and the ball dropping	The ball may not drop at all and still be attracted to the electromagnet	The electromagnet may not always switch off properly	Electromagnets are more expensive
11.	What is the problem with using small heights?	Increases percentage uncertainty when measuring time	Ball bearing will not accelerate that much	Harder to measure smaller changes in time	Will give erroneous results for the value of g
12.	Why must the distance between the first light gate and the ball be kept the same?	Reduces percentage uncertainty when measuring height	So that the ball enters the first light gate with the same speed	Easier to measure height	Distance doesn't matter
13.	Why must the ball be dense?	To mitigate the effect of air resistance	To ensure the ball falls in a straight line	So the ball falls faster	So the ball doesn't bounce off the pad

IMPLEMENTATION:

For my program, I will be using Python IDLE with a combination of Tkinter, Pygame and Matplotlib. For my main program, I will be using Tkinter as it has the best tools to create a GUI for logging in and creating a quiz. For example, it is much easier to implement buttons and entry boxes. To organise the widgets and labels on each screen, I will be using the grid geometry manager as I found it easiest to manipulate their positions.

The animation will be done in Pygame as it is more suited to moving objects. Once the student moves on to the animation, a new Pygame window can be drawn separate from the Tkinter window. To draw the graph of the student's progress, Matplotlib will be used as I can import the student's scores into it. Other modules that will be imported will be CSV as I plan to have all the student's details in a single file from which I can read from and write to. This fulfils the fourth item in my success criteria which was to use CSV files.

My program will be structured using classes. I found object-orientated programming best suited as it will make my program easier to follow. This is because I will be including many methods and attributes.

Creating a window:

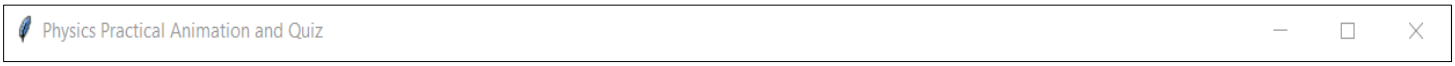
To begin with, I have imported all the required modules and used Tkinter to create a simple window which I have called 'Physics Practical Animation and Quiz'.

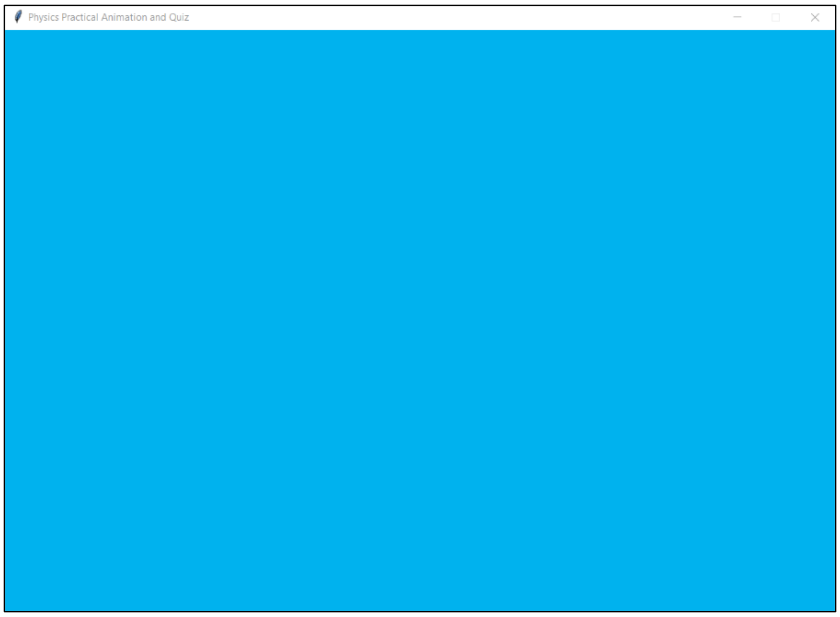
```
1  ##A Level Programming Project##  by  ##Mayur Shankar##
2
3  import pygame #---Importing all the required modules
4  import csv
5  import time
6
7  #Setting up the tkinter window
8  from tkinter import*
9  window = Tk()
10
11 window.title("Physics Practical Animation and Quiz")#---Name of the window
12 window.geometry("1000x700") #---Sets the height and width of the tkinter window
13 window.configure(bg="DeepSkyBlue2") #---Sets the background colour of the window
14 window.resizable(False, False) #--Tkinter window is not resizable
```

On line 9, I have set 'window' to be the main window in Tkinter as this is where I will place all my widgets. Additionally, on line 12, I have set the size of the window to 1000x700 - this provides more than enough space for the widgets that I want to be displayed. Furthermore, due to this increased size, it means I can make fonts and widgets larger which improves the usability of my program.

On line 14, I have chosen for this window to not be resizable. Due to the grid geometry manager, resizing the display could negatively affect the existing position of the widgets. Also, a non-resizable screen allows the student to have other applications open, such as a music player, which was said in my secondary research.

This produces a 1000x700 window with the inputted title like shown below:





Format Class:

To begin with, I have created a new class called 'Format'. Here, I will include all the attributes that relate to changing widget and window options such as font or colour. This is so I can make system wide-changes by changing just one attribute in the class rather than changing every single variable in the program which would be too time-consuming.

```
18 class Format:#--Class for all the formatting that will be used throughout the program
19   def __init__(self, white, black, blue, red, grey, font, bd, fontSizeLarge, fontSizeMedium, fontSizeSmall):
20     self.white = white
21     self.black = black
22     self.blue = blue
23     self.red = red
24     self.grey = grey
25     self.font = font
26     self.bd = bd
27     self.fontSizeLarge = fontSizeLarge
28     self.fontSizeMedium = fontSizeMedium
29     self.fontSizeSmall = fontSizeSmall
30
31 form = Format("white", "black", "medium blue", "firebrick1", "grey40", "Arial", 1, 20, 15, 13)
32 #--This will be expanded as needed
```

On line 19, I have used the parameterized constructor 'def__init__ (self, ….)' to define variables such as font, border width and colour. Tkinter has built-in colours that can be called using keywords which is what I have used. This is much easier than using hexadecimal values. On line 31, the class is instantiated with values. Therefore, 'form.red' will be an instance of this class and can be written instead of 'red'.

Login Class:

The next class that I have written is the main 'Login' class. This will be used to hold most of the attributes that will be required when new methods are created to handle checking entered

credentials or creating a new account. Below are the widgets I have used to draw a screen using attributes from the 'Format' class.

```
36 class Login:#---Initialising the first class that will be used for the login screen
37   def __init__(self):#---Key Attributes labelled
38     #--Tkinter Widgets
39     self.welcomeText = Label(window, text="Welcome, to get started please log in", fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
40     self.accountText = Label(window, text="Don't have an account? Create one!", fg=form.grey,bg=form.white,font=(form.font, form.fontSizeSmall))
41     self.username = Label(text="Username:", fg=form.red, bg=form.white, font=(form.font, form.fontSizeMedium))
42     self.password = Label(text="Password:", fg=form.blue, bg=form.white, font=(form.font, form.fontSizeMedium))
43     self.entryUsername = Entry(window, bd = form.bd, font=(form.font, form.fontSizeSmall))
44     self.entryPassword = Entry(window, bd = form.bd, font=(form.font, form.fontSizeSmall), show='*') #---This is where the user will enter their details
45     self.submit = Button(window, text="Submit", fg=form.black, bg=form.white,font=(form.font, form.fontSizeMedium))
46     self.createButton = Button(window, text="Create a new account", bd = form.bd, fg = form.black, bg=form.white, font=(form.font, form.fontSizeMedium))
```

From lines 39 to 46 are the widgets that will be displayed. Even though on line 39 I have written window, I do not need to write this as there is only one window that Tkinter can display to. When adding options to each widget, I have included the attributes from the 'Format' class such as the colours which change the foreground and background appearance. The three main widgets that have been used are 'Label' which is used for text, 'Entry' so the user can input details and 'Button' which the user can press.

The option 'bd' stands for border width. I have used this to make the buttons have more depth to them. Below, is a border width of 0 versus a border width of 5. I want the buttons to be distinguishable which is why I chose a larger border width.

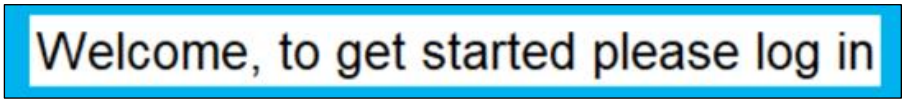

```
55      self.welcomeText.grid()
```

I positioned the widgets using the grid method. However, this displayed them at the top left of the window.

```
login = Login() #--Instantiates the main class
window.mainloop()
```

The above piece of code is used to instantiate the main class. 'window.mainloop()' is used to update the GUI.

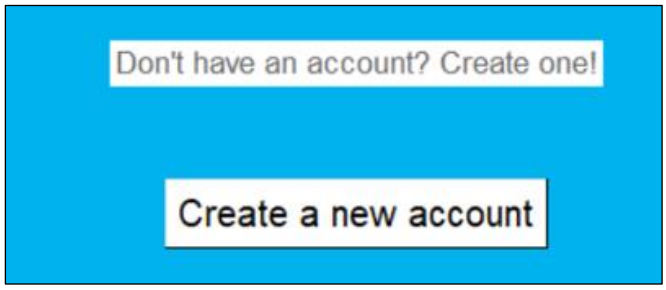
These are the widgets which correspond to my screen designs:



This the 'welcomeText' widget. I have used black text against a light grey background to create contrast as I found it difficult to see the text otherwise.

This widget allows the user to input their details and then submit them using the 'Submit' button. This will call a method which will check the entered details.

On line 44, a nice feature that Tkinter offers is that when a user enters their password, it will appear as asterisks using the 'show' option, which adds to the security of my login system.

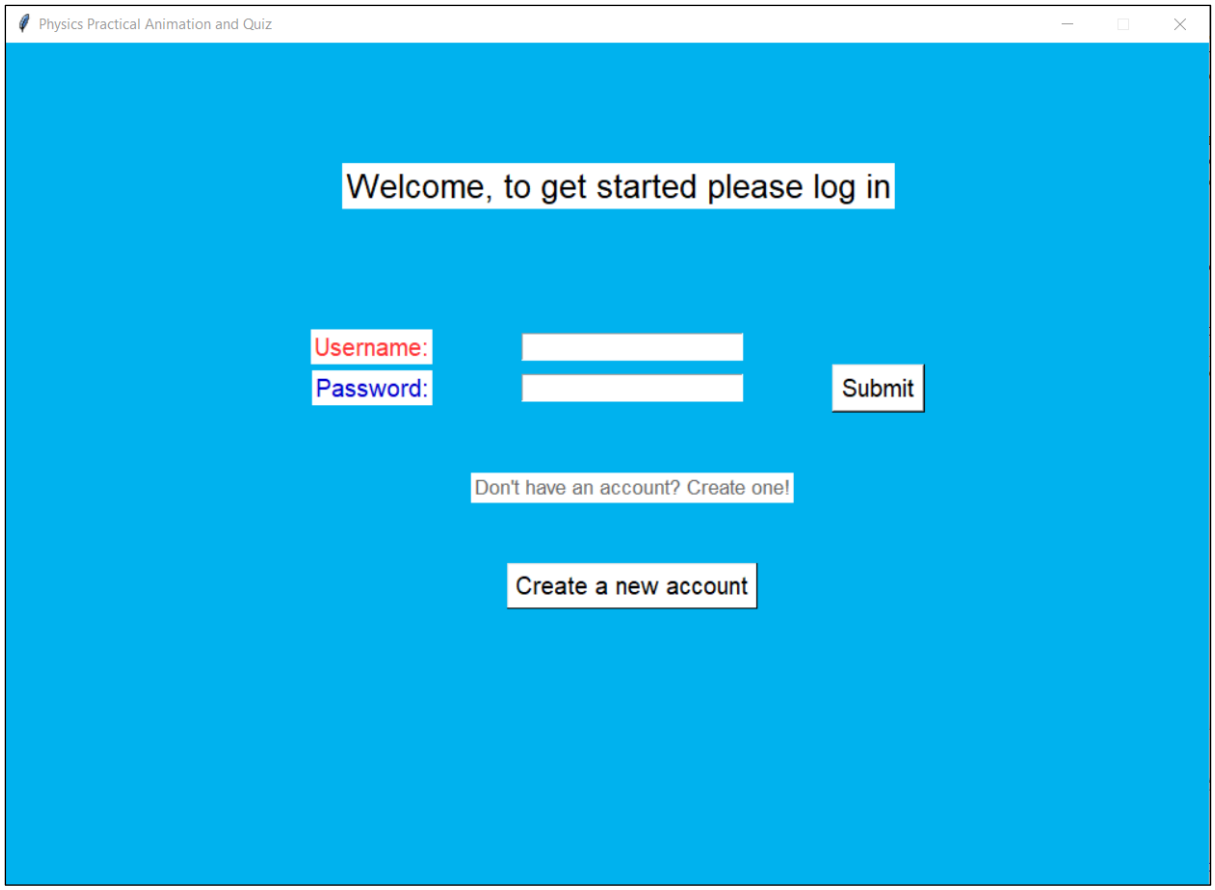


This allows the user to create a new account. The text widget 'accountText' is displayed to prompt the user to create an account. The button will allow the user to transition to a new screen so that they can create a new account.

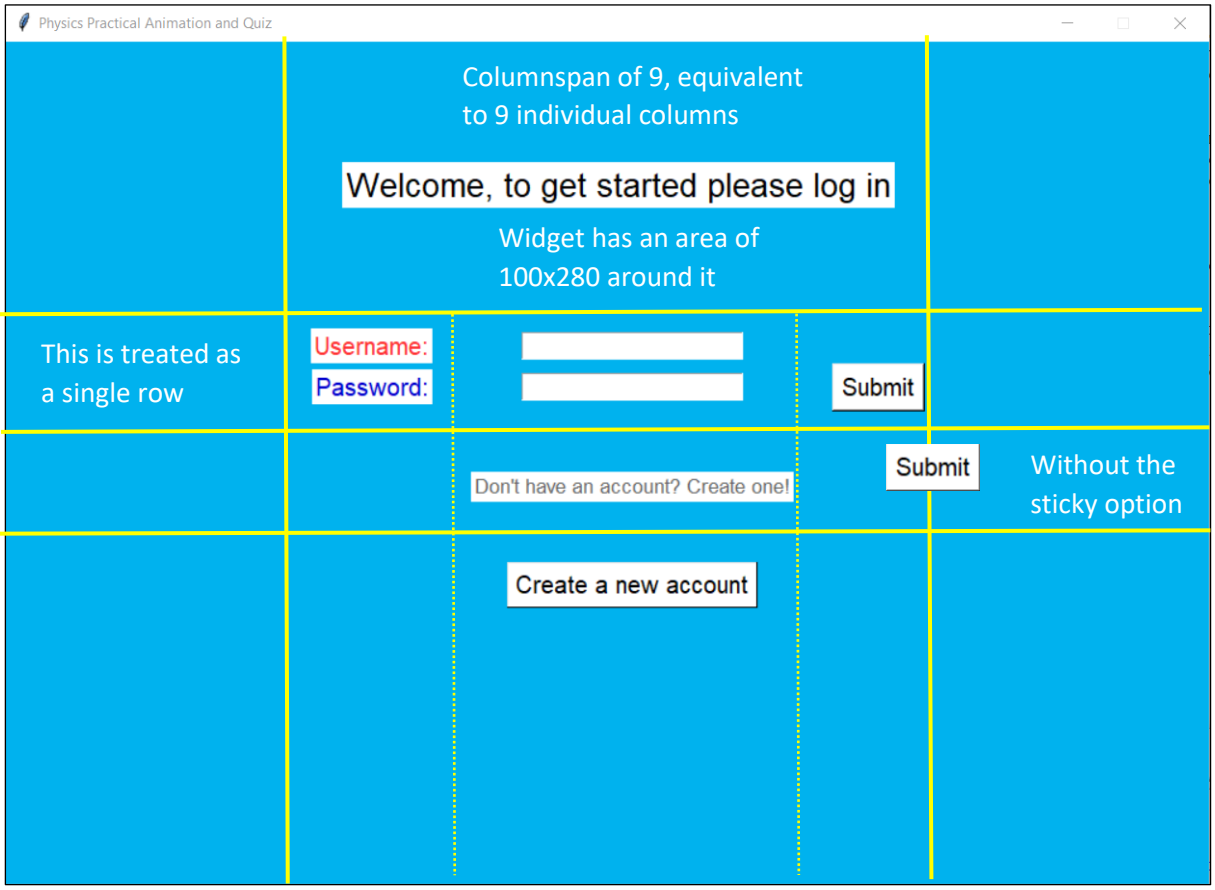
For the widgets to be displayed onscreen properly, I will need to organise the widgets like shown below.

```
46      #--Widget Positioning
47      self.welcomeText.grid(row=1, columnspan=9, pady = 100, padx = 280) #--Main widget in the window
48      self.accountText.grid(row=5, column=4, pady = 50)
49      self.username.grid(row=3, column=3, sticky = E) #--Sticky determines to which side the widget will bind to
50      self.password.grid(row=4, column=3, sticky = E)
51      self.entryUsername.grid(row=3, column=4)
52      self.entryPassword.grid(row=4, column=4)
53      self.submit.grid(row=4, column = 5, sticky = W)
54      self.createButton.grid(row=6, column = 4)
```

This produces a window like this:



Below is a more in-depth analysis of the grid geometry manager.

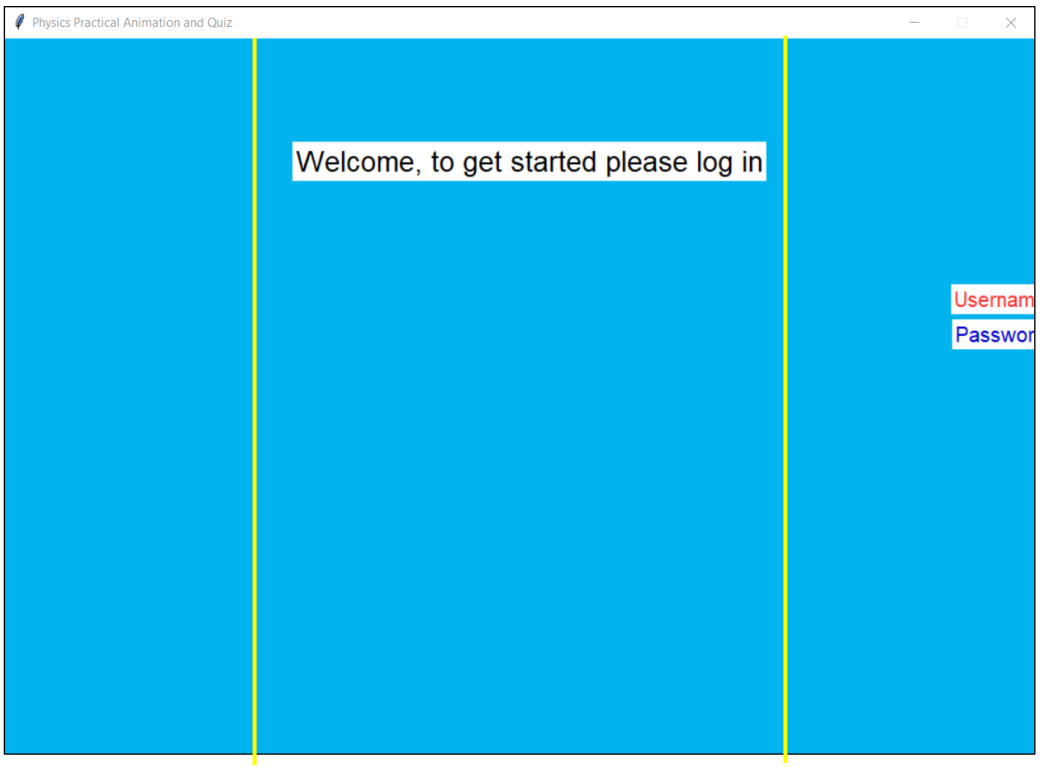


The yellow lines represent the row and column options which are the simplest way to organise widgets, however, this only works in relation to other widgets. For example, a row equal to 7 will not be any further away from a row equal to 2 than a row equal to 3 will be, unless there is more than one widget.

The ‘padx’ and ‘pady’ options make it easy to position single widgets but not multiple widgets close together since ‘padx’ and ‘pady’ create an area around widgets where nothing can be placed in it.

Another option that is being used is ‘sticky’ which determines how are widget will bind to other widgets in the window. Using this, the widgets can be more compact as demonstrated in my labelled window.

Since the ‘welcomeText’ label is the largest widget in the window, I have given it the option of ‘columnspan’ so all the other widgets can be placed underneath it. Otherwise, this would have caused other widgets would have been skewed to the right like below:



Tkinter recognises the ‘welcomeText’ widget as a column which is why the other widgets are not visible. To rectify this, I have set a column span to 9 so the widget covers 9 rather than 1 column. Also, ‘columnspan’ means that all the widgets can be moved easily by just moving the main widget.

Creating a new account:

Below are the methods for creating an account. I have separated this into two parts: changing the screen so the user can enter their new details and appending the new details to a CSV file.

Below are the attributes for the widgets contained in the Login class. These widgets follow a similar layout to the main login screen widgets. This ensures a smooth transition and reflects my screen design.

```
52      #--Creating new account
53      self.createText = Label(text="Please create a new account",fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
54      self.newUsername = Label(text="New Username:", fg=form.red, bg=form.white, font=(form.font, form.fontSizeMedium))
55      self.newPassword = Label(text="New Password:", fg=form.blue, bg=form.white, font=(form.font, form.fontSizeMedium))
56      #---This is where the user will enter their new username and password
57      self.entryNewUsername = Entry(window, bd = form.bd, font=(form.font, form.fontSizeSmall))
58      self.entryNewPassword = Entry(window, bd = form.bd, font=(form.font, form.fontSizeSmall), show='*')
59      self.submitNew = Button(window, text="Submit", fg=form.black, font=(form.font, form.fontSizeMedium)
```

This will be contained in a method called 'createAccount'. Since this method is part of the main Login class, it inherits all of its attributes.

```
82    def createAccount(self): #--Method to create a new user account
```

```
85      #--Positioning similar to the login screen as per my screen designs
86      self.createText.grid(row=1, columnspan=9, pady = 130, padx = 330)
87      self.newUsername.grid(row=3, column=3, sticky = E)
88      self.newPassword.grid(row=4, column=3, sticky = E)
89      self.entryNewUsername.grid(row=3, column=4)
90      self.entryNewPassword.grid(row=4, column=4)
91      self.submitNew.grid(row=4, column = 5, sticky = W)
```

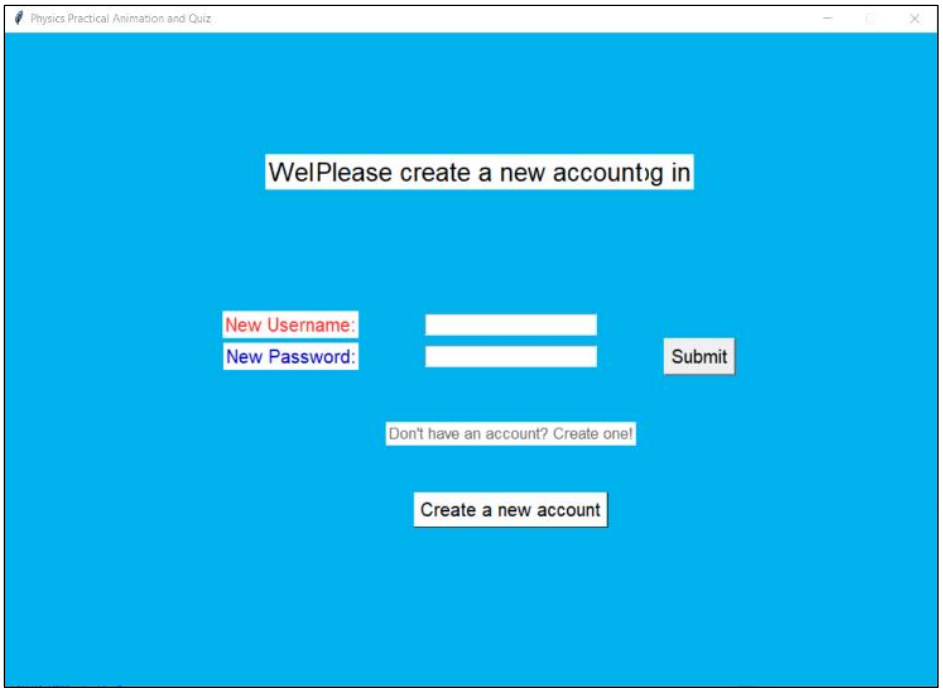
Line 59 displays the 'submit' button. When pressed, it will initiate a command which will add the entered details to a file that I will detail later.

If this program was to be run now, it wouldn't run properly since I have ignored the fact that the user has to press the 'self.createButton' for this new screen to be displayed. Therefore, when the button is pressed, the 'createAccount' method will be called.

```
command=self.createAccount
```

```
50      self.createButton = Button(window, text="Create a new account", bd = form.bd, fg = form.black, bg=form.white, font=(form.font, form.fontSizeMedium),
51                                                                                                                  command=self.createAccount)
```

However, when run, the result is not what I wanted. Since the widgets from the login screen remain, the new widgets are displayed on top of them. Therefore, the 'grid_forget' command is needed to remove the previous widgets like below.

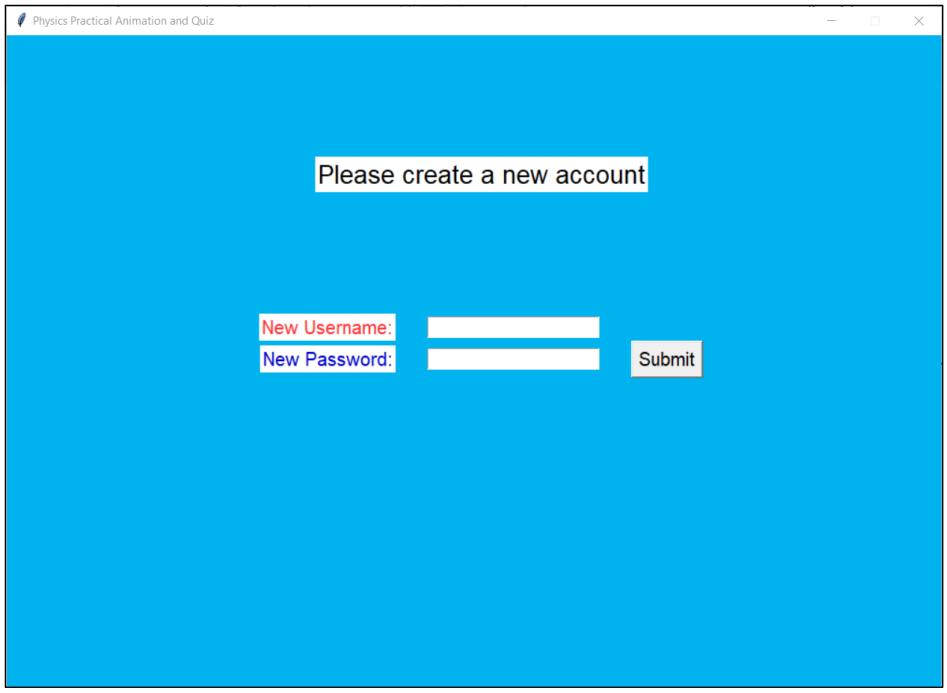



```
79   def createAccount(self): #--Method to create a new user account
80
81     self.accountText.grid_forget() #--Removing widgets that are no longer needed
82     self.createButton.grid_forget()
83     self.retryText.grid_forget()
84     self.welcomeText.grid_forget()
```

Therefore, the final method looks like this:

```
82   def createAccount(self): #--Method to create a new user account
83     self.accountText.grid_forget() #--Removing widgets that are no longer needed
84     self.createButton.grid_forget()
85     self.retryText.grid_forget()
86     self.welcomeText.grid_forget()
87
88     #--Positioning similar to the login screen as per my screen designs
89     self.createText.grid(row=1, columnspan=9, pady = 130, padx = 330)
90     self.newUsername.grid(row=3, column=3, sticky = E)
91     self.newPassword.grid(row=4, column=3, sticky = E)
92     self.entryNewUsername.grid(row=3, column=4)
93     self.entryNewPassword.grid(row=4, column=4)
94     self.submitNew.grid(row=4, column = 5, sticky = W)
```

Resulting in this screen design:



This matches my screen design and I am happy with it as it is functional and clean.

Appending details to a CSV file:

The next part is appending the student's entered details to a CSV file. I chose the CSV (Comma Separated Values) format for many reasons. Since I will not be storing large amounts of data, it is more suited than a database and the fact that I can use Microsoft Excel to view the files makes it easier to work with.

Below, is the method that will be used to append details to the 'studentCredentials' file. Once, the student clicks 'submit' this method will be called.

```
98    def addCredentials(self): #--Appending entered credentials to a csv file
99        rows = [self.entryNewUsername.get(),self.entryNewPassword.get()]#---Items that will be added
100       #---Appending to the file as I want the original contents to be stored
101       with open('studentCredentials.csv','a', newline='') as studentCredentials:
102         writer = csv.writer(studentCredentials)
103         writer.writerows(rows) #--Rows written
```

studentCredentials 01/09/2020 20:25 Microsoft Excel Com... 1 KB

I am appending rather than writing as the data in this file will need to be permanent– writing to this file would remove all the data that is already in it meaning the student cannot log in again.

On line 99, I have used the 'get()' method to retrieve what is in the entry boxes and then I have stored it in a 2D array.

When I ran this, the Excel file had the username and password on different rows with the letters split across columns. This is not ideal as I planned for each row to contain the username and password of every student, which would make up each array of the 2D array.

	A	B	C	D	E	F	G	H
1								
2	U	s	e	r	n	a	m	e
3	P	a	s	s	w	o	r	d
4								

To solve this, I added an extra pair of brackets:

```
98  def addCredentials(self): #--Appending entered credentials to a csv file
99      rows = [[self.entryNewUsername.get(),self.entryNewPassword.get()]]#---Items that will be added
100     #---Appending to the file as I want the original contents to be stored
101     with open('studentCredentials.csv','a', newline='') as studentCredentials:
102       writer = csv.writer(studentCredentials)
103       writer.writerows(rows) #--Rows written
```

One thing that confused me was that the username and password were stored on line 2 instead of line 1. After running some tests, I couldn't figure it out, but it shouldn't matter all that much. When the program is checking whether the entered details match with what is in the file, the empty boxes should be skipped.

	A	B	C	D
1				
2	Username	Password		
3				

Line 62 displays the 'create' button. When pressed, it will initiate the below command which will allow the user to create a new account using the 'addCredentials' method.

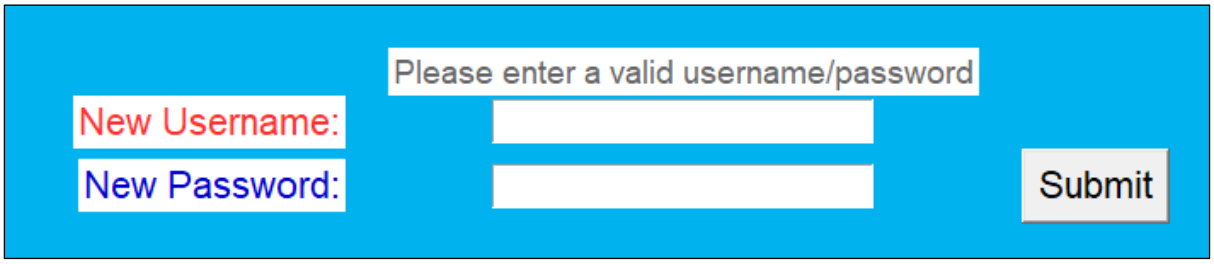
```
command=self.addCredentials
```

```
62    self.submitNew = Button(window, text="Submit", fg=form.black, font=(form.font, form.fontSizeMedium), command=self.addCredentials)
```

Soon after, I realised that some sort of validation was required since the user could create an account with no characters. To make sure the user enters something, the program will check the length of the inputted characters and so long as this is greater than 0, the account will be created successfully.

```
91  def addCredentials(self): #--Appending entered credentials to a csv file
92    #--Part of validation where at least one character has to be entered to be saved
93    if len(self.entryNewUsername.get()) == 0:
94      if len(self.entryNewPassword.get()) == 0:
95        self.enterText.grid(row=2, column = 4)
96        command = self.addCredentials #--Repeats Method
```

On line 95, I have introduced a new attribute called 'self.enterText'. This was instantiated in the Login class and is a text widget telling the user that their username or password is too short.



Line 96 allows the user another go at entering their details.

For the user to be returned to the login screen after creating a new account, two things were needed:

Using the grid_forget() option, I needed to remove widgets that didn't belong in the original login screen and second, I needed to use a command to return the user to the login screen.

The below code illustrates both of those solutions:

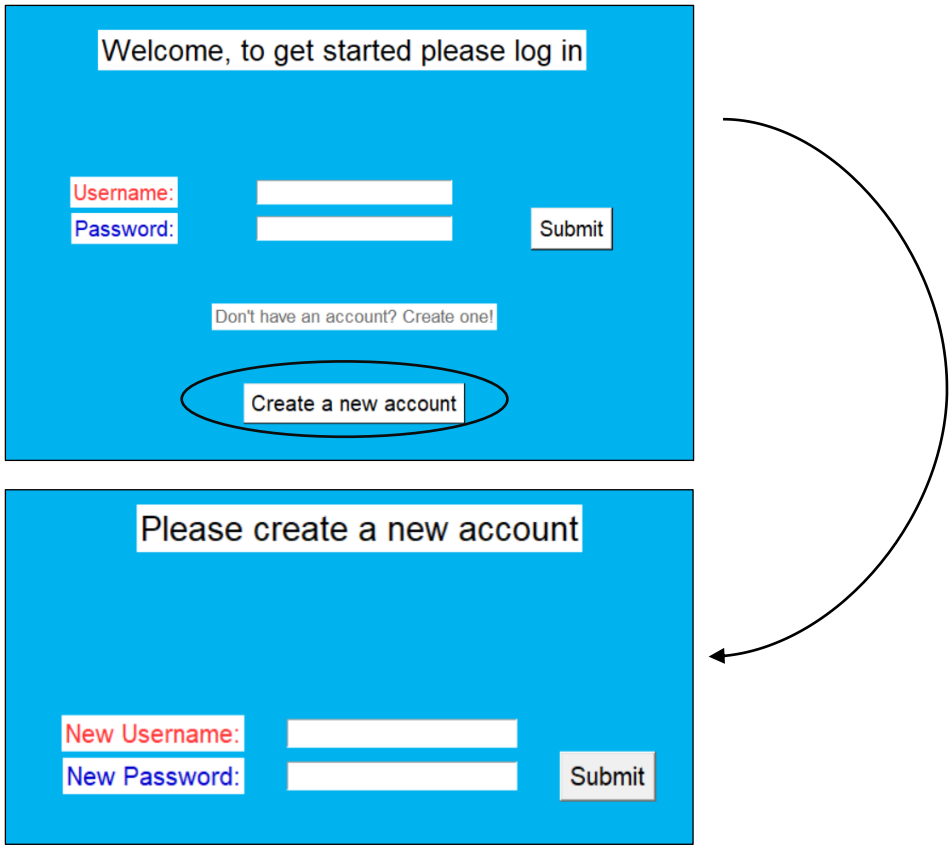
```
108         self.newUsername.grid_forget()
109         self.newPassword.grid_forget()
110         self.enterText.grid_forget()
111         self.createText.grid_forget()
112         command = Login() #--Taking user back to main login screen
```

Development Testing:

Test 1.7

To begin with, I will be testing the creating account functionality of my program. Therefore, I will need to begin with test 1.7 to enable the user to access the correct screen.

Create Account Button	Valid input – button is pressed	Allows the student to move to a new screen where they can create a new account	The student can create a new account	1.7



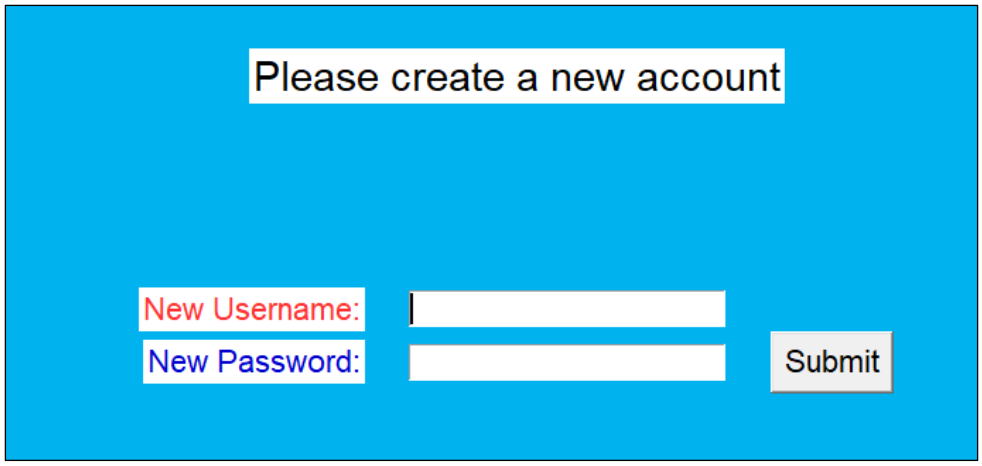
When the button was pressed, it initiated the 'createAccount' method which drew the correct screen. This test was a success.

Tests 2.0 and 2.2

Username	Invalid input– no characters are entered	Username must be rejected and re-entered	To ensure a suitable username is saved	2.0
Password	Invalid input– no characters entered	Password must be rejected and re-entered	To ensure a suitable password is saved	2.2

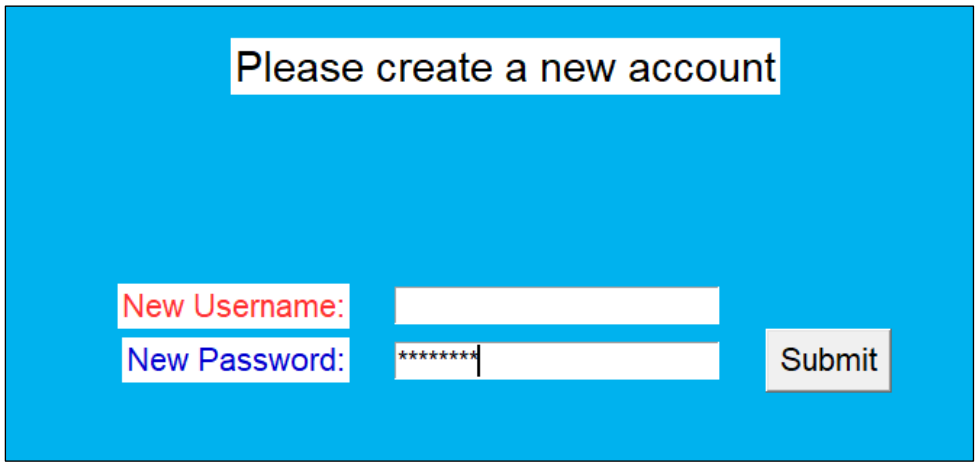
Test 2.0

Entering no username or password:

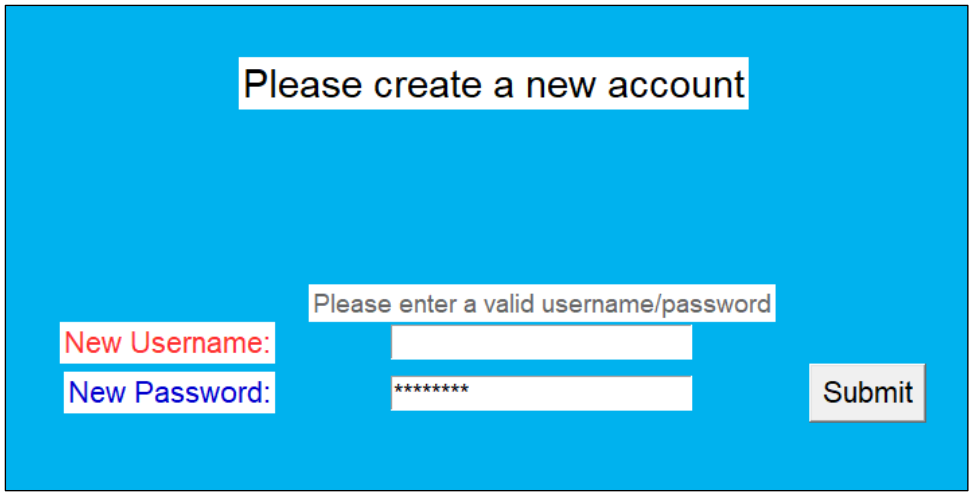


Result:

Entering only a password:

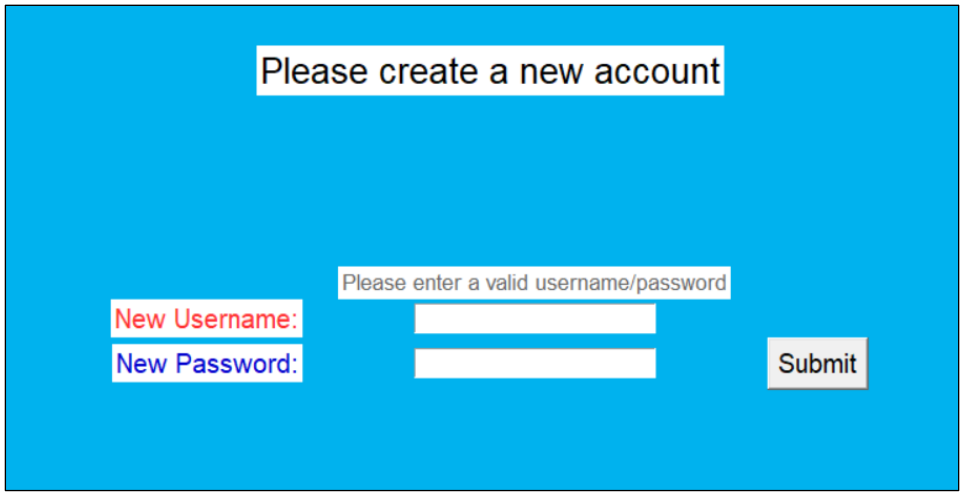


Result:



The username and password are rejected. However, for Test 2.0 to be successful, the user must be able to enter their details again. I have written some code which deletes the contents of the entry boxes if the username or password is invalid.

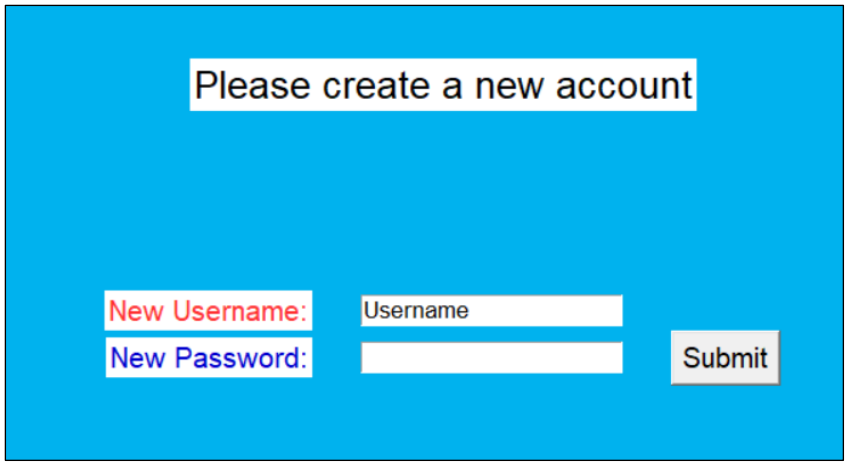
```
101         self.entryNewUsername.delete(0, 'end') #--Deletes content in widgets
102         self.entryNewPassword.delete(0, 'end')
```



Therefore, test 2.0 is a success.

Test 2.2

If a password is not entered but a username is:



Resulting Excel file:

	A	B	C
1			
2	Username		
3			
4			

Here, I faced an unsuccessful test as the username was accepted and written to a file. This should not have happened as no password was entered. When reading back through my code, I discovered the problem:

```
94    def addCredentials(self): #--Appending entered
95      if len(self.entryNewUsername.get()) == 0: #-
96        if len(self.entryNewPassword.get()) == 0:
```

Using this double 'if' statement, the program will only reject usernames or passwords with no characters when the username contains no characters. If not, then the second 'if' statement is effectively bypassed.

To solve this, I will use an 'or' statement so that if either the username or password contains no characters, access will be denied.

```
95      #--Part of validation where at least one character has to be entered to be saved
96      if len(self.entryNewUsername.get()) == 0 or len(self.entryNewPassword.get()) == 0:
```

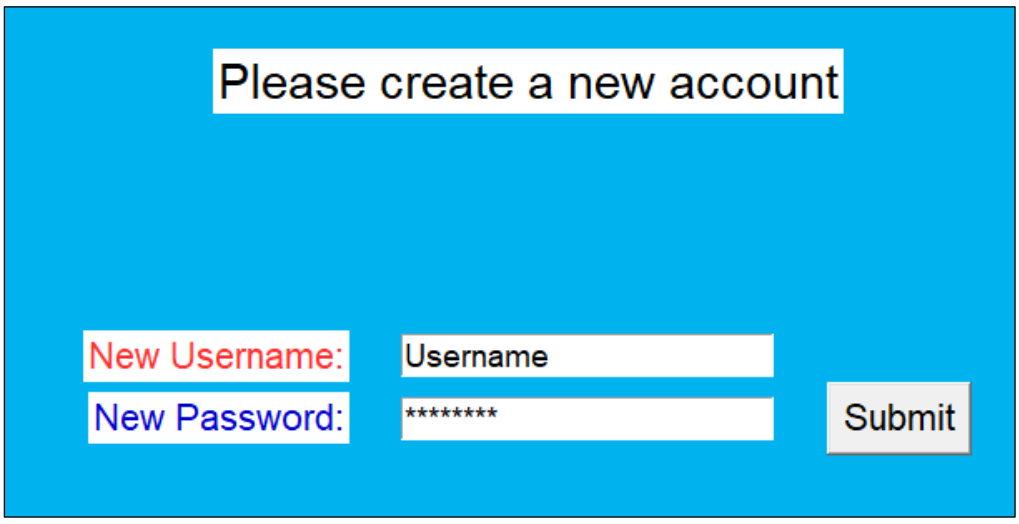
Therefore, when entering only a username or only a password, this was the result:

I repeated test 2.0 where no username or password was entered and the results were the same. Consequently, test 2.2 was successful since a message was displayed telling the user to re-enter their details, which they can do.

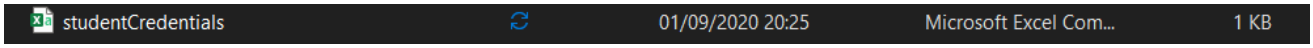
Password	Invalid input– no characters entered	Password must be rejected and re-entered	To ensure a suitable password is saved	2.2

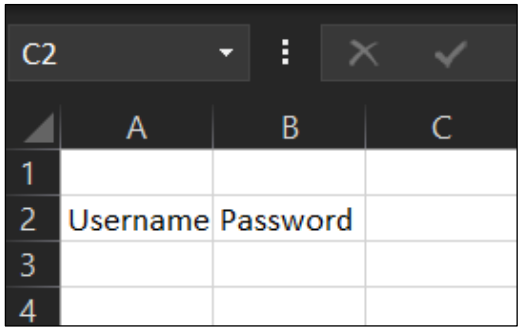
Tests 2.1 and 2.3:

Valid input – valid username entered	Username is accepted and will be written to a file	To ensure a suitable username is saved	2.1
Valid input – valid password entered	Password is accepted and will be written to file	To ensure a suitable password is saved	2.3



When a valid username and password is entered, the file is updated with the new contents.



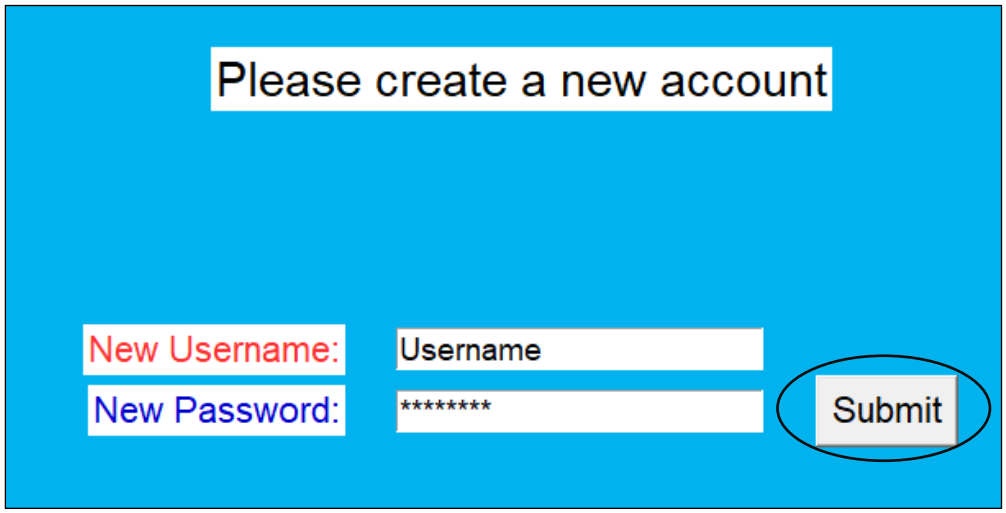


Tests 2.1 and 2.3 are successful since the username and password have been written to a file

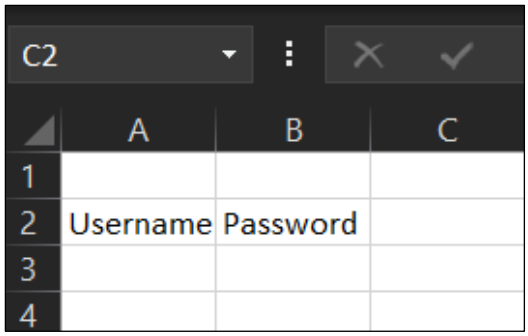
Test 2.4

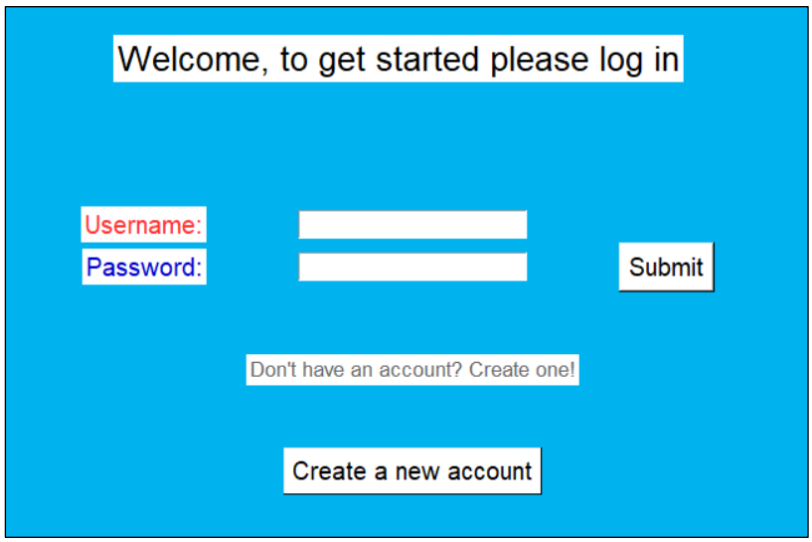
Submit Button	Valid input – button is pressed	Username and password will be successfully written to file and then the user will be returned to the login screen.	Allows the user to log in with these credentials	2.4

For the above tests to be successful, they rely on test 2.4 which is the 'submit' button, which should be pressed to save the username and password to a file.



When pressed, the 'submit' button must save the username and password to a CSV file and return the user to the login screen.





Both were executed successfully so test 2.4 is a success.

Validation

As part of my development testing, it is important to validate the user's inputs so the program accepts only the right input.

Result	Valid input – Entering a valid username and password	Invalid input – Not entering anything in the entry fields	Boundary input – If only one entry field is filled out
Expected output?	Yes, the username and password were saved and could be used to log the user in.	Yes, the program rejected this input and allowed the user another go.	Yes. After resolving test 2.2, my program rejects these boundary inputs.
Test number	2.4	2.0	2.0 and 2.2

Checking entered login details:

To begin with, I need to validate what the user has entered. Therefore, I will check whether the user has entered anything, otherwise, they will be prompted to do so.

```
150     #--Makes sure usernames and passwords with no characters cannot be entered
151     if len(self.entryUsername.get()) == 0 or len(self.entryPassword.get()) == 0:
152         self.enterText.grid(row=2, column = 4)
```

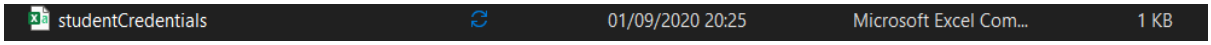
In order to compare the user's entry with the values in the CSV file, I will append the file items to a 2D array.

Line 37 shows the list that will be used to store student details.

```
37    self.credentialsList = [] #--Lists of all the student details to be used later
```

Below, I have opened the correct file so I can read from it and append its contents to the 'credentials List', as a 2D array.

```
149         with open('studentCredentials.csv', 'r') as studentCredentials:
150           reader = csv.reader(studentCredentials)
151           for row in reader: #--File contents are appended as a 2d array
152             self.credentialsList.append(row)
```



Below, is my first attempt at validating the student's entered username and password. On line 154, I have used a for loop to iterate through every array in the 2D array. For access to be granted, each array must have the correct username and correct corresponding password. If this is not met, then the user will have to re-enter their details.

```
154         for x in self.credentialsList: #--Goes through each array in the 2d array
155           if self.entryUsername.get() in x and self.entryPassword.get() in x:
156             print("Access Granted")
157
158           else:
159             print("Please retry")
```

Below is the result of the Python shell when a valid username and password was entered:

```
Access Granted
```

While this worked when there were a single username and password in the file, it did not when there were multiple usernames and passwords in the file as shown below.

	A	B	C
1	Abbas	Password1	
2	Joe	Password2	
3	Tom	Password3	
4			

The result of the Python shell:

```
Access Granted
Please retry
Please retry
```

This is what the 'self.credentialsList' attribute looks like:

```
[['Abbas', 'Password1'], ['Joe', 'Password2'], ['Tom', 'Password3']]
```

Seemingly, the program would check the first array to see if the credentials matched. If it did not, then access would be denied even though the correct details may be further down the array.

To solve this, I needed a way for the program to loop through the entire array before deciding whether access should be given or not. Therefore, I introduced a new Boolean attribute called 'self.correct' which is set to False.

```
148         self.correct = False #--Attribute to determine whether access is given
```

```
154         for x in self.credentialsList: #--Goes through each array in the 2d array
155           if self.entryUsername.get() in x and self.entryPassword.get() in x:
156             self.correct = True
157             print("Access Granted")
158
159         if self.correct !=True:
160           print("Please retry")
```

Therefore, in the Python shell, only one result was printed.

In the screenshot below, lines 150 to 154 show how the CSV file will be read and then written to a list. Lines 156 to 167 show the validation implemented to ensure that only valid entry details are accepted.

```
149     else:
150       self.correct = False #--Attribute to determine whether access is given
151       with open('studentCredentials.csv', 'r') as studentCredentials:
152         reader = csv.reader(studentCredentials)
153         for row in reader: #--File contents are appended as a 2d array
154           self.credentialsList.append(row)
155
156         for x in self.credentialsList: #--Goes through each array in the 2d array
157           if self.entryUsername.get() in x and self.entryPassword.get() in x:
158             print("Correct Credentials")
159             self.correct = True #--Sets attribute to True so access is granted
160
161         if self.correct !=True:
162           loginTries += 1 #--Adds one to the loginTries count
163           self.entryUsername.delete(0, 'end') #--Deletes content in widgets
164           self.entryPassword.delete(0, 'end')
165           self.enterText.grid_forget()
166           self.retryText.grid(row = 2, column = 4)
167           command = self.removeWidgets #--Sends user back to the login screen
```

On line 166, a new attribute called 'self.retryText' has been referenced. This was instantiated in the Login class:

```
69      self.retryText = Label(window, text="Wrong credentials, please try again", fg=form.grey,bg=form.white,
70                                                                  font=(form.font, form.fontSizeSmall))
```

On line 167, I have called the 'removeWidgets' method which removes widgets from the window when the user enters the wrong details. This allows the user to return to the login screen. Ideally, I

would use this: command = Login(), however, the program wouldn't correctly remove the 'enterText' and 'retryText' widgets. Therefore, I chose to call a new method to execute it:

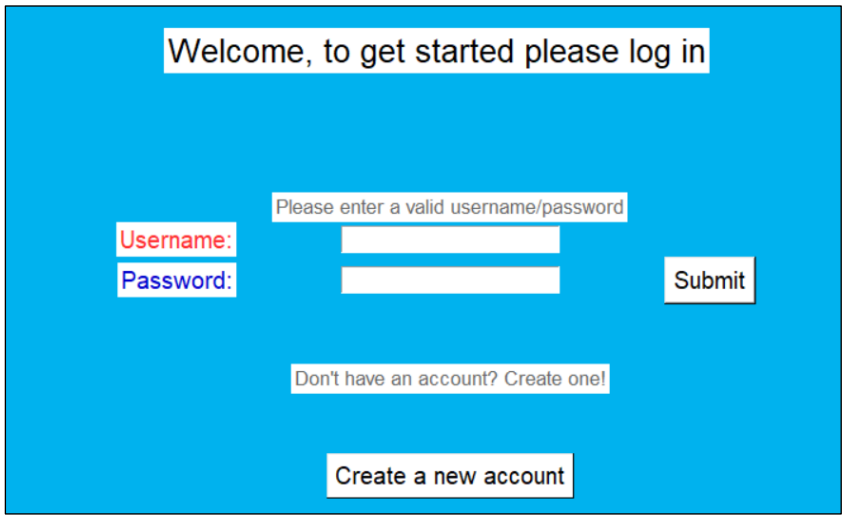
```
168   def removeWidgets(self):
169     self.retryText.grid_forget()
170     command = Login() #--Takes user back to the original screen
```

Development testing:

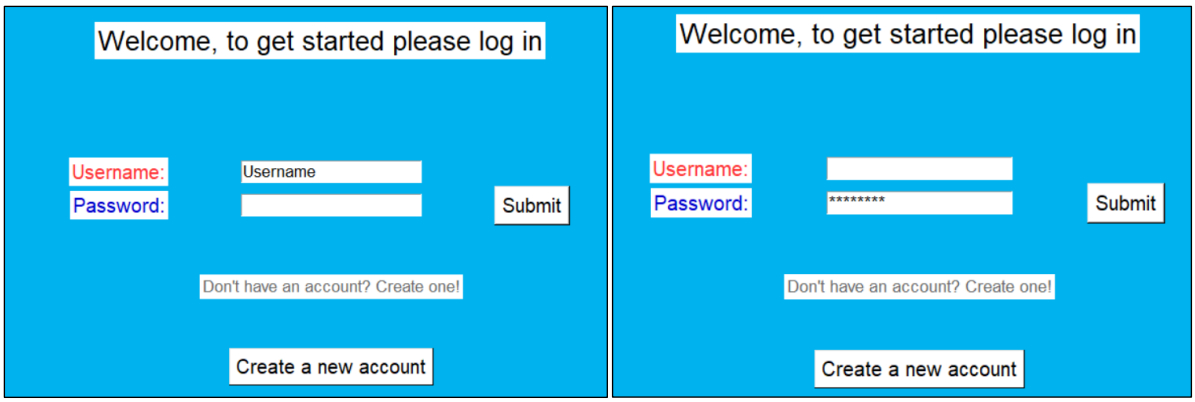
Tests 1.0 and 1.3

Username	Invalid input – no characters or wrong username entered	Username does not match the stated value in the CSV file so must be rejected and re-entered	To ensure authorised access	1.0
Password	Invalid input– no characters or wrong password entered	Password does not match the stated value in the CSV file so must be rejected and re-entered	To ensure authorised access	1.3

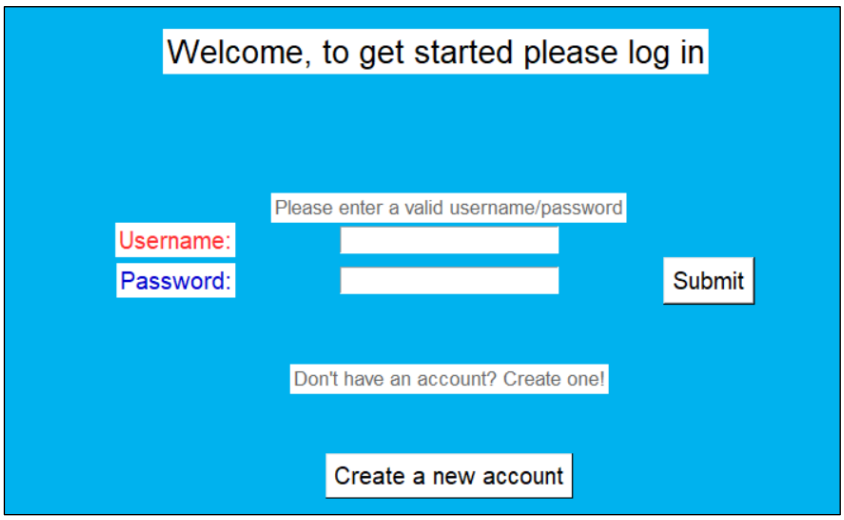
When both the user's username and password are not entered:



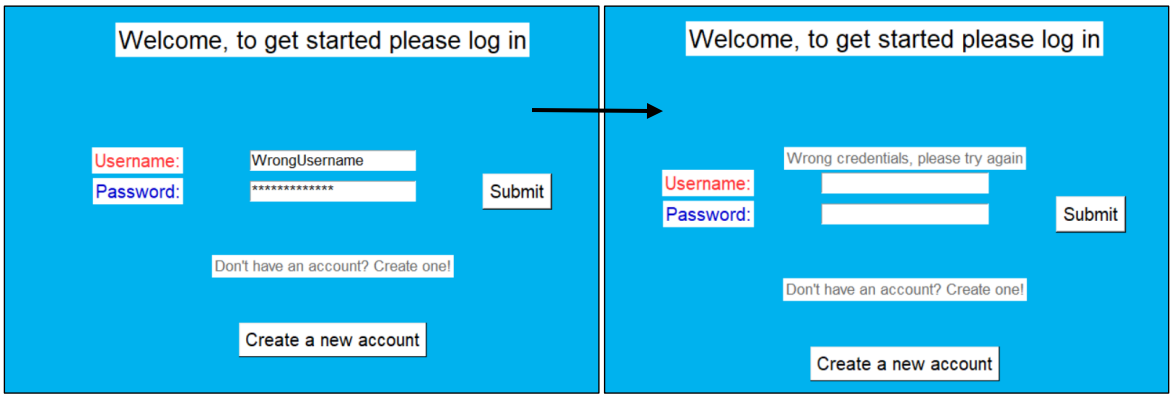
When only a username or only a password is entered:



The resulting screen:



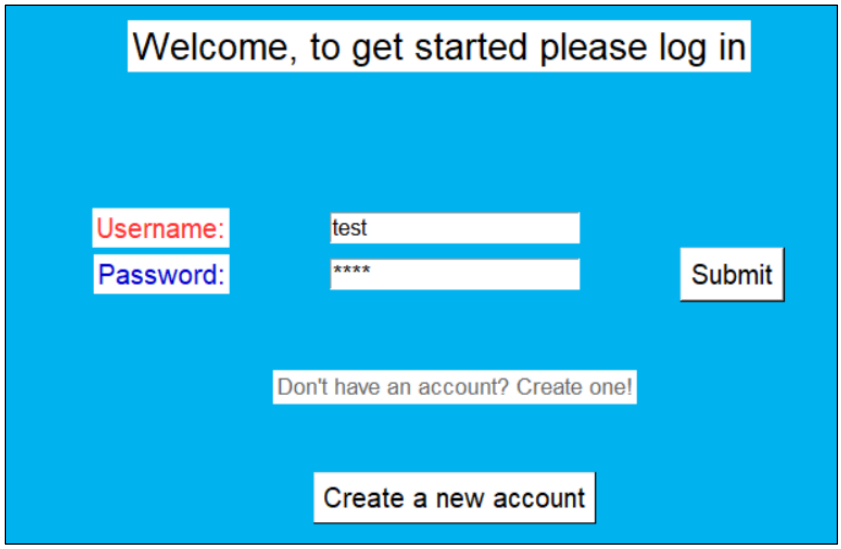
Part of the test is to handle erroneous data inputs:

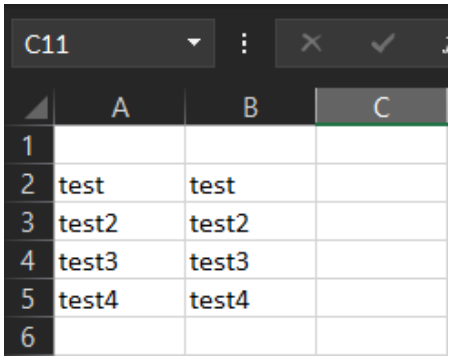


Tests 1.0 and 1.3 were a success since my program did not allow access and forced the user to re-enter their username and password.

Tests 1.2 and 1.5

Valid input – correct username	Username does match the stated value in the CSV file so is accepted	To ensure authorised access	1.2
Valid input – correct password	Password does match the stated value in the CSV file so is accepted	To ensure authorised access	1.5





C11

	A	B	C
1			
2	test	test	
3	test2	test2	
4	test3	test3	
5	test4	test4	
6			

Since the entered username and password match with what is in the file, the user can successfully login. Tests 1.2 and 1.5 were a success.

```
Correct Credentials
```

Validation

Result	Valid input – Entering a valid username and password	Invalid input –Entering incorrect credentials	Boundary input – If only one of the entry fields is filled out
Expected output?	Yes, the user was able to log in.	Yes, the program rejected this input and	Yes, my program rejected this input and

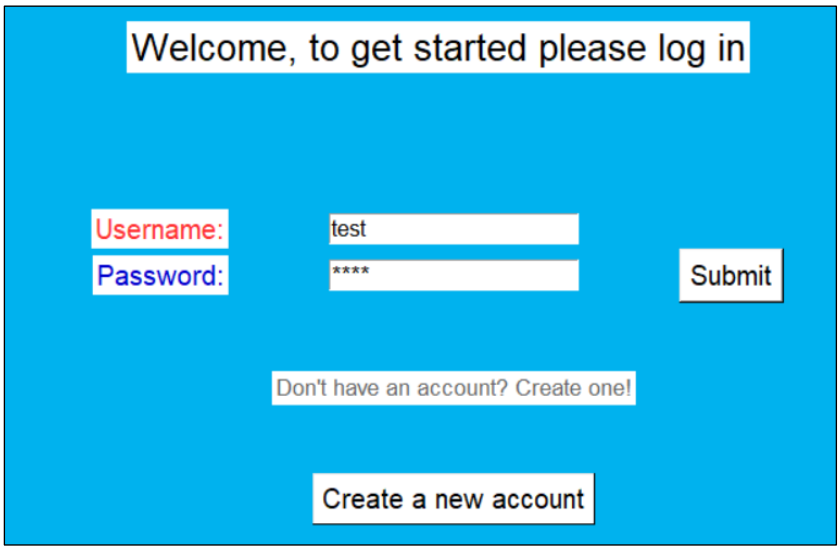
		allowed the user another go.	allowed the user another go.
Test number	1.2 and 1.5	1.0 and 1.3	1.0 and 1.3

Test 1.6:

Submit Button	Valid input – button is pressed	Allows the user to successfully login or to re-enter their details	To ensure that the student can log in successfully	1.6

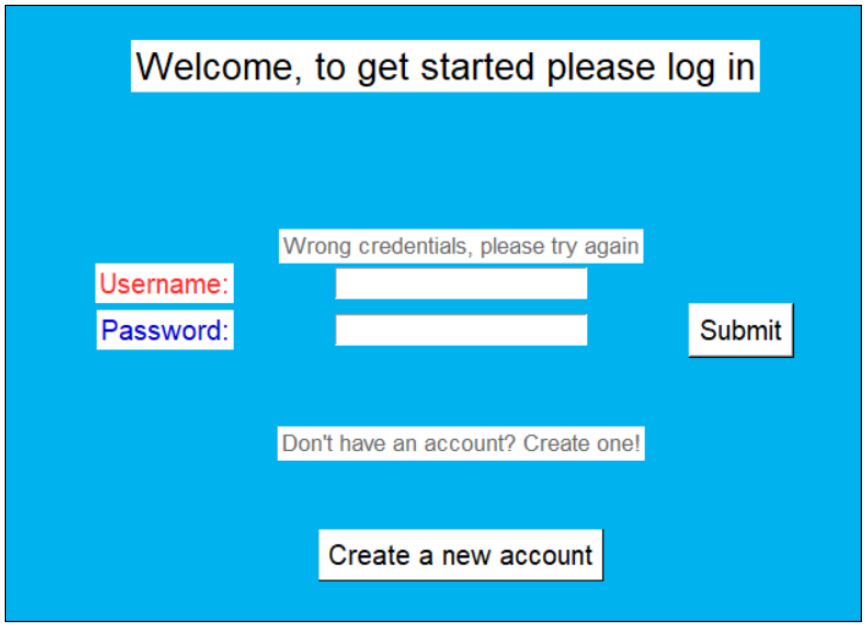
For the above tests to work, I need to test the 'submit' button so that it transitions the user to the correct screen. When pressed, it should allow the user to log in otherwise they have to re-enter their details.

Entering the correct details:



Correct Credentials

Entering incorrect details:



I need to make sure that the user can re-enter their details again after their first try. When entering correct details for the second time, the Python shell printed this out twice instead of once, which was unexpected.

```
Correct Credentials
Correct Credentials
```

Since the program would transition to the next screen, this would not matter, however, I feel that it still needs to be fixed.

When looking back at the program, I found that when I appended the file items to a list, after a second try (since the class was being repeated), the items would be appended to the same list creating duplicates:

After the first try:

```
[['Username', 'test'], ['Mayur', 'test'], ['test', 'test']]
```

After the second try:

```
[['Username', 'test'], ['Mayur', 'test'], ['test', 'test'], ['Username', 'test'], ['Mayur', 'test'], ['test', 'test']]
```

Therefore, the solution would be to clear the list every time:

```
164             self.credentialsList.clear()
```

In conclusion, test 1.6 was successful.

Countdown Timer

As part of my success criteria, I need to limit the login entries to 5 tries. Once this limit has been met, a timer would countdown the remaining time before they could have another go.

I have introduced a new variable called 'loginTries' and have set it to 0 representing 0 login attempts.

```
16 loginTries = 0
```

Below, on line 114, I have set this variable to be global so its value can be changed anywhere.

```
113   def checkCredentials(self): #--Method to check the inputted details
114     global loginTries #--Global variable which keeps track of how many attempts the user has had
115
116     if loginTries > 4: #--Giving the user 5 attempts as per my success criteria
```

The code for the countdown timer comes before checking the entered details. This is because the countdown timer is a form of validation which is important for the robustness of my code.

Below, I have coded what will be displayed onscreen. Then I will move onto the loop that iterates through every time value.

```
116     if loginTries > 4: #--Giving the user 5 attempts as per my success criteria
117       self.retryText.grid_forget()
118       self.attemptText.grid(row=2, column=4)
119
120       second=StringVar()
121       second.set("30")
122       timeEntryLabel = Label(window, text="30", width=3, bg = form.white, font=(form.font, form.fontSizeLarge,""),
123                              #--Timer that will be displayed
124                          textvariable=second)
125       timeEntryLabel.grid(row = 1, column = 3)
126       loop = 30 #--Sets the total time in seconds
```

On line 118, the attribute 'self.attemptText' will display the following when the limit of 5 tries has been exceeded:

You have attempted more than 5 tries, please wait 30 seconds

This attribute was instantiated in the Login class.

```
66     self.attemptText = Label(window, text="You have attempted more than 5 tries, please wait 30 seconds",
67                              fg=form.grey,bg=form.white,font=(form.font, form.fontSizeSmall))
```

On line 120, I have set the variable 'second' as StringVar(), which is a Tkinter variable, and have given it a value of 30 which corresponds to 30 seconds.

Then on line 122, I have created a label which will display the time remaining. On line 124, the option 'textvariable' allows the text within the label to change which is more efficient than removing the label and re-displaying it.

30

```
128       while loop >-1: #--Loop allows timer to iterate
129         mins,secs = divmod(loop,60)
130         second.set("{0:2d}".format(secs))
131         window.update() #--Updates window as time changes
132         time.sleep(1)
133         if (loop == 0):
134           timeEntryLabel.grid_forget()
135           self.enterText.grid_forget()
136           self.attemptText.grid_forget()
137
138           loginTries = 0 #--Resets variable so user can enter their details again
139           command = self.removeWidgets #--Sends user back to the login screen
140         loop -= 1
```

On line 129, the operation ‘divmod’, will display the quotient and remainder when dividing one number by another. By doing divmod of the integer ‘loop’ and 60, the variable ‘secs’ will output 30, 29, 28 and so on and ‘mins’ will have a constant value of 0. Looking back, it would have easier to just have used a remainder as the variable ‘mins’ is not needed. However, when faced with larger values of time, divmod is better suited.

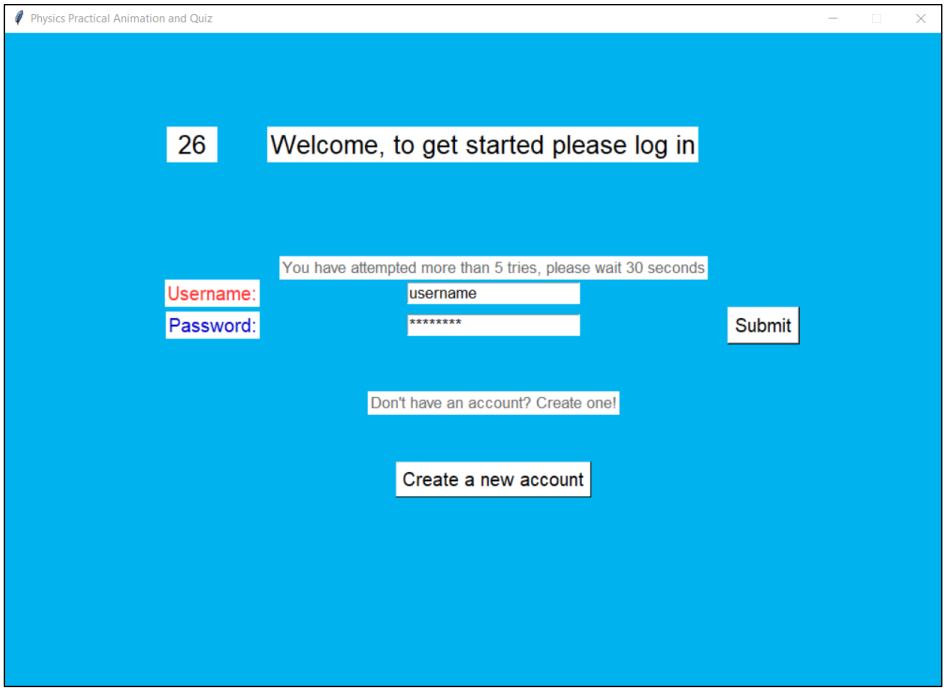
On line 130, I have changed the value of the variable ‘seconds’ and this is reflected through the ‘textvariable’ option on line 124.

Lines 131 updates the window to reflect the changes while line 132 makes the program pause for one second, giving the effect that the countdown timer is decreasing in seconds.

Line 134 to 136 clears the screen before the ‘loginTries’ variable is set back to 0 and the self.removeWidgets method is called, taking the user back to the login screen to re-enter their details.

On line 140, a value of 1 is deducted from the loop variable to continue the loop

Like in my screen designs, this is the screen when a user attempts more than 5 tries:



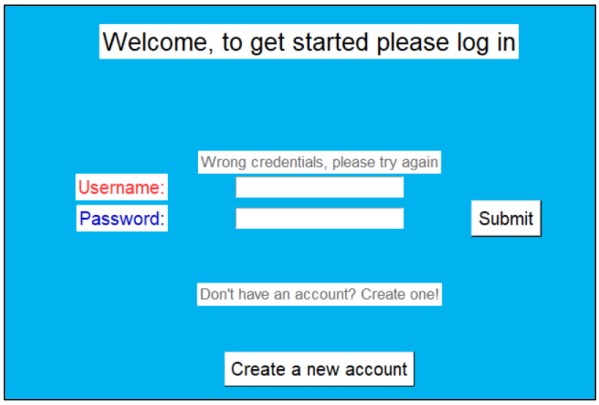
An interesting side effect is when the user attempts to enter something new in the entry boxes while the timer is counting down. The timer resets back to 30 seconds making the countdown start all over again. While I did not intend for this to happen, it is still a welcome feature as it improves the robustness of my program.

Development testing:

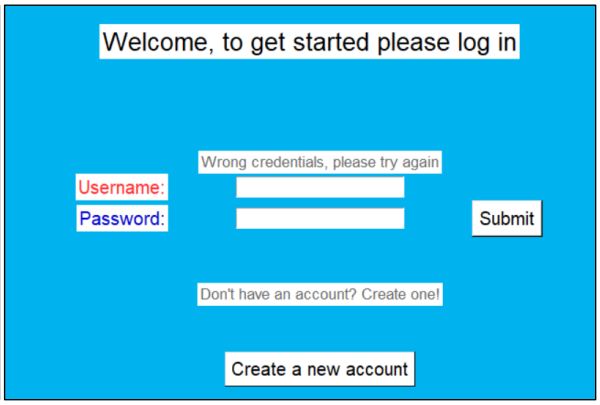
Tests 1.1 and 1.4

Invalid input – and the maximum number of tries has been reached	User will be unable to enter their credentials for some time	To prevent brute-force attacks	1.1
Invalid input – and the maximum number of tries has been reached	User will be unable to enter their credentials for some time	To prevent brute-force attacks	1.4

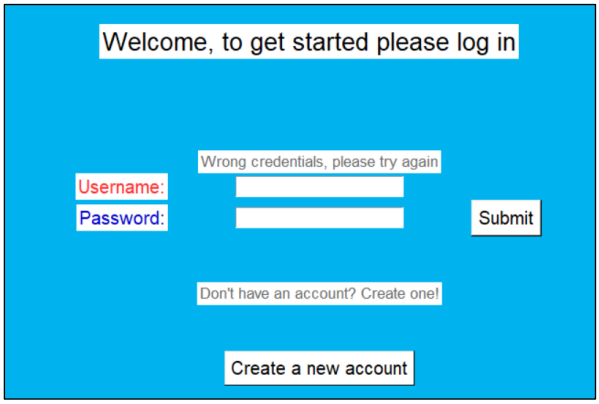
First Try:



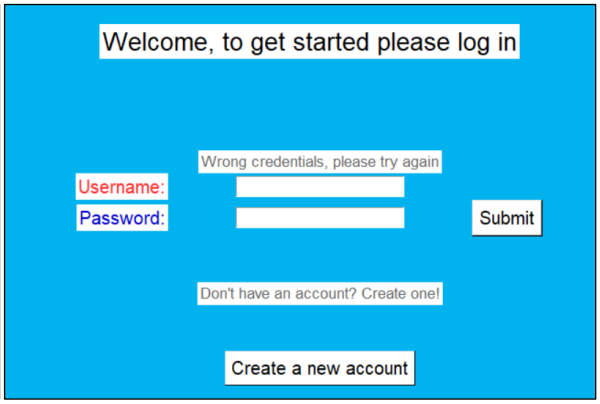
Second Try:



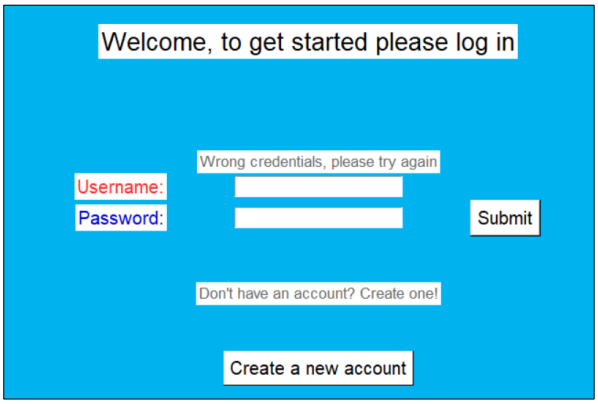
Third Try:



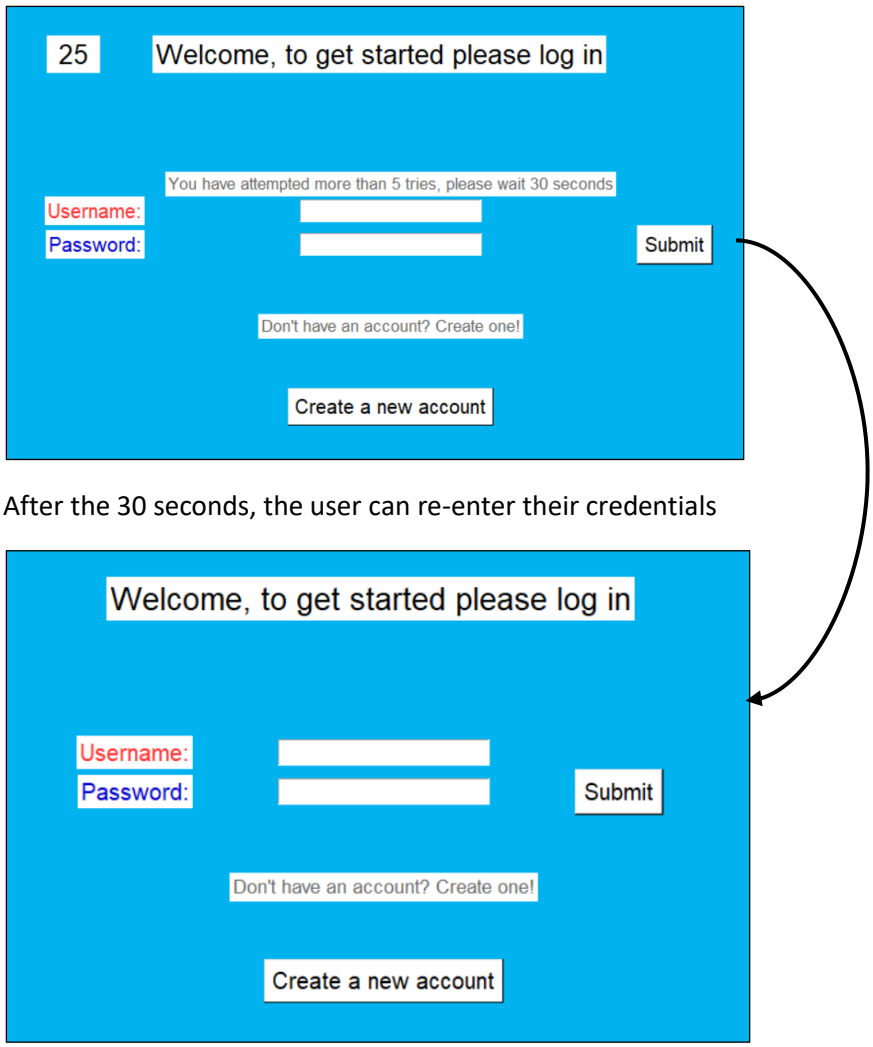
Fourth Try:



Fifth Try:



Once, the user has passed their fifth try, access must be denied.



After the 30 seconds, the user can re-enter their credentials

Tests 1.1 and 1.4 are a success since access has been denied for 30 seconds before a user can enter their details again.

Additionally, this fulfils the second item in my success criteria. My program has now validated the user's inputs when logging in and creating an account.

Choosing a practical:

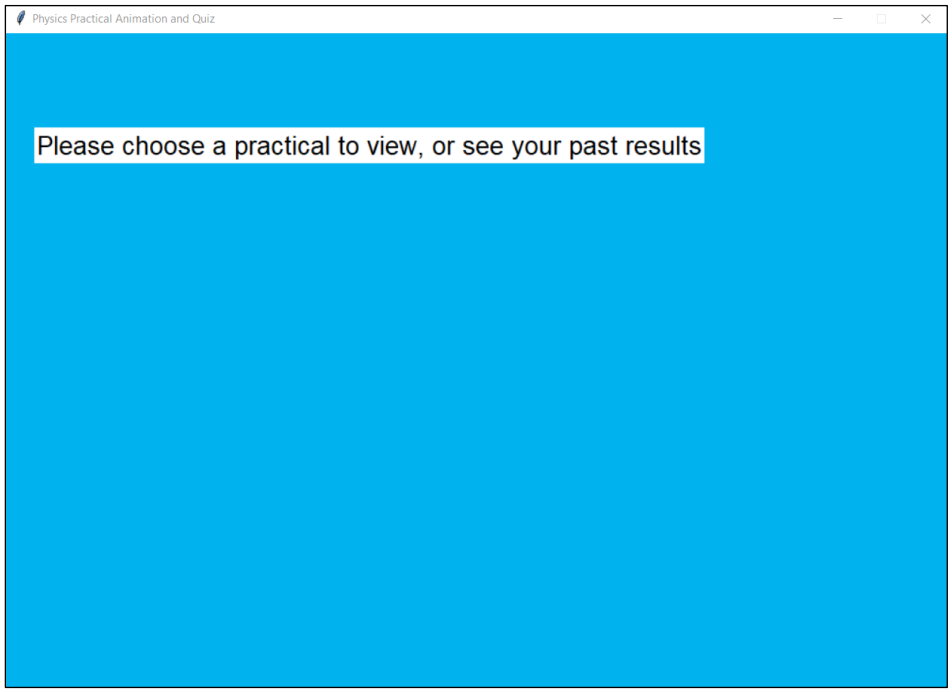
After logging in, the student should now be able to choose a practical to watch. Below, I have removed all the previous widgets to create a blank blue screen.

```
160             self.username.grid_forget()
161             self.password.grid_forget()
162             self.entryUsername.grid_forget()
163             self.entryPassword.grid_forget()
164             self.welcomeText.grid_forget()
165             self.createButton.grid_forget()
166             self.submit.grid_forget()
167             command = choosePractical()
```

On line 167, I have instantiated a new class called ‘choosePractical’ which will allow the user to choose a practical.

Here is the new class where I have created a new heading for the screen called ‘headingText’. Like with my previous screens, I have given this label a columspan of 9 as it will be the main widget on the screen.

```
183 class choosePractical:
184   def __init__(self):
185     #---Main heading in the window
186     self.headingText = Label(window, text="Please choose a practical to view, or see your past results",
187                              fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
188     self.headingText.grid(row=1, columnspan=9, pady = 100, padx = 30)
```



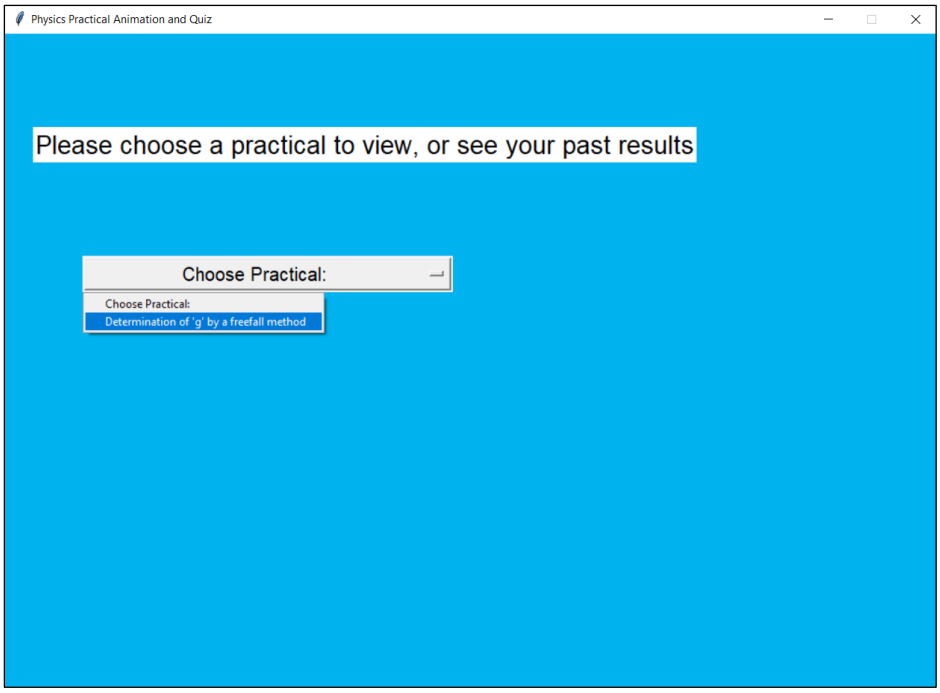
Next, I will need to create a new drop-down menu as per my screen designs, to aid usability. To do this, I will be using the OptionMenu function as part of Tkinter.

On line 202, is the list of practicals that a student can choose to view. By default, this will be set to ‘Choose Practical’ which will not produce a result. In the list below, I have chosen the practical ‘Determination of g by a freefall method’. Due to time constraints, I will be limited to creating one practical but ideally, there would be a larger range of practicals to choose from, especially in further development.

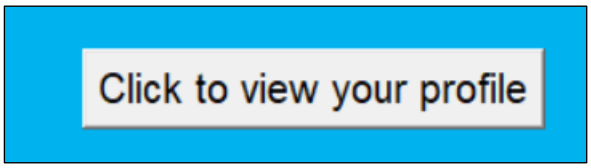
On line 207 and 208, this creates the drop-down menu onscreen.

```
201     #--List of options in the dropdown menu
202     self.OptionList = ["Click to choose Practical:", "Determination of 'g' by a freefall method"]
203     self.menu = StringVar(window) #--Sets up a tkinter variable
204     self.menu.set(self.OptionList[0]) #--First option will be 'Choose Practical'
205     self.dropDown = OptionMenu(window, self.menu, *self.OptionList)
206     #--Formats and positions the dropdown menu
207     self.dropDown.config(width=32, font=(form.font, form.fontSizeMedium))
208     self.dropDown.grid(row=3, column=2)
```

Resulting screen:



Additionally, as per my system diagram, there should be an option for the user to go straight to their 'profile' where they can view their past results. I will first finish the animation and quiz and then return to this screen to add this.



I have put this button in the program as an indicator, but the button has no method attached to it.

For the user to witness the practical, the program must register when the user has clicked on it.

```
199     def change_dropdown(*args): #--Function which runs when the user clicks an option
200       if self.menu.get() == "Determination of 'g' by a freefall method":
201         command = PracticalAnimation() #--Initiates function which contains the 'confirm' button
202
203     self.menu.trace("w", change_dropdown) #--Allows the function to be run
```

Once the user selects an option from the drop-down menu, the program needs to open the actual animation window as shown by line 201.

On line 199 I have passed arguments using the special keyword '*args'. This passes a variable number of arguments to a function.

On line 203, I have used the trace option as part of Tkinter to add a variable observer which will track the 'self.menu' attribute. The 'w' signifies that the function 'change_dropdown' will be called when the variable is selected by the user.

When testing this, I noticed that the subsequent animation window would open immediately when the practical was selected from the dropdown menu. I decided to add a 'confirm' button to make the transition smoother. This links to my screen designs and the usability of my program as the user could unintentionally click on an option in the drop-down menu.

On line 202, I will call a new method called ‘confirmButton’

```
201          #--Initiates function which contains the 'confirm' button
202          command = self.confirmButton()
```

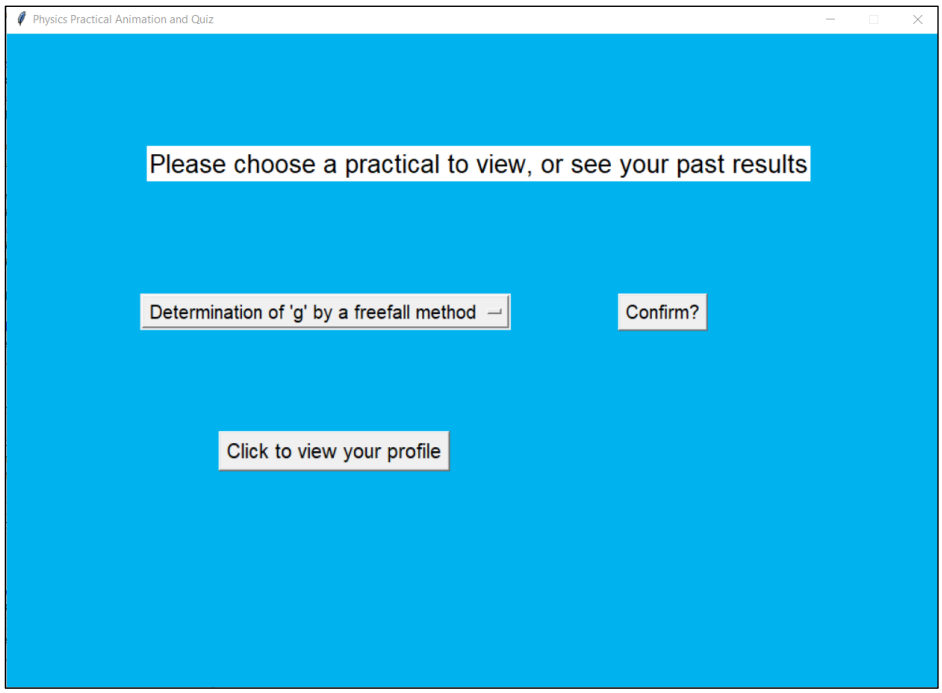
```
205  def confirmButton(self): #--Allows the user to confirm that they want to view the practical
206    self.confirm_Button = Button(text="Confirm?",fg=form.black, font=(form.font, form.fontSizeMedium), command = self.practical)
207    self.confirm_Button.grid(row=3, column=3)
```

On line 206, when the button is clicked a new method will be called. Ideally, I would instantiate ‘PracticalAnimation()’ here, however, I ran into trouble. When the practical was selected from the drop-down menu, the new window would immediately open, before the user would have the chance to click the ‘confirm’ button.

```
209    def practical(self): #--Instantiates the PracticalAnimation class which creates  new window
210      command = PracticalAnimation()
```

Therefore, the method above will run the PracticalAnimation class when the ‘confirm’ button is clicked.

After centralising the widgets by increasing the column values for the grid placements, the resulting screen now looks like this:



Since I will not be doing anything with the profile button for now, I will ignore it in my future screens.

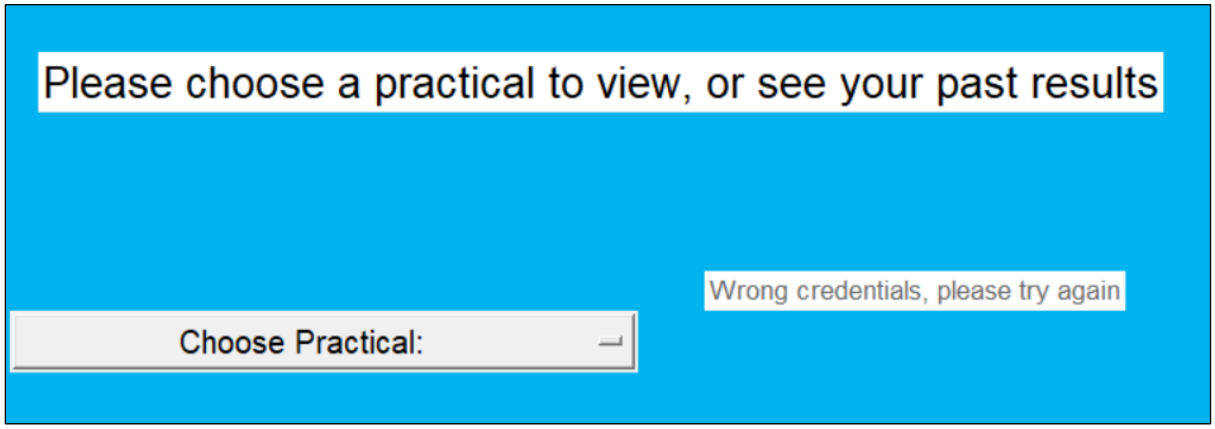
Development testing:

Test 1.6

Submit Button	Valid input – button is pressed	Allows the user to successfully login or to re-enter their details	To ensure that the student can log in successfully	1.6

As a continuation from before, I will document whether the screen change initiated by the 'submit' button has been successful.

When clicking it, having entered correct credentials, the screen looks like this when the student has successfully logged in:



I need to use the forget command for one of the widgets. Then this test is successful.

```
168                 self.retryText.grid_forget()
```

Practical Animation:

When the practical is selected and the confirm button is pressed, the program must open a new animation window. I will be using Pygame to create the practical: 'Determination of 'g' by a freefall method'. I will create a new window separate from the Tkinter window.

Setting up the new window

First, I will initialise Pygame. This is so I can use any Pygame module that I may need in the future:

```
  7 pygame.init() #--Initialises pygame
```

```
249 class PracticalAnimation:
250     win = pygame.display.set_mode((600, 600))
251     pygame.display.set_caption('Practical Animation')
252     win.fill((255,255,255))
253 
254     for event in pygame.event.get():
255         if event.type == pygame.QUIT:
256             pygame.quit()
257             sys.exit()
258     pygame.display.update()
259 
260 command = PracticalAnimation()
```

Lines 250 to 252 create a screen with a size of 600 x 600, a title of 'Practical Animation' and a background colour of white.

Lines 254 to 257 allow the program to check whether the window has been exited. If Pygame detects this, the program is terminated. On line 258, the window will be updated allowing it to be seen by the user.

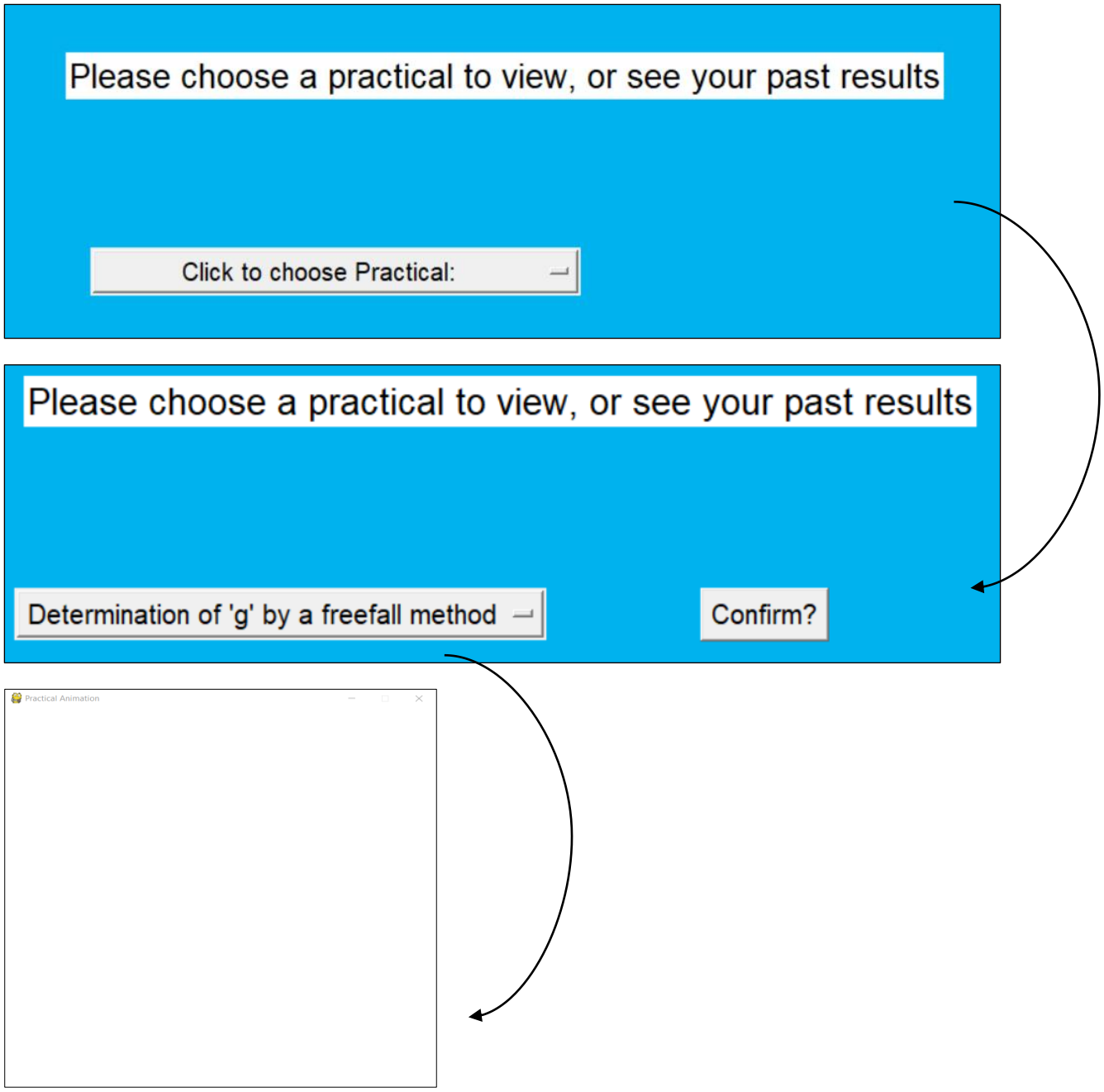
This produces my first Pygame window.

Development testing:

Test 3.0

Choose Practical Button	Valid input – button is pressed	Open a new screen containing the animations	Allows the user to choose an animation to watch	3.0

When the student clicks on the dropdown menu and chooses an option, a new screen containing the animation should be displayed.



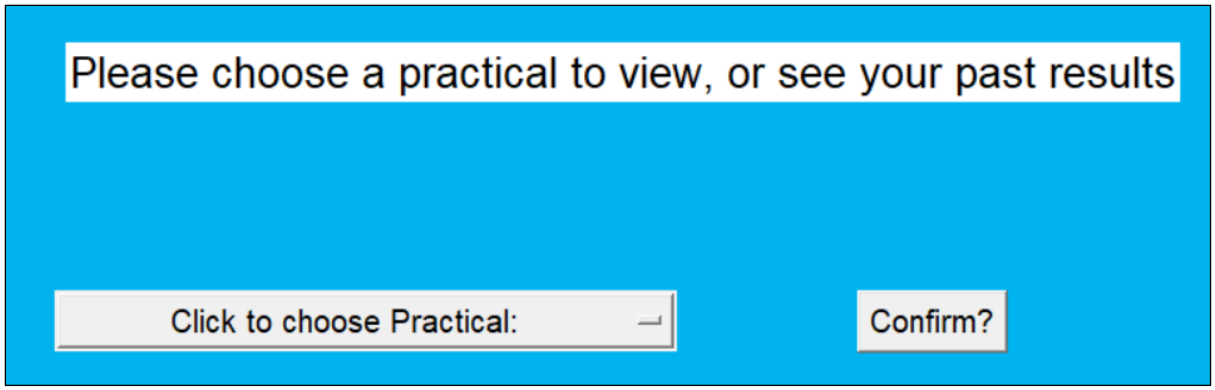
This was the case and therefore this test was a success.

Validation:

Result	Valid input – Choosing a practical from the drop-down menu	Invalid input – Not clicking the drop-down menu	Boundary input – Switching between items in the drop-down menu
Expected output?	Yes, a new window containing the animation was displayed	Yes, since there was no input, there was no screen change	No, I encountered an error
Test number	3.0	3.0	Evidence is below

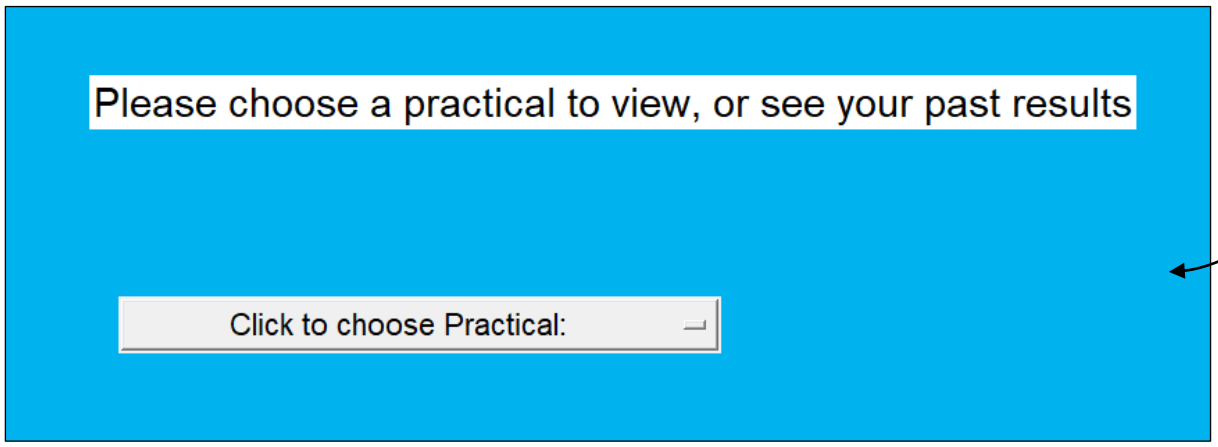
Boundary input:

When I attempted to click back on the 'Choose Practical' item after the Pygame had already been opened, I found that the 'confirm' button remained and when the user clicked on it, it would again open the Pygame window. This should not happen as only when a practical is selected should the Pygame window open.



To rectify this, in the 'change_dropdown' function, I added an if statement where if the student clicks on the 'choose practical' item again, the button will be removed.

```
211         elif self.menu.get() == "Click to choose Practical:":
212             self.confirm_Button.grid_forget()
```



The whole code now looks like this:

```
class ChoosePractical:
  def __init__(self):
    #---Main heading in the window
    self.headingText = Label(window, text="Please choose a practical to view, or see your past results",
                             fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
    self.confirm_Button = Button(text="Confirm?",fg=form.black, font=(form.font, form.fontSizeMedium), command = self.practical)
    studentProfile_Button = Button(text="Click to view your profile",fg=form.black, font=(form.font, form.fontSizeMedium))
    studentProfile_Button.grid(row=4, column=2, padx = 0, pady = 100,)

    #--List of options in the dropdown menu
    self.OptionList = ["Click to choose Practical:", "Determination of 'g' by a freefall method"]
    self.menu = StringVar(window) #--Sets up a tkinter variable
    self.menu.set(self.OptionList[0]) #--First option will be 'Choose Practical'
    self.dropDown = OptionMenu(window, self.menu, *self.OptionList)
    #--Formats and positions the dropdown menu
    self.dropDown.config(width=32, font=(form.font, form.fontSizeMedium))
    self.dropDown.grid(row=3, column=2)

    self.headingText.grid(row=1, columnspan=9, pady = 120, padx = 150)

    def change_dropdown(*args): #--Function which runs when the user clicks an option
      print(self.menu.get())
      if self.menu.get() == "Determination of 'g' by a freefall method":
        #--Initiates function which contains the 'confirm' button
        command = self.confirmButton()
      elif self.menu.get() == "Click to choose Practical:":
        self.confirm_Button.grid_forget()

    self.menu.trace("w", change_dropdown) #--Allows the function to be run

  def confirmButton(self): #--Allows the user to confirm that they want to view the practical
    self.confirm_Button.grid(row=3, column=4)

  def practical(self): #--Instantiates the PracticalAnimation class which creates  new window
    command = PracticalAnimation()
```

DETERMINATION OF ‘G’ BY A FREEFALL METHOD

This practical will involve a falling ball which can be used to find out the value of the gravitational constant ‘g’. This can be done by measuring the distance between two set points and recording the time it takes for the ball to fall between them. I will split this development into sections which will link to my success criteria:

- Explaining the practical equipment needed and including pictures of them for reference.
- Displaying the animation with labels.
- Allowing the user to interact with the animation which was a key requirement set out by my stakeholders.
- Have an option for a graph to be displayed and for the animation to be repeated.

I have already created a Pygame screen so I will continue from there.

Attributes for the animation:

Like with the Tkinter window, I am going to create a new class where I can store import attributes such as co-ordinates and colours to make subsequent code easier to write. Since there will many different objects displayed on the screen, I will use multiple classes to store attributes.

This is the first class which will be used for storing attributes. I will call this ‘PracticalAttributes’.

```
225 class PracticalAttributes: #--Main class to store attributes
226   def __init__(self, WHITE, GREY, BLACK, BLUE, ORANGE, PURPLE, fontObj):
227     self.WHITE = WHITE #--Examples of colours
228     self.GREY = GREY
229     self.BLACK = BLACK
230     self.BLUE = BLUE
231     self.ORANGE = ORANGE
232     self.PURPLE = PURPLE
233     self.fontObj = fontObj #--Used to set a font
234     self.width = 50 #--Values to set the size of shapes
235     self.height = 30
236     self.x = 200 #--Co-ordinates needed to position objects in the window
237     self.y = 200
238     self.condition = True #--Allows future loops to be run and broken
239 pract = PracticalAttributes((255,255,255),(125,125,125),(0,0,0),(20,20,255), #--Class is instantiated
240                             (255,165,0),(230,230,250),pygame.font.SysFont('arial', 30))
```

Here, I have stored various colours, fonts, widths, heights and coordinates which will be important for creating labels and shapes. I have set the colours to capital letters to help differentiate them from the 'Format' class used for the Tkinter windows. This increases the readability of the code, especially when I am using two different GUIs in my program.

This is the next piece of code. Using attributes from the 'PracticalAttributes' class, I have created a Pygame screen with a new background colour, a new heading label and a button as part of the 'PracticalAnimation' class.

```
326 class PracticalAnimation:
327   def __init__(self):
328     win = pygame.display.set_mode((700, 600)) #--Sets the size of the Pygame window
329     pygame.display.set_caption('Practical Animation') #--Gives the window a name
330     win.fill(pract.ORANGE) #--Allows a background colour to be set
331     #--First heading in the window
332     heading_text = pract.fontObj.render("Determination of 'g' by a freefall method", True, pract.BLACK, pract.WHITE)
333     begin_text = pract.fontObj.render('Begin', True, pract.BLACK)
334     #--Allows the user to start the animation
335     beginButton = pygame.draw.rect(win, pract.GREY, (pract.x_beginButton, pract.y_beginButton, pract.width, pract.height))
336     #--Draws the text onscreen
337     win.blit(heading_text, (pract.x_heading, pract.y_heading))
338     win.blit(begin_text, (pract.x_beginText, pract.y_beginText))
339
340     #--While the program is running, checks whether the user is quitting the window
341     for event in pygame.event.get():
342       if event.type == pygame.QUIT:
343         pygame.quit()
344         sys.exit()
345     pygame.display.update() #--Updates display accordingly
346
347 #--Class is instantiated
348 command = PracticalAnimation()
```

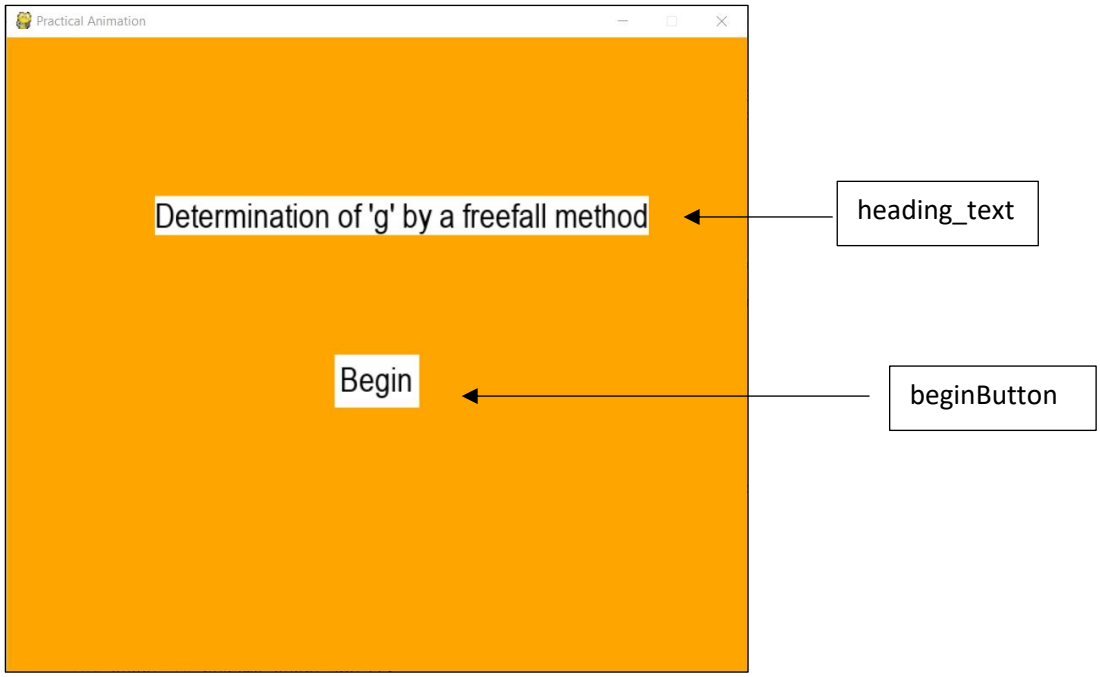
On lines 332 and 333, I have written text using 'fontObj' as part of Pygame. This is a two-step process as this text has to be displayed at specific coordinates using lines 337 and 338.

Line 345 updates the window with any new objects.

```
234     self.width = 80 #--Values to set the size of shapes
235     self.height = 50
236     self.x_beginText = 315 #--Co-ordinates needed to position objects in the window
237     self.y_beginText = 305
238     self.x_heading = 140
239     self.y_heading = 150
240     self.x_beginButton = 310
241     self.y_beginButton = 300
```

This second screenshot shows the coordinates that I have added to the 'PracticalAttributes' class so that the following screen can be displayed.

This is the resulting screen:



On line 335, I have changed the colour from white to grey to make the 'Begin' button more distinguishable:

For this button to be pressed and received as an input by Pygame I must record the position of the user's mouse. Compared to Tkinter, Pygame does not explicitly support buttons.

Below, I have started a while loop on line 293 which prints the user's mouse position in the Python shell using lines 294 and 295. For the button to be received as an input, the cursor must be within the button rectangle and the user must be pressing the button at the same time. On line 297, I have used an if statement to decide whether the cursor is within the rectangle – I have done this by adding the heights and widths of the beginButton to the cursor's current position. On line 299, another if statement is used to determine whether the button is being clicked – if so, on line 300 the while loop is broken.

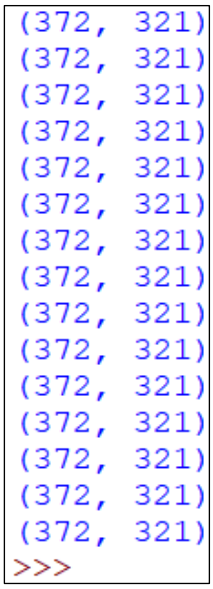
```
292   #--Starts a while loop which repeatedly checks the user's mouse position
293   while pract.condition == True:
294     pos = pygame.mouse.get_pos()
295     print(pos)
296     #--Boundaries the user's mouse position has to be in for the button to be pressed
297     if pos[0] > 310 and pos[0]< 390 and pos[1] > 300 and pos[1]< 350:
298       event = pygame.event.poll()
299       if event.type == pygame.MOUSEBUTTONDOWN:
300         pract.condition = False #--Breaks the while loop
301
302     #--While the program is running, checks whether the user is quitting the window
303     for event in pygame.event.get():
304         if event.type == pygame.QUIT:
305             pygame.quit()
306             sys.exit()
307     pygame.display.update() #--Updates display accordingly
```

Validation:

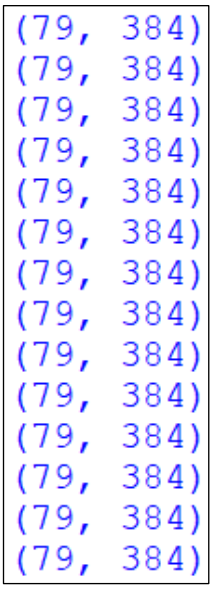
To make sure that the use enters the correct input, I have tested a variety of inputs. I have displayed a table to show these validation tests.

Results	Valid input – Mouse button pressed	Invalid input – Mouse button not pressed	Boundary input – Mouse button is clicked and held for a short duration
Expected outcome?	Yes, the while loop was broken	Yes, the while loop was not broken	Yes, the while loop was broken

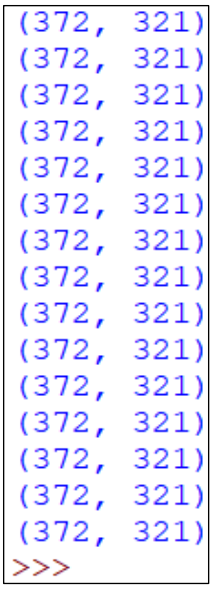
This is the result of the Python shell for each input – I have printed the positions of the mouse cursor.



Valid input – loop is ended



Invalid input – loop is not ended



Boundary input – loop is ended

Reflecting on my validation, everything worked as expected meaning the program will accept valid and boundary inputs. This is important for the robustness of my code and means later code that I will write will be subject to correct inputs making it easier for me to debug my code.

Images of practical equipment

As stated in my bullet points, I will include images of practical equipment when the Begin button is pressed. The equipment needed will include a clamp stand, an electromagnet, steel ball bearings, light gates and a stopwatch.

Clamp and Stand

Electromagnet

Light Gates

Light gates

Stopwatch

I have kept these pictures in the same folder as my python file so that they can be imported into the program.

Below I have imported the images into my program:

```
273     self.img_clamp = pygame.image.load('Clamp Stand.jpg')
274     self.img_magnet = pygame.image.load('Electromagnet.jpeg')
275     self.img_bearings = pygame.image.load('Ball bearings.jpg')
276     self.img_gates = pygame.image.load('Light Gates.jpg')
277     self.img_stopwatch = pygame.image.load('Stopwatch.jpg')
```

I will include labels which will be displayed alongside the pictures. This fulfils my success criteria.

```
251     self.clampStand_text = pract.fontObj.render("Stand and Clamp", True, pract.BLACK, pract.WHITE)
252     self.magnet_text = pract.fontObj.render("Electromagnet", True, pract.BLACK, pract.WHITE)
253     self.bearings_text = pract.fontObj.render("Steel ball bearings", True, pract.BLACK, pract.WHITE)
254     self.gates_text = pract.fontObj.render("Light Gates", True, pract.BLACK, pract.WHITE)
255     self.stopwatch_text = pract.fontObj.render("Stopwatch", True, pract.BLACK, pract.WHITE)
```

Below are the x and y coordinates of the labels and images. I used trial and improvement to calculate the correct placement. I will use the same coordinates for every picture to make the transition between each one smooth.

```
261     self.x_apparatusText = 140
262     self.y_apparatusText = 150
263     self.x_apparatusPicture = 310
264     self.y_apparatusPicture = 300
```

I have created a new class called ‘Apparatus’ for the above attributes. On line 266, it has been instantiated.

```
248 class Apparatus: #--Class which stores attributes for the apparatus part of the practical
249   def __init__(self):
```

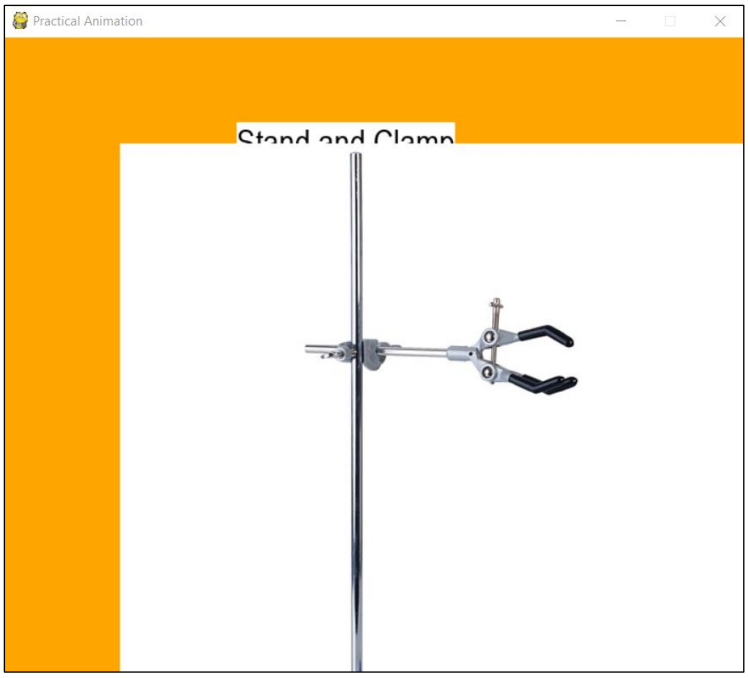
```
266 apparatus = Apparatus()
```

For the text and images to be displayed, I have used ‘win.blit’. This will come after the while loop once it has been broken.

```
      if event.type == pygame.MOUSEBUTTONDOWN:
        pract.condition = False #--Breaks the while loop
```

```
286     win.fill(pract.ORANGE)
287     win.blit(apparatus.clampStand_text, (apparatus.x_apparatusText, apparatus.y_apparatusText))
288     win.blit(apparatus.img_clamp, (apparatus.x_apparatusPicture, apparatus.y_apparatusPicture))
```

The resulting screen looked like this:



When displaying the pictures, I faced a problem. Since the pictures are different sizes, I will have to resize them to the same dimension. I have chosen a size of 200 by 200 pixels. A benefit is that I can use the same coordinates when displaying all five images.

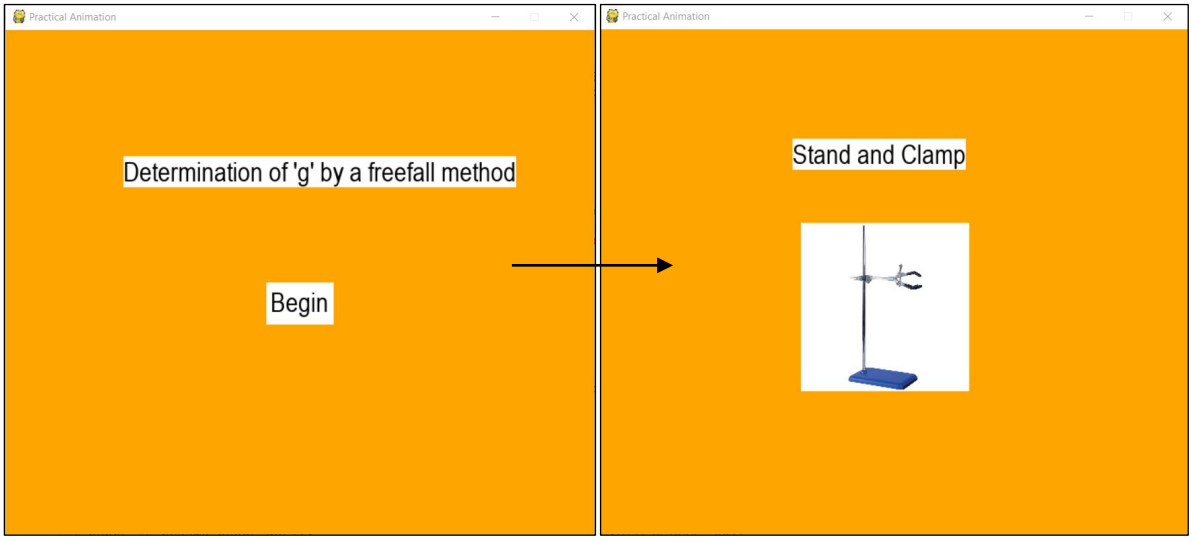
```
261     self.img_clamp = pygame.transform.scale(self.img_clamp, (200,200))
262     self.img_magnet = pygame.transform.scale(self.img_magnet, (200,200))
263     self.img_bearings = pygame.transform.scale(self.img_bearings, (200,200))
264     self.img_gates = pygame.transform.scale(self.img_gates, (200,200))
265     self.img_stopwatch = pygame.transform.scale(self.img_stopwatch, (200,200))
```

Therefore, the text and pictures now look like this:

Development Testing:

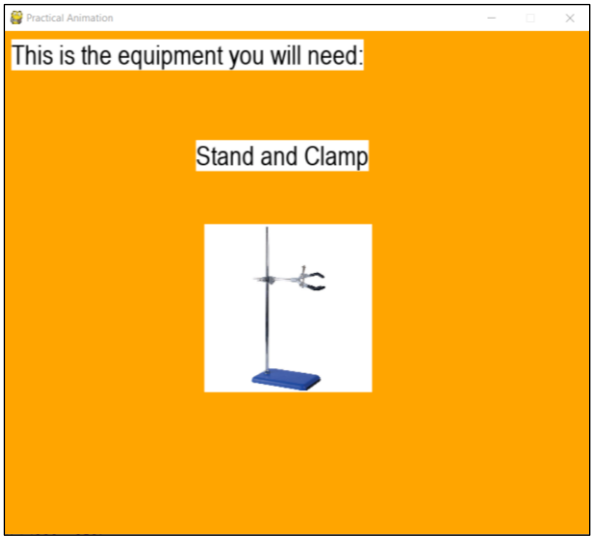
Images button	Valid input – button is pressed	Shows images of the practical equipment that will be used	Adds more realism to the animation	4.0

When the Begin button is pressed by the student, they should be able to watch the animation.



When the mouse button is clicked and the cursor's position is within the specified boundaries, the user will be transitioned to a new screen, starting the animation. This test was a success.

One modification I decided to make was displaying a label telling the user what they are viewing:



The whole Apparatus class now looks like this:

```
248 class Apparatus: #--Class which stores attributes for the apparatus part of the practical
249   def __init__(self):
250     self.time_delay = 5000 #--Time delays that will be used
251     #--List of text to be rendered and displayed onscreen using attributes from the PracticalAttributes class
252     self.equipment_text = pract.fontObj.render("This is the equipment you will need:", True, pract.BLACK, pract.WHITE)
253     self.clampStand_text = pract.fontObj.render("Stand and Clamp", True, pract.BLACK, pract.WHITE)
254     self.magnet_text = pract.fontObj.render("Electromagnet", True, pract.BLACK, pract.WHITE)
255     self.bearings_text = pract.fontObj.render("Steel ball bearings", True, pract.BLACK, pract.WHITE)
256     self.gates_text = pract.fontObj.render("Light Gates", True, pract.BLACK, pract.WHITE)
257     self.stopwatch_text = pract.fontObj.render("Stopwatch", True, pract.BLACK, pract.WHITE)
258     #--List of images to be projected, saved as jpg files
259     self.img_clamp = pygame.image.load('Clamp Stand.jpg')
260     self.img_magnet = pygame.image.load('Electromagnet.jpeg')
261     self.img_bearings = pygame.image.load('Ball bearings.jpg')
262     self.img_gates = pygame.image.load('Light Gates.jpg')
263     self.img_stopwatch = pygame.image.load('Stopwatch.jpg')
264     #Setting the overall size of each image
265     self.img_clamp = pygame.transform.scale(self.img_clamp, (200,200))
266     self.img_magnet = pygame.transform.scale(self.img_magnet, (200,200))
267     self.img_bearings = pygame.transform.scale(self.img_bearings, (200,200))
268     self.img_gates = pygame.transform.scale(self.img_gates, (200,200))
269     self.img_stopwatch = pygame.transform.scale(self.img_stopwatch, (200,200))
270     #--Coordinates of the text and images that will be displayed
271     self.x_apparatusHeading = 10
272     self.y_apparatusHeading = 10
273     self.x_apparatusText = 230
274     self.y_apparatusText = 130
275     self.x_apparatusPicture = 240
276     self.y_apparatusPicture = 230
277
278 apparatus = Apparatus() #--Class is instantiated
```

To summarise:

Lines 252 to 257 contain the text and images attributes that will be displayed to the screen. Lines 259 to 269 load the images to be displayed and resizes them to the same dimensions. Lines 271 to 276 contain the coordinates for the text and image attributes. On line 278, the class is instantiated.

Displaying images one after another

In my program, I want to display every image one after another. Therefore, I will need some sort of delay between each screen. Something like 'time.sleep' will not work in Pygame as when I tried it, it did not affect the screens. Therefore a solution will be measuring the time at two different points and if the difference between them is greater than a set time, there will be a screen transition.

On line 297, I have used pygame.time.get.ticks() which will store the current time.

```
295             pract.condition = False
296             win.fill(pract.ORANGE)
297             previous_time = pygame.time.get_ticks()
```

Below is the time delay that I have used. Since Pygame works in milliseconds, 3000 refers to 3 seconds.

```
251 class Apparatus: #--Class which stores attributes for the apparatus part of the practical
252   def __init__(self):
253     self.time_delay = 3000 #--Time delays that will be used
```

Next, I will draw the text and images onscreen shown by lines 307 to 309. On line 310, the current time will be stored.

```
304         previous_time = pygame.time.get_ticks() #--Measures current time
305         while True: #Starts another while loop
306           #--Draws the first set of text and images onscreen
307           win.blit(apparatus.equipment_text, (apparatus.x_apparatusHeading, apparatus.y_apparatusHeading))
308           win.blit(apparatus.clampStand_text, (apparatus.x_apparatusText, apparatus.y_apparatusText))
309           win.blit(apparatus.img_clamp, (apparatus.x_apparatusPicture, apparatus.y_apparatusPicture))
310           current_time = pygame.time.get_ticks() #--Measures the current time relative to the previous time
```

```
311         if current_time - previous_time > apparatus.time_delay: #--Decides when the next batch of text and images should be displayed
312           win.fill(pract.ORANGE)
313           #--Next batch of text and images are displayed when the if statement is fulfilled
314           win.blit(apparatus.equipment_text, (apparatus.x_apparatusHeading, apparatus.y_apparatusHeading))
315           win.blit(apparatus.magnet_text, (apparatus.x_apparatusText, apparatus.y_apparatusText))
316           win.blit(apparatus.img_magnet, (apparatus.x_apparatusPicture, apparatus.y_apparatusPicture))
317           current_time = pygame.time.get_ticks() #--Measures the current time relative to the previous time on line 306
```

On line 311, my program calculates the time difference between the two variables and if it is bigger than the time delay, the next batch of text and images will be displayed. I have made a mistake with my commenting on line 317 – it should say line 304 rather than 306.

Below, I have repeated the iteration for the rest of the images:

```
426             if current_time - previous_time > apparatus.time_delay:
427               win.fill(pract.ORANGE)
428               win.blit(apparatus.equipment_text, (apparatus.x_apparatusHeading, apparatus.y_apparatusHeading))
429               win.blit(apparatus.bearings_text, (apparatus.x_apparatusText, apparatus.y_apparatusText))
430               win.blit(apparatus.img_bearings, (apparatus.x_apparatusPicture, apparatus.y_apparatusPicture))
431               current_time = pygame.time.get_ticks()
432               #--Process iterates
433               if current_time - previous_time > apparatus.time_delay:
434                 win.fill(pract.ORANGE)
435                 win.blit(apparatus.equipment_text, (apparatus.x_apparatusHeading, apparatus.y_apparatusHeading))
436                 win.blit(apparatus.gates_text, (apparatus.x_apparatusText, apparatus.y_apparatusText))
437                 win.blit(apparatus.img_gates, (apparatus.x_apparatusPicture, apparatus.y_apparatusPicture))
438                 current_time = pygame.time.get_ticks()
439                 if current_time - previous_time > apparatus.time_delay:
440                   win.fill(pract.ORANGE)
441                   win.blit(apparatus.equipment_text, (apparatus.x_apparatusHeading, apparatus.y_apparatusHeading))
442                   win.blit(apparatus.stopwatch_text, (apparatus.x_apparatusText, apparatus.y_apparatusText))
443                   win.blit(apparatus.img_stopwatch, (apparatus.x_apparatusPicture, apparatus.y_apparatusPicture))
444                   current_time = pygame.time.get_ticks()
```

Now, the bearings, light gates and stopwatch images will be displayed. When running the program at this stage, I noticed that the delay between each screen decreased as I reached towards the stopwatch image. I realised that my if statement on lines 426, 433 and 439 is comparing the current time to the previous time which is on the line before the while loop starts. Therefore, with every iteration, the current time increases and so does the time difference between the previous_time and current_time, rather than staying constant.

To solve this, I have introduced multiple delays, each of which is larger than the previous one:

```
251 class Apparatus: #--Class which stores attributes for the apparatus part of the practical
252   def __init__(self):
253     self.time_delay = 3000 #--Time delays that will be used
254     self.time_delay2 = 6000
255     self.time_delay3 = 9000
256     self.time_delay4 = 12000
```

This is the code with the new time delays:

```
425             #--Larger time gap means a larger time delay is used
426             if current_time - previous_time > apparatus.time_delay2:
427               win.fill(pract.ORANGE)
428               win.blit(apparatus.equipment_text, (apparatus.x_apparatusHeading, apparatus.y_apparatusHeading))
429               win.blit(apparatus.bearings_text, (apparatus.x_apparatusText, apparatus.y_apparatusText))
430               win.blit(apparatus.img_bearings, (apparatus.x_apparatusPicture, apparatus.y_apparatusPicture))
431               current_time = pygame.time.get_ticks()
432               #--Process iterates
433               if current_time - previous_time > apparatus.time_delay3:
434                 win.fill(pract.ORANGE)
435                 win.blit(apparatus.equipment_text, (apparatus.x_apparatusHeading, apparatus.y_apparatusHeading))
436                 win.blit(apparatus.gates_text, (apparatus.x_apparatusText, apparatus.y_apparatusText))
437                 win.blit(apparatus.img_gates, (apparatus.x_apparatusPicture, apparatus.y_apparatusPicture))
438                 current_time = pygame.time.get_ticks()
439                 if current_time - previous_time > apparatus.time_delay4:
440                   win.fill(pract.ORANGE)
441                   win.blit(apparatus.equipment_text, (apparatus.x_apparatusHeading, apparatus.y_apparatusHeading))
442                   win.blit(apparatus.stopwatch_text, (apparatus.x_apparatusText, apparatus.y_apparatusText))
443                   win.blit(apparatus.img_stopwatch, (apparatus.x_apparatusPicture, apparatus.y_apparatusPicture))
444                   current_time = pygame.time.get_ticks()
```

These are all the screens for every iteration:

Practical Animation

This is the equipment you will need:

Stand and Clamp

Practical Animation

This is the equipment you will need:

Electromagnet

Practical Animation

This is the equipment you will need:

Steel ball bearings

Practical Animation

This is the equipment you will need:

Light Gates

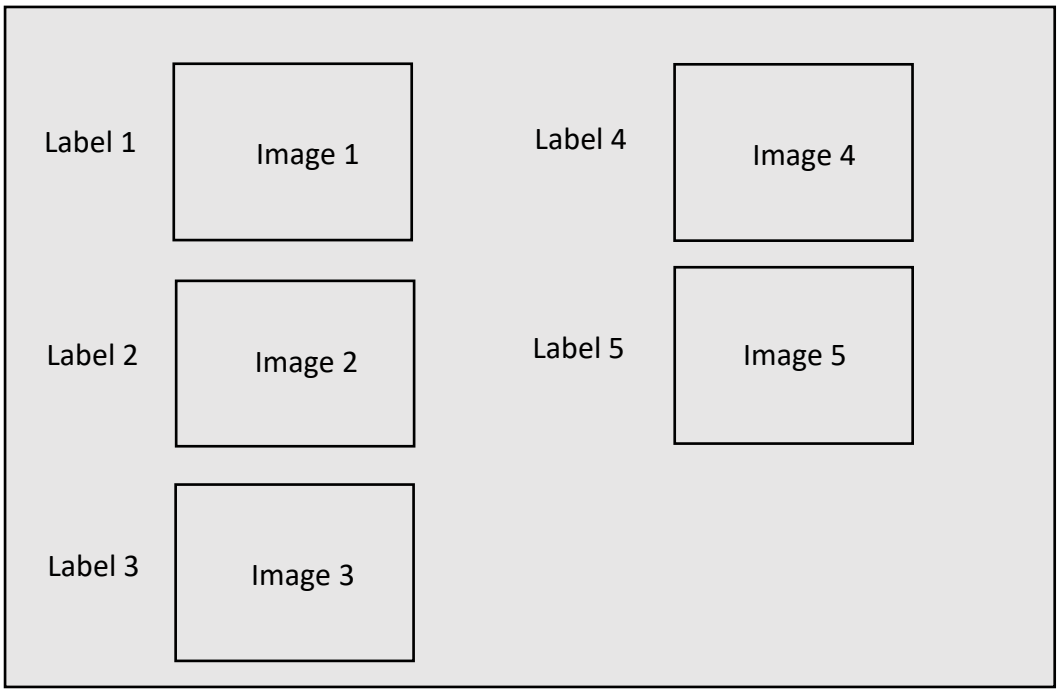
Practical Animation

This is the equipment you will need:

Stopwatch

When reflecting on these screens, I decided to make them neater and more modern. Since my program is targeted at students, I think a modern interface is important to implement. This relates to my secondary research on Memrise as I found its user interface enjoyable to use. Therefore, I will shrink the size of each image even more and try to fit it on the same screen. I will still keep the iteration where every image comes one after another.

I plan to have the images in two columns and three rows like below:



I have written some new coordinates which are labelled below:

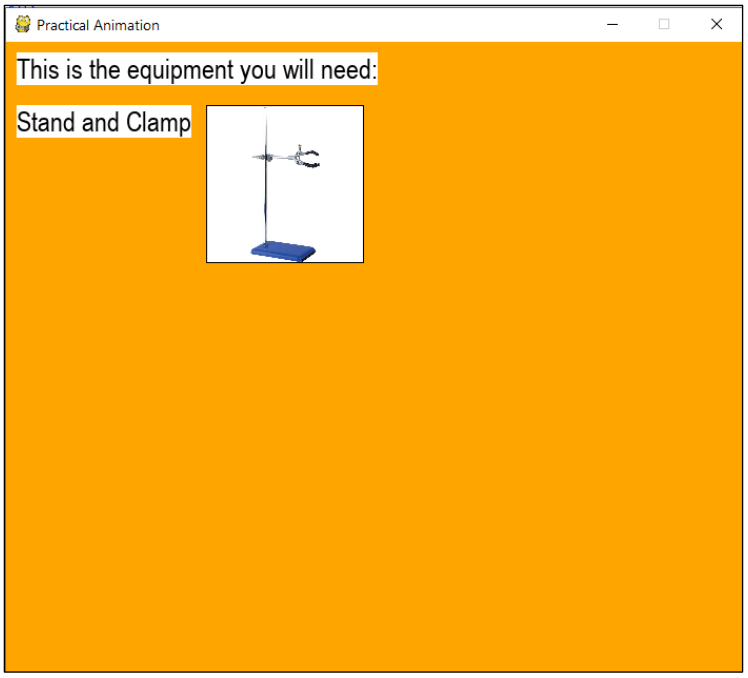
```
286     self.x_apparatusText = 10
287     self.x_apparatusPicture = 190
288     self.x2_apparatusText = 380
289     self.x2_apparatusPicture = 510
290
291     self.y_apparatusRow1 = 60
292     self.y_apparatusRow2 = 230
293     self.y_apparatusRow3 = 400
```

On line 288 and 289, the character ‘x2’ refers to the second column of text and images that will be displayed onscreen – labels and images 4 & 5. From lines 291 to 293, I have written the y coordinates for every row.

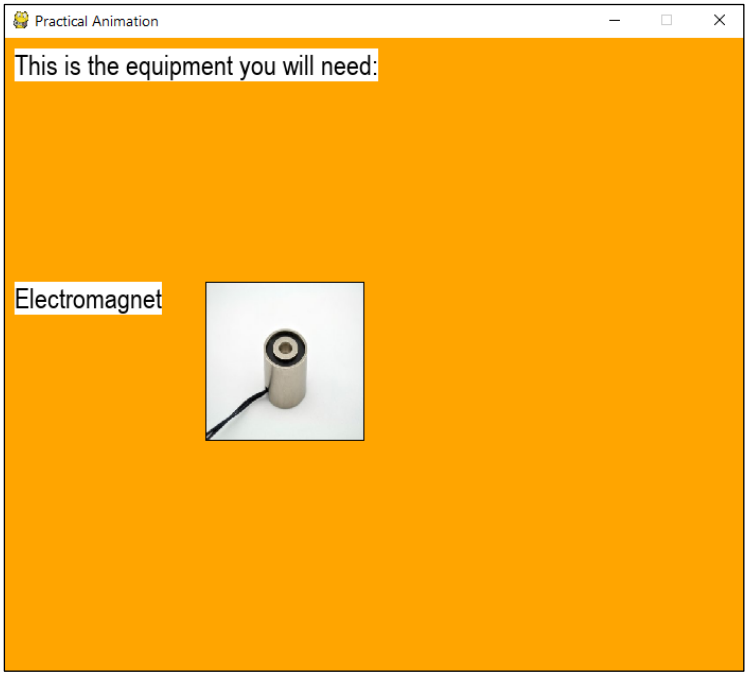
After, I just substituted the new coordinates into the old coordinates. This is shown by lines 413 and 414. On line 415, I have created a black rectangle which will form the border of this image. This is to make the images look nicer.

```
409         previous_time = pygame.time.get_ticks() #--Measures current time
410         while pract.condition2 == True: #Starts another while loop
411           #--Draws the first set of text and images onscreen
412           win.blit(apparatus.equipment_text, (apparatus.x_apparatusHeading, apparatus.y_apparatusHeading))
413           win.blit(apparatus.clampStand_text, (apparatus.x_apparatusText, apparatus.y_apparatusRow1))
414           win.blit(apparatus.img_clamp, (apparatus.x_apparatusPicture, apparatus.y_apparatusRow1))
415           clampStand_rect = pygame.draw.rect(win, pract.BLACK, (apparatus.x_apparatusPicture, apparatus.y_apparatusRow1, 150,150),1)
416           current_time = pygame.time.get_ticks() #--Measures the current time relative to the previous time
```

This is what the first screen looks like:



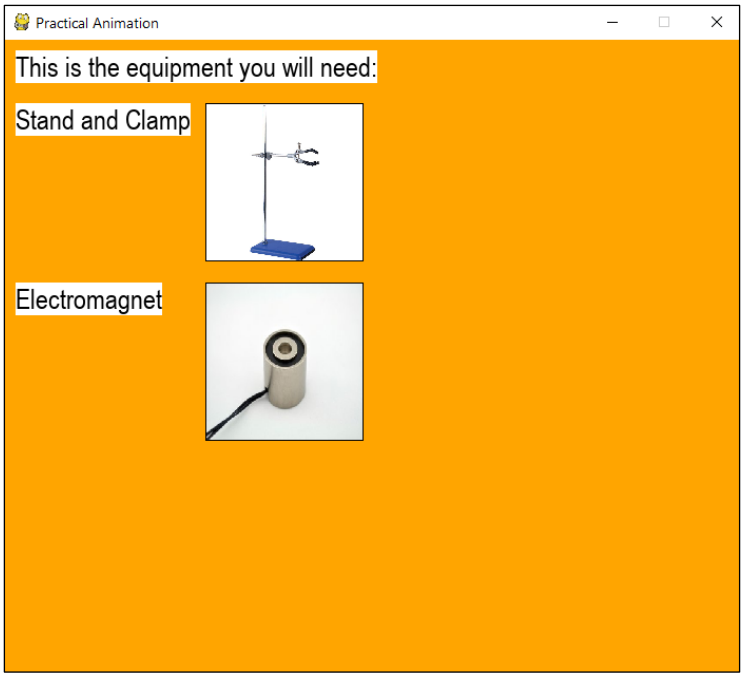
I continued this for the rest of the images and text. However, when running the program for the second image, the first image disappeared.



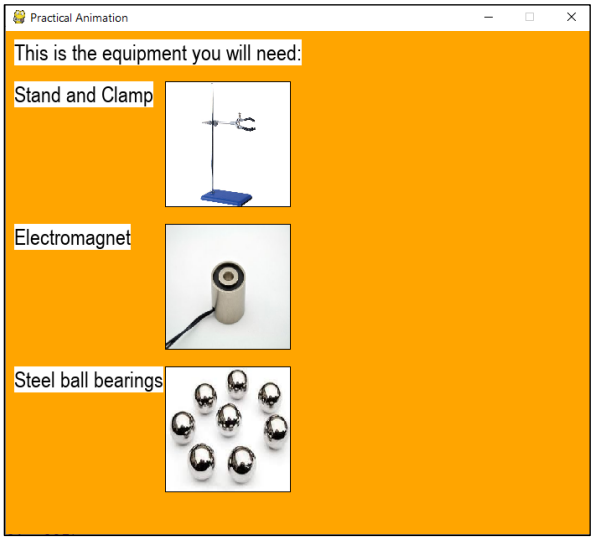
I realised that before, I had re-painted the background to simulate a screen change but this isn't needed since all the images are now on the same screen.

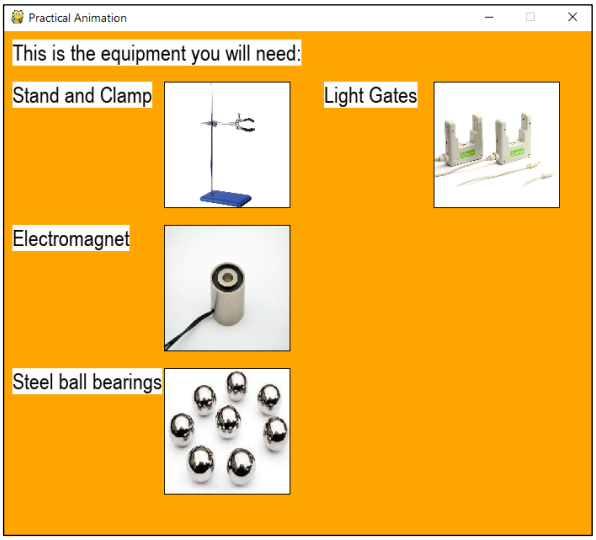
```
419                 win.fill(pract.ORANGE)
```

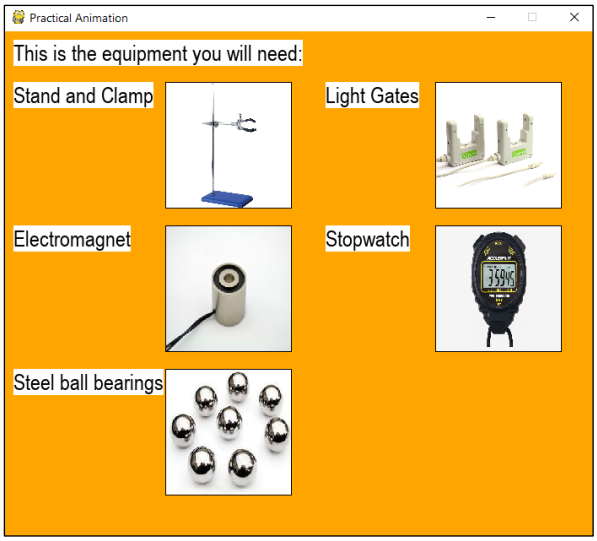
After removing this line of code, the screen now looked like this:



Repeating this for the rest of the images and texts:







To improve the usability of my program, I have reduced the time delay by one second. I feel two seconds is a good compromise between the user having enough time to view the screen and wanting to move on.

```
257 class Apparatus: #--Class which stores attributes for the apparatus part of the practical
258   def __init__(self):
259     self.time_delay = 2000 #--Time delays that will be used
260     self.time_delay2 = 4000
261     self.time_delay3 = 6000
262     self.time_delay4 = 8000
```

After the iteration finishes, the student will have to be transitioned to the next screen. I will include another time delay on line 263 after the fifth image has been displayed. Once the final image has been displayed, the user will wait 3 seconds before the screen changes. As part of usability, I have set the final time delay slightly longer so the user has a final chance to look at all of the images and labels.

```
257 class Apparatus: #--Class which stores attributes for the apparatus part of the practical
258   def __init__(self):
259     self.time_delay = 2000 #--Time delays that will be used
260     self.time_delay2 = 4000
261     self.time_delay3 = 6000
262     self.time_delay4 = 8000
263     self.time_delay5 = 11000
```

Below, on lines 441 and 442, if the delay equals three seconds, the AnimationInteraction method will be run. This method will run the next part of the animation which I will detail later. Since there are two while loops in this class – one for checking the cursor's position and one for the iteration of images - there are two 'for' statements which will update the screens.

```
440                     #--Waits some time before transitioning the screen
441                     if current_time - previous_time > apparatus.time_delay5:
442                       command = self.AnimationInteraction() #--Moves onto the next method
443
444             #--While the program is running, checks whether the user is quitting the window
445             for event in pygame.event.get():
446               if event.type == pygame.QUIT:
447                 pygame.quit()
448                 sys.exit()
449             pygame.display.update()
450
451       #--Since there are two while loops, there needs to be two lots of this code
452       for event in pygame.event.get():
453           if event.type == pygame.QUIT:
454             pygame.quit()
455             sys.exit()
456       pygame.display.update()
```

This is what the entire iteration looks like:

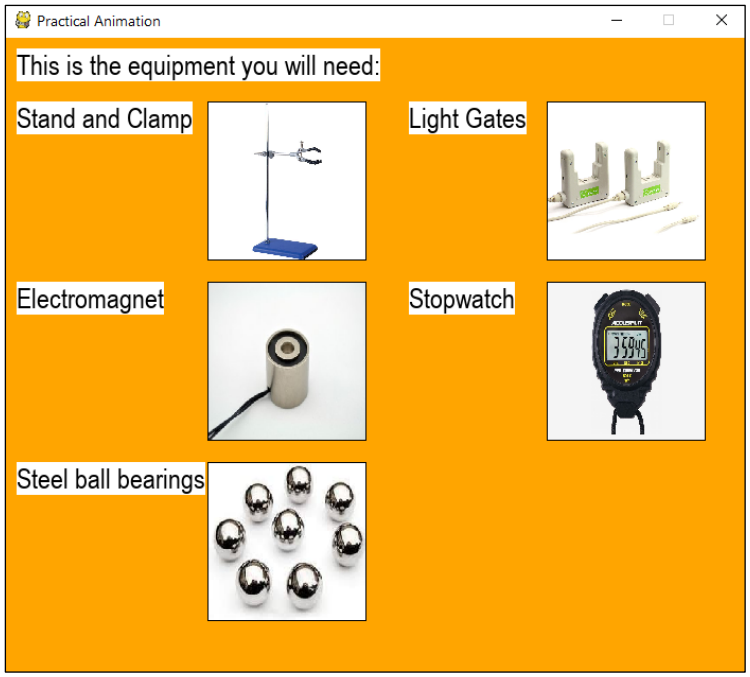
```
414     #--Starts a while loop which repeatedly checks the user's mouse position
415     while pract.condition == True:
416       pos = pygame.mouse.get_pos()
417       print(pos)
418       #--Boundaries the user's mouse position has to be in for the button to be pressed
419       if pos[0] > 310 and pos[0]< 390 and pos[1] > 300 and pos[1]< 350:
420         event = pygame.event.poll()
421         if event.type == pygame.MOUSEBUTTONDOWN:
422           pract.condition = False #--Breaks the while loop
423           win.fill(pract.ORANGE) #--New backgroud which replaces everything onscreen
424           previous_time = pygame.time.get_ticks() #--Measures current time
425           while pract.condition2 == True: #Starts another while loop
426             #--Draws the first set of text and images onscreen
427             win.blit(apparatus.equipment_text, (apparatus.x_apparatusHeading, apparatus.y_apparatusHeading))
428             win.blit(apparatus.clampStand_text, (apparatus.x_apparatusText, apparatus.y_apparatusRow1))
429             win.blit(apparatus.img_clamp, (apparatus.x_apparatusPicture, apparatus.y_apparatusRow1))
430             clampStand_rect = pygame.draw.rect(win, pract.BLACK, (apparatus.x_apparatusPicture, apparatus.y_apparatusRow1, 150,150),1)
431             current_time = pygame.time.get_ticks() #--Measures the current time relative to the previous time
432             if current_time - previous_time > apparatus.time_delay: #--Decides when the next batch of text and images should be displayed
433               #--Next batch of text and images are displayed when the if statement is fulfilled
434               win.blit(apparatus.equipment_text, (apparatus.x_apparatusHeading, apparatus.y_apparatusHeading))
435               win.blit(apparatus.magnet_text, (apparatus.x_apparatusText, apparatus.y_apparatusRow2))
436               win.blit(apparatus.img_magnet, (apparatus.x_apparatusPicture, apparatus.y_apparatusRow2))
437               magnet_rect = pygame.draw.rect(win, pract.BLACK, (apparatus.x_apparatusPicture, apparatus.y_apparatusRow2, 150,150),1)
438               current_time = pygame.time.get_ticks() #--Measures the current time relative to the previous time on line 306
439               #--Larger time gap means a larger time delay is used
440               if current_time - previous_time > apparatus.time_delay2:
441                 win.blit(apparatus.equipment_text, (apparatus.x_apparatusHeading, apparatus.y_apparatusHeading))
442                 win.blit(apparatus.bearings_text, (apparatus.x_apparatusText, apparatus.y_apparatusRow3))
443                 win.blit(apparatus.img_bearings, (apparatus.x_apparatusPicture, apparatus.y_apparatusRow3))
444                 bearings_rect = pygame.draw.rect(win, pract.BLACK, (apparatus.x_apparatusPicture, apparatus.y_apparatusRow3, 150,150),1)
445                 current_time = pygame.time.get_ticks()
446                 #--Process iterates
447                 if current_time - previous_time > apparatus.time_delay3:
448                   win.blit(apparatus.equipment_text, (apparatus.x_apparatusHeading, apparatus.y_apparatusHeading))
449                   win.blit(apparatus.gates_text, (apparatus.x2_apparatusText, apparatus.y_apparatusRow1))
450                   win.blit(apparatus.img_gates, (apparatus.x2_apparatusPicture, apparatus.y_apparatusRow1))
451                   gates_rect = pygame.draw.rect(win, pract.BLACK, (apparatus.x2_apparatusPicture, apparatus.y_apparatusRow1, 150,150),1)
452                   current_time = pygame.time.get_ticks()
453                   if current_time - previous_time > apparatus.time_delay4:
454                     win.blit(apparatus.equipment_text, (apparatus.x_apparatusHeading, apparatus.y_apparatusHeading))
455                     win.blit(apparatus.stopwatch_text, (apparatus.x2_apparatusText, apparatus.y_apparatusRow2))
456                     win.blit(apparatus.img_stopwatch, (apparatus.x2_apparatusPicture, apparatus.y_apparatusRow2))
457                     stopwatch_rect = pygame.draw.rect(win, pract.BLACK, (apparatus.x2_apparatusPicture, apparatus.y_apparatusRow2, 150,150),1)
458                     current_time = pygame.time.get_ticks()
459                     #--Waits some time before transitioning the screen
460                     if current_time - previous_time > apparatus.time_delay5:
461                       command = self.AnimationInteraction() #--Moves onto the next method
```

To summarise, on line 415, I use a while loop to detect the user's cursor position. If this is in the required boundary and the mouse button is clicked, the next while loop on line 425 is started and the images and text are then displayed onscreen. If the time delay condition is met, the next lot of images and text are displayed. Once all the images have been displayed, the AnimationInteraction method is instantiated on line 461.

Development Testing

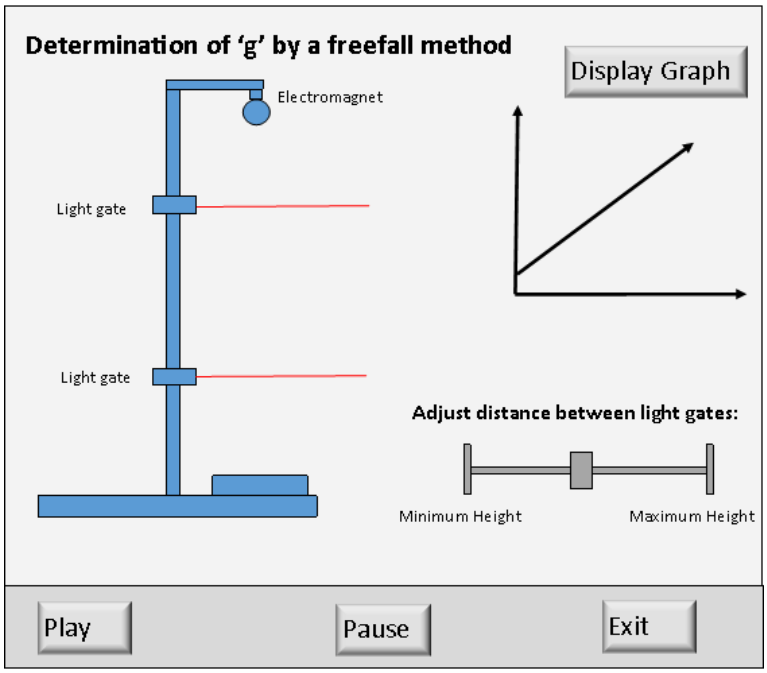
Images button	Valid input – button is pressed	Shows images of the practical equipment that will be used	Adds more realism to the animation	4.0

I have repeated test 4.0 to show that all the images have been displayed.



All images were displayed meaning that this test is a success.

INTERACTING WITH THE ANIMATION



As previously said, this practical will involve a falling ball which can be used to find out the value of the gravitational constant 'g'. The student should have control over the height between the light gates shown by the adjustable slider that I hope to implement. Changing the height will affect the time it takes for the ball to fall - this can be recorded and displayed as a graph allowing the student to find the value of 'g'.

I will tackle this animation in three stages:

1. Display the diagram onscreen
2. Accurately showing the ball's motion when falling
3. Having a stopwatch that records the time it takes for the ball to fall
4. Allowing the student to adjust the height of the light gates and showing how this affects the time recorded
5. Allowing the student to view a graph of the recorded results

This list will be the order of priority for this animation due to time constraints.

New class to store new attributes:

I have introduced a new class called MovingAnimation which will store attributes for objects needed for the animation.

```
284 class MovingAnimation: #--Class for all the attributes to create the moving animation
285   def __init__(self):
286     self.x_stand = 50 #--These attributes are for the x and y coordinate of every object in the screen
287     self.y_stand = 50
288     self.x_base = 30
289     self.y_base = 530
290     self.x_clamp = 50
291     self.y_clamp = 50
292 
293     self.x_electromagnet = 155
294     self.y_electromagnet = 62
295     self.x_ball = 170
296     self.y_ball = 90
297 
298     #--Since many of my objects are rectangles, here, I have included a height and width.
299     self.width_stand = 12
300     self.height_stand = 480
301     self.width_base = 220
302     self.height_base = 12
303     self.width_clamp = 140
304     self.height_clamp = 12
305 
306     self.width_electromagnet = 35
307     self.height_electromagnet = 20
308     self.radius_ball = 10
```

For other classes to use these attributes, I have instantiated the class.

```
377 #--Class is instantiated
378 mov = MovingAnimation()
```

To draw objects onscreen, I have created a new method called 'AnimationInteraction' where I have redefined the window size and repainted the screen so the images and text from the previous screen are removed.

```
457   def AnimationInteraction(self):
458     win = pygame.display.set_mode((700, 600))
459     win.fill(pract.ORANGE)
```

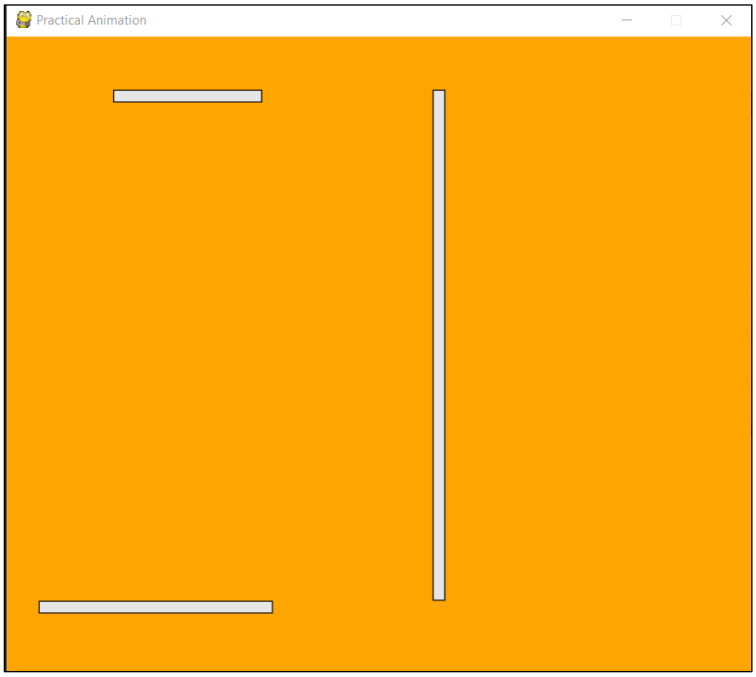
Creating the apparatus onscreen:

To create the stand and clamp, I have created three rectangles. This reflects my screen design.

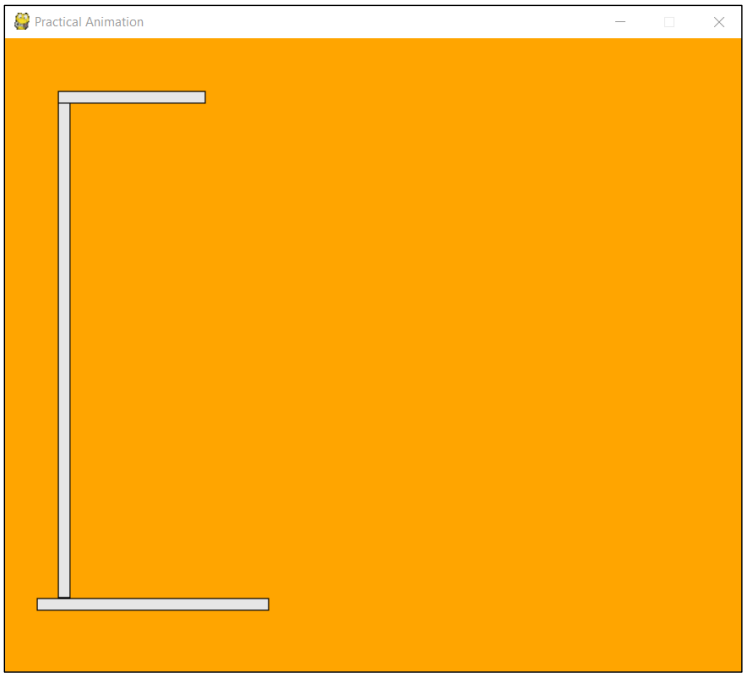
```
409                     stand = pygame.draw.rect(win, pract.GREY,(mov.x_stand, mov.y_stand, mov.width_stand, mov.height_stand))
410                     #--An identical rectangle is created for the border
411                     stand = pygame.draw.rect(win, pract.BLACK,(mov.x_stand, mov.y_stand, mov.width_stand, mov.height_stand), 1)
412
413                     #--Rectangles for the base of the stand
414                     base = pygame.draw.rect(win, pract.GREY,(mov.x_base, mov.y_base, mov.width_base, mov.height_base))
415                     base = pygame.draw.rect(win, pract.BLACK,(mov.x_base, mov.y_base, mov.width_base, mov.height_base), 1)
416
417                     #--Rectangles for the top of the stand
418                     clamp = pygame.draw.rect(win, pract.GREY,(mov.x_clamp, mov.y_clamp, mov.width_clamp, mov.height_clamp))
419                     clamp = pygame.draw.rect(win, pract.BLACK,(mov.x_clamp, mov.y_clamp, mov.width_clamp, mov.height_clamp), 1)
```

I have two rectangles for every object so I can make a border. I will be using a thickness of 1. This makes the rectangles look better and improves the user interface which is important for a program targeted at students.

This is what the rectangles look like:



After changing the coordinates through trial and improvement:



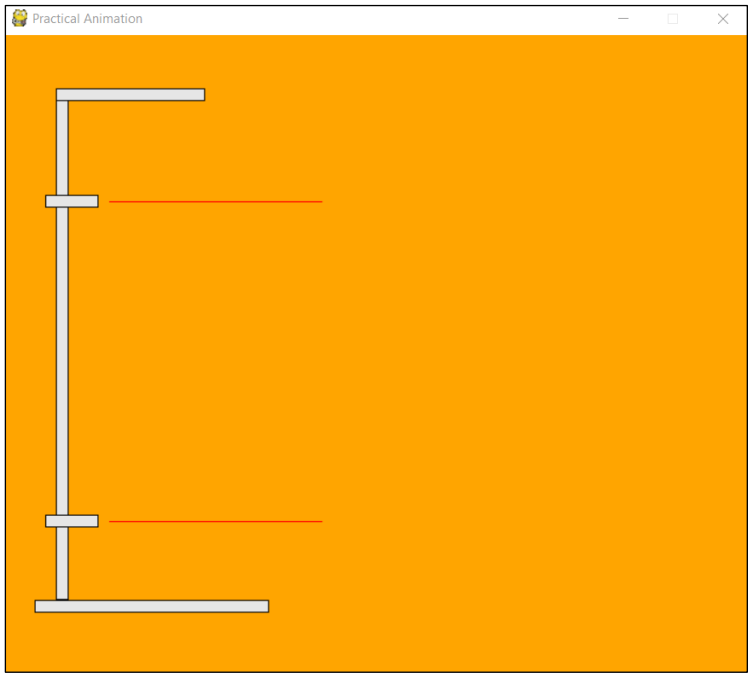
Next, I will add the light gates and light rays. I have taken the same approach of adding a border to each rectangle.

```
422                 #--Drawing lightgates as simple rectangles
423                 lightgate = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate))
424                 lightgate = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate), 1)
425
426                 lightgate2 = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate2, mov.y_lightgate2, mov.width_lightgate2, mov.height_lightgate2))
427                 lightgate2 = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate2, mov.y_lightgate2, mov.width_lightgate2, mov.height_lightgate2), 1)
428
429                 #--Treating these as rectangles but with very small heights
430                 lightray = pygame.draw.rect(win, pract.RED,(mov.x_lightray, mov.y_lightray, mov.width_lightray, mov.height_lightray))
431                 lightray2 = pygame.draw.rect(win, pract.RED,(mov.x_lightray2, mov.y_lightray2, mov.width_lightray2, mov.height_lightray2))
```

Using these coordinates and dimensions:

```
311     self.x_lightgate = 40
312     self.y_lightgate = 150
313     self.x_lightgate2 = 40
314     self.y_lightgate2 = 450
315
316     self.x_lightray = 100
317     self.y_lightray = 156
318     self.x_lightray2 = 100
319     self.y_lightray2 = 456
```

```
338     self.width_lightgate = 50
339     self.height_lightgate = 12
340     self.width_lightgate2 = 50
341     self.height_lightgate2 = 12
342
343     self.width_lightray = 200
344     self.height_lightray = 1
345     self.width_lightray2  = 200
346     self.height_lightray2 = 1
```



While the colour red for the light rays is not realistic, they will improve usability as they will be clearer for a student. Since my program is an abstraction of real-life physics practicals, I can ignore things such as colour to improve the user experience.

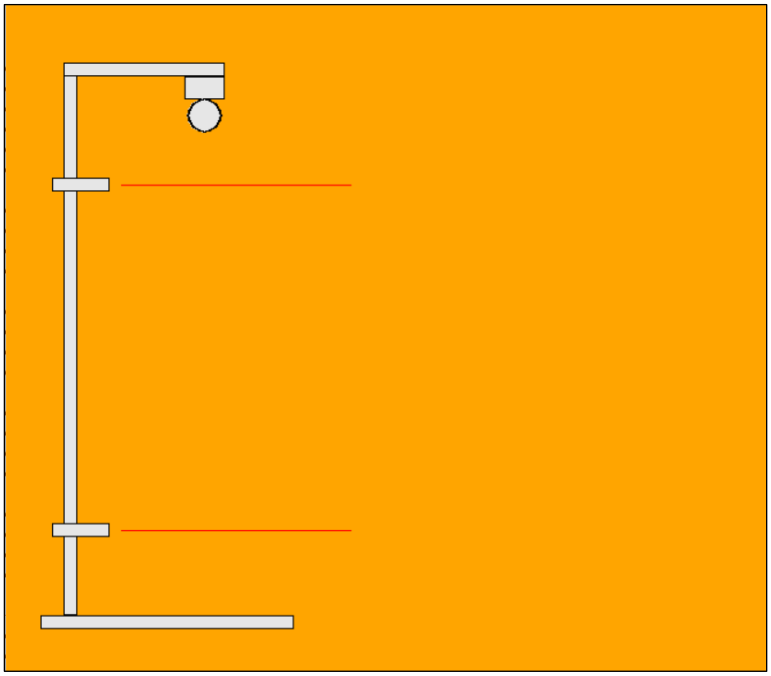
Next, I will add the electromagnet and ball bearing:

```
433                 #--Drawing electromagnets as simple rectangles
434                 electromagnet = pygame.draw.rect(win, pract.GREY,(mov.x_electromagnet, mov.y_electromagnet, mov.width_electromagnet,
435                                                                   mov.height_electromagnet))
436                 electromagnet = pygame.draw.rect(win, pract.BLACK,(mov.x_electromagnet, mov.y_electromagnet, mov.width_electromagnet,
437                                                                    mov.height_electromagnet), 1)
438
439                 #--Will be drawn as a circle
440                 ball_bearing = pygame.draw.circle(win, pract.GREY,(mov.x_ball, mov.y_ball), mov.radius_ball)
441                 ball_bearing = pygame.draw.circle(win, pract.BLACK,(mov.x_ball, mov.y_ball), mov.radius_ball,1)
```

With the following coordinates and dimensions:

```
306     self.x_electromagnet = 155
307     self.y_electromagnet = 62
308     self.x_ball = 172
309     self.y_ball = 96
```

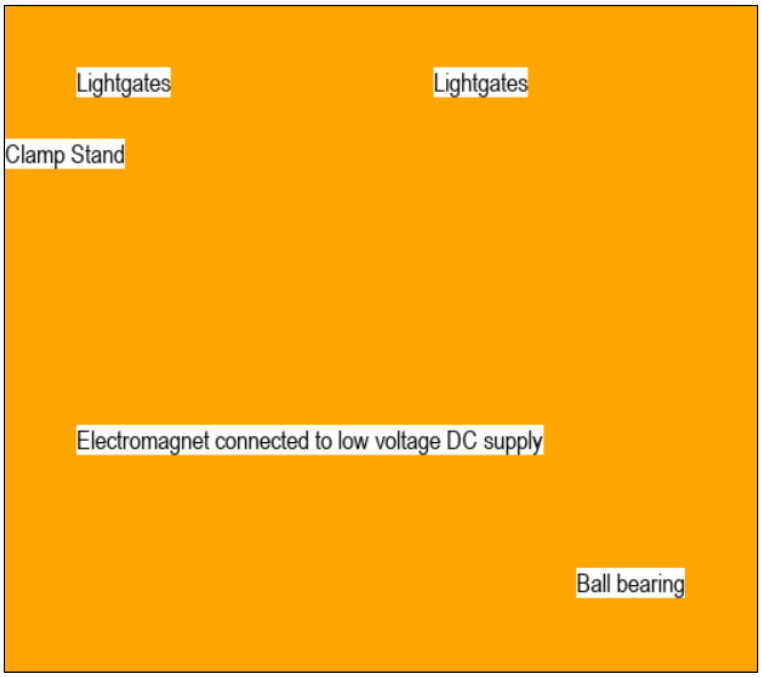
```
334     self.width_electromagnet = 35
335     self.height_electromagnet = 20
336     self.radius_ball = 15
```



Next, I will need to include text labels for each apparatus as part of my success criteria.

```
343     #--Labels for objects in the screen
344     self.clampStand_label = pract.fontObj_small.render("Clamp Stand", True, pract.BLACK, pract.WHITE)
345     self.lightgates_label = pract.fontObj_small.render("Lightgates", True, pract.BLACK, pract.WHITE)
346     self.magnet_label = pract.fontObj_small.render("Electromagnet connected to low voltage DC supply", True, pract.BLACK, pract.WHITE)
347     self.bearings_label = pract.fontObj_small.render("Ball bearing", True, pract.BLACK, pract.WHITE)
```

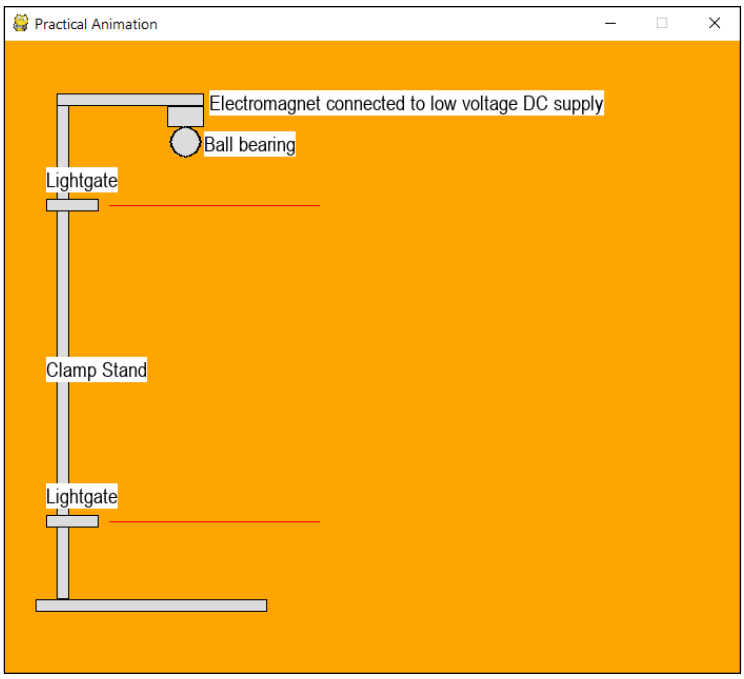
```
463                     #--Drawing to the screen the various labels at the specified x and y coordinates
464                     win.blit(mov.clampStand_label, (mov.x_clampStandlabel, mov.y_clampStandlabel))
465                     win.blit(mov.lightgates_label, (mov.x_lightgateslabel, mov.y_lightgateslabel))
466                     win.blit(mov.lightgates_label, (mov.x_lightgateslabel2, mov.y_lightgateslabel2))
467                     win.blit(mov.magnet_label, (mov.x_magnetlabel, mov.y_magnetlabel))
468                     win.blit(mov.bearings_label, (mov.x_bearingslabel, mov.y_bearingslabel))
```



I will arrange the labels in the correct positions using the below coordinates:

```
359     #--X and Y coordinate for every label
360     self.x_clampStandlabel = 40
361     self.y_clampStandlabel = 300
362     self.x_lightgateslabel = 40
363     self.y_lightgateslabel = 120
364     self.x_lightgateslabel2 = 40
365     self.y_lightgateslabel2 = 420
366     self.x_magnetlabel = 195
367     self.y_magnetlabel = 47
368     self.x_bearingslabel = 190
369     self.y_bearingslabel = 86
```

This produces the below screen:



Now my diagram is complete, I can move on to simulating the ball falling.

Creating the falling ball motion.

The next step is to make the ball bearing fall due to acceleration due to gravity. To create a falling ball motion, I will need to change the y coordinate of the ball bearings repeatedly using a loop, giving the illusion that the ball is accelerating.

To begin with, I will use a simple while loop to change the y coordinate of the ball. When the space key is pressed the ball will move, otherwise, the ball will be displayed as normal. This will be part of the method 'AnimationInteraction'.

Illustrated below is how I have changed the y coordinate of the ball before redisplaying it to the screen.

```
496         mov.y_ball += mov.acceleration #--Moves the ball downwards
497         ball_bearing = pygame.draw.circle(win, pract.GREY,(mov.x_ball, mov.y_ball), mov.radius_ball)
498         ball_bearing = pygame.draw.circle(win, pract.BLACK,(mov.x_ball, mov.y_ball), mov.radius_ball,1)
```

I have introduced a new attribute on line 496 which was instantiated in the 'MovingAnimation' class.

```
373     self.acceleration = 3
```

Below shows how I have integrated this into the AnimationInteraction class.

```
457   def AnimationInteraction(self):
458     win = pygame.display.set_mode((700, 600))
459     win.fill(pract.ORANGE)
460     while True:
461       keys = pygame.key.get_pressed() #--Detects user's key inputs
462       if keys[pygame.K_SPACE]: #--When the spacebar is pressed
463         #--Practical diagram is redrawn
464         stand = pygame.draw.rect(win, pract.GREY,(mov.x_stand, mov.y_stand, mov.width_stand, mov.height_stand))
465         #--An identical rectangle is created for the border
466         stand = pygame.draw.rect(win, pract.BLACK,(mov.x_stand, mov.y_stand, mov.width_stand, mov.height_stand), 1)
467         #--Rectangles for the base of the stand
468         base = pygame.draw.rect(win, pract.GREY,(mov.x_base, mov.y_base, mov.width_base, mov.height_base))
469         base = pygame.draw.rect(win, pract.BLACK,(mov.x_base, mov.y_base, mov.width_base, mov.height_base), 1)
470         #--Rectangles for the top of the stand
471         clamp = pygame.draw.rect(win, pract.GREY,(mov.x_clamp, mov.y_clamp, mov.width_clamp, mov.height_clamp))
472         clamp = pygame.draw.rect(win, pract.BLACK,(mov.x_clamp, mov.y_clamp, mov.width_clamp, mov.height_clamp), 1)
473         #--Drawing lightgates as rectangles
474         lightgate = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate))
475         lightgate = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate), 1)
476         lightgate2 = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate2, mov.y_lightgate2, mov.width_lightgate2, mov.height_lightgate2))
477         lightgate2 = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate2, mov.y_lightgate2, mov.width_lightgate2, mov.height_lightgate2), 1)
478         #--Treating these as rectangles but with very small heights
479         lightray = pygame.draw.rect(win, pract.RED,(mov.x_lightray, mov.y_lightray, mov.width_lightray, mov.height_lightray))
480         lightray2 = pygame.draw.rect(win, pract.RED,(mov.x_lightray2, mov.y_lightray2, mov.width_lightray2, mov.height_lightray2))
481         #--Drawing electromagnets as rectangles
482         electromagnet = pygame.draw.rect(win, pract.GREY,(mov.x_electromagnet, mov.y_electromagnet, mov.width_electromagnet,mov.height_electromagnet))
483         electromagnet = pygame.draw.rect(win, pract.BLACK,(mov.x_electromagnet, mov.y_electromagnet, mov.width_electromagnet,mov.height_electromagnet), 1)
484         #--Will be drawn as a circle
485         ball_bearing = pygame.draw.circle(win, pract.GREY,(mov.x_ball, mov.y_ball), mov.radius_ball)
486         ball_bearing = pygame.draw.circle(win, pract.BLACK,(mov.x_ball, mov.y_ball), mov.radius_ball,1)
487
488         #--Drawing to the screen the various labels at the specified x and y coordinates
489         win.blit(mov.clampStand_label, (mov.x_clampStandlabel, mov.y_clampStandlabel))
490         win.blit(mov.lightgates_label, (mov.x_lightgateslabel, mov.y_lightgateslabel))
491         win.blit(mov.lightgates_label, (mov.x_lightgateslabel2, mov.y_lightgateslabel2))
492         win.blit(mov.magnet_label, (mov.x_magnetlabel, mov.y_magnetlabel))
493         win.blit(mov.bearings_label, (mov.x_bearingslabel, mov.y_bearingslabel))
494
495         mov.y_ball += mov.acceleration #--Moves the ball downwards
496         ball_bearing = pygame.draw.circle(win, pract.GREY,(mov.x_ball, mov.y_ball), mov.radius_ball)
497         ball_bearing = pygame.draw.circle(win, pract.BLACK,(mov.x_ball, mov.y_ball), mov.radius_ball,1)
```

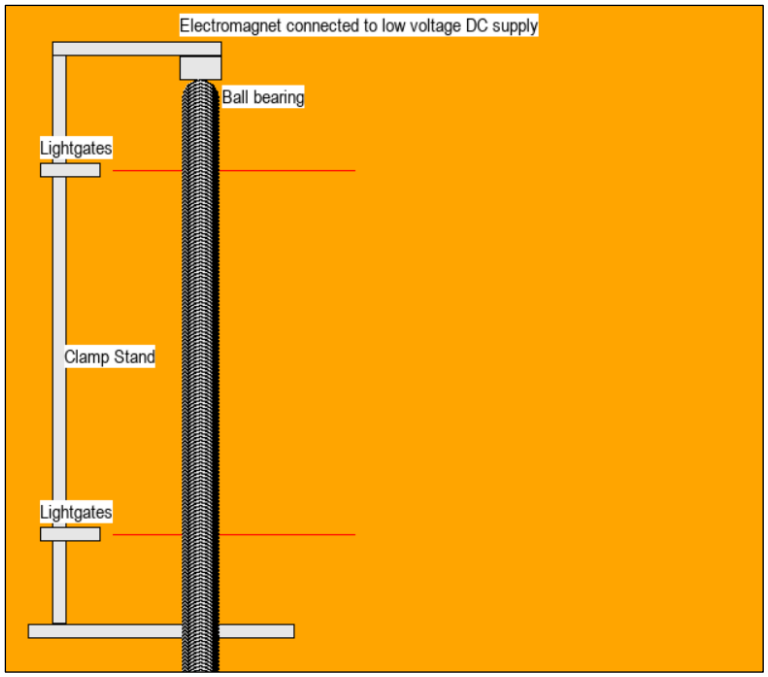
On line 462, the program detects whether the space bar is pressed. If so, the diagram is displayed shown by lines 464 to 493. The while loop on line 460 repeats this process but with a new y-coordinate for the ball shown by lines 495 to 497. If nothing is pressed, the program will just display the diagram with no ball movement shown by the below code.

```
498       else:
499         stand = pygame.draw.rect(win, pract.GREY,(mov.x_stand, mov.y_stand, mov.width_stand, mov.height_stand))
500         #--An identical rectangle is created for the border
501         stand = pygame.draw.rect(win, pract.BLACK,(mov.x_stand, mov.y_stand, mov.width_stand, mov.height_stand), 1)
502         #--Rectangles for the base of the stand
503         base = pygame.draw.rect(win, pract.GREY,(mov.x_base, mov.y_base, mov.width_base, mov.height_base))
504         base = pygame.draw.rect(win, pract.BLACK,(mov.x_base, mov.y_base, mov.width_base, mov.height_base), 1)
505         #--Rectangles for the top of the screen
506         clamp = pygame.draw.rect(win, pract.GREY,(mov.x_clamp, mov.y_clamp, mov.width_clamp, mov.height_clamp))
507         clamp = pygame.draw.rect(win, pract.BLACK,(mov.x_clamp, mov.y_clamp, mov.width_clamp, mov.height_clamp), 1)
508         #--Drawing lightgates as rectangles
509         lightgate = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate))
510         lightgate = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate), 1)
511         lightgate2 = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate2, mov.y_lightgate2, mov.width_lightgate2, mov.height_lightgate2))
512         lightgate2 = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate2, mov.y_lightgate2, mov.width_lightgate2, mov.height_lightgate2),1)
513         #--Treating these as rectangles but with very small heights
514         lightray = pygame.draw.rect(win, pract.RED,(mov.x_lightray, mov.y_lightray, mov.width_lightray, mov.height_lightray))
515         lightray2 = pygame.draw.rect(win, pract.RED,(mov.x_lightray2, mov.y_lightray2, mov.width_lightray2, mov.height_lightray2))
516         #--Drawing electromagnets as rectangles
517         electromagnet = pygame.draw.rect(win, pract.GREY,(mov.x_electromagnet, mov.y_electromagnet, mov.width_electromagnet,mov.height_electromagnet))
518         electromagnet = pygame.draw.rect(win, pract.BLACK,(mov.x_electromagnet, mov.y_electromagnet, mov.width_electromagnet,mov.height_electromagnet), 1)
519         #--Will be drawn as a circle
520         ball_bearing = pygame.draw.circle(win, pract.GREY,(mov.x_ball, mov.y_ball), mov.radius_ball)
521         ball_bearing = pygame.draw.circle(win, pract.BLACK,(mov.x_ball, mov.y_ball), mov.radius_ball,1)
522
523         #--Drawing to the screen the various labels at the specified x and y coordinates
524         win.blit(mov.clampStand_label, (mov.x_clampStandlabel, mov.y_clampStandlabel))
525         win.blit(mov.lightgates_label, (mov.x_lightgateslabel, mov.y_lightgateslabel))
526         win.blit(mov.lightgates_label, (mov.x_lightgateslabel2, mov.y_lightgateslabel2))
527         win.blit(mov.magnet_label, (mov.x_magnetlabel, mov.y_magnetlabel))
528         win.blit(mov.bearings_label, (mov.x_bearingslabel, mov.y_bearingslabel))
529
530         #--Will be drawn as a circle
531         ball_bearing = pygame.draw.circle(win, pract.GREY,(mov.x_ball, mov.y_ball), mov.radius_ball)
532         ball_bearing = pygame.draw.circle(win, pract.BLACK,(mov.x_ball, mov.y_ball), mov.radius_ball,1)
```

Development Testing:

Play button	Valid input – button is pressed	Animation is played	User can watch the animation	4.1 Part 1

Below is what happened when I pressed the spacebar. Ideally, the fall should fall toward the bottom of the screen.



Validation:

I have also tested for validation to make sure that the correct input has been received by the program.

Result	Valid input – Pressing the spacebar	Invalid input – Pressing a key that is not the spacebar
Expected outcome?	Yes, the ball fell	Yes, the ball did not fall

The correct input has been received by the program.

When the spacebar was pressed, the screen produced was not the result I wanted. Firstly, while the ball technically moves, the position of the ball before the new position is still there. Therefore, I will need to clear the screen every time the ball moves. Additionally, the ball goes past the boundary of the Pygame window meaning I will need to set a condition to check whether the ball has gone past it. This test was not a success so I will detail my solution to solving these problems.

Additionally, since my Physics practical is on the force of gravity, the ball will need to accelerate. I will move on to this later once I have my ball move in a linear motion.

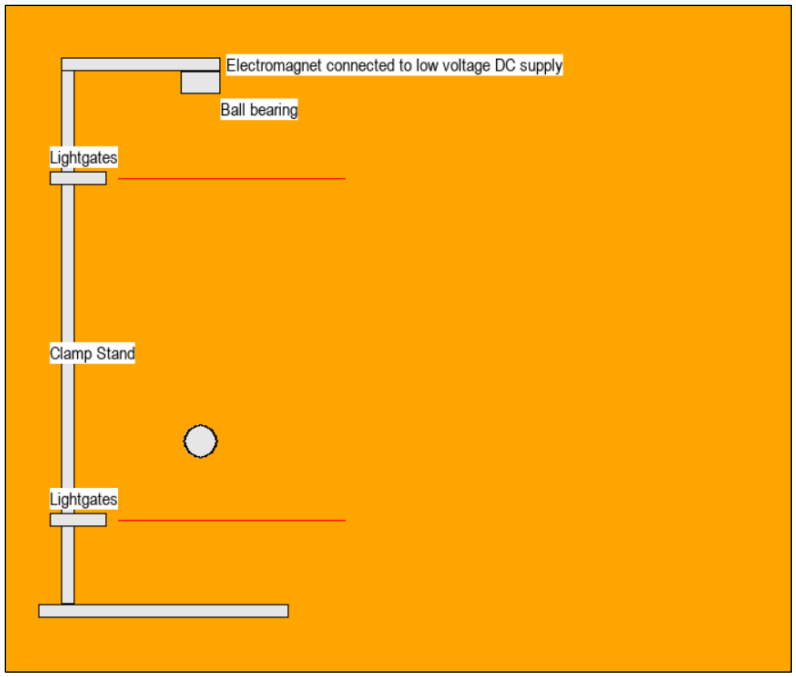
Solution:

In order to remove the previous position of the ball, I will clear the screen every time the while loop iterates. I have done this twice depending on whether the user presses the spacebar or not.

```
463            win.fill(pract.ORANGE)
```

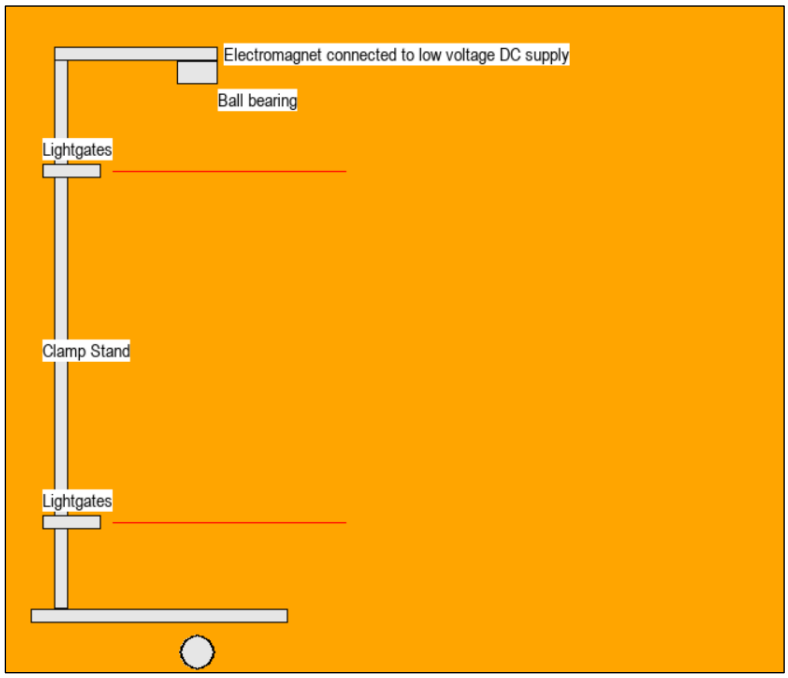
```
500            win.fill(pract.ORANGE)
```

Now the screen looks like this:



Next, I will add conditions to prevent the ball from going past the Pygame screen like below:

```
462         if keys[pygame.K_SPACE] and mov.y_ball< 570: #--When the spacebar is pressed
```



Now this test was a success since the student can interact with the animation which fulfils my success criteria. I will refine this process further and will include further tests on aspects such as whether the ball accelerates or not.

Further Refinements:

Next, I will add a pad to the base of the stand and a counterweight to the other side of the base to improve the clarity of the diagram. This will mean moving most of the objects to the right slightly.

Here are the coordinates and dimensions of the pad and counterweight:

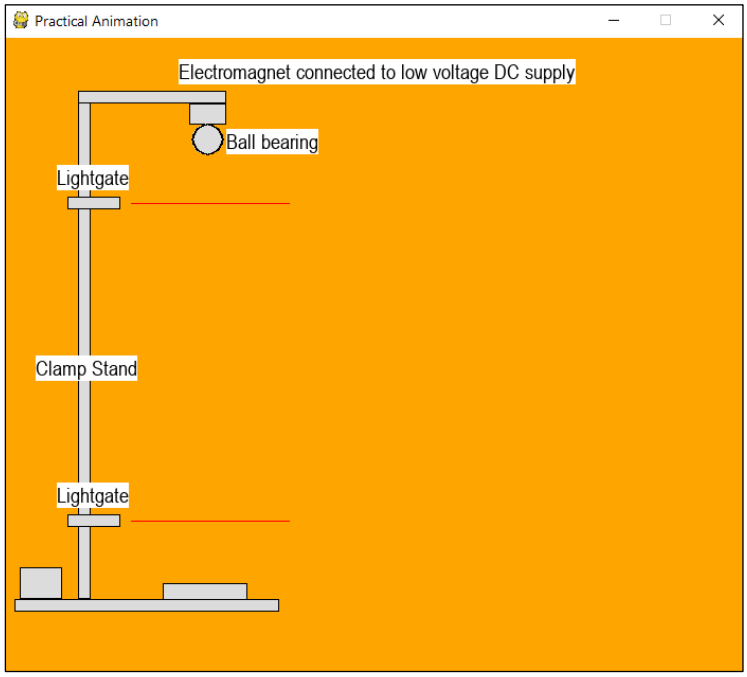
```
306         self.x_pad = 150
307         self.y_pad = 515
308         self.x_counterweight = 15
309         self.y_counterweight = 500
```

```
339         self.width_pad = 80
340         self.height_pad = 16
341         self.width_counterweight = 40
342         self.height_counterweight = 30
```

Below is the code for the rectangles that will be drawn:

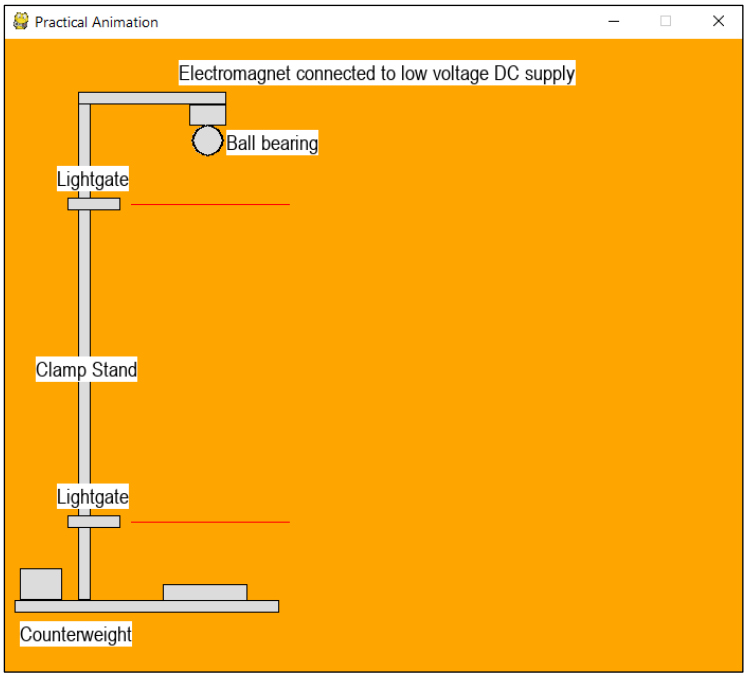
```
485         pad = pygame.draw.rect(win, pract.GREY,(mov.x_pad, mov.y_pad, mov.width_pad, mov.height_pad))
486         pad = pygame.draw.rect(win, pract.GREY,(mov.x_pad, mov.y_pad, mov.width_pad, mov.height_pad), 1)
487         counterweight = pygame.draw.rect(win, pract.GREY,(mov.x_counterweight, mov.y_counterweight, mov.width_counterweight, mov.height_counterweight))
488         counterweight = pygame.draw.rect(win, pract.GREY,(mov.x_counterweight, mov.y_counterweight, mov.width_counterweight, mov.height_counterweight), 1)
```

These rectangles follow the same design pattern as the other objects.



Next, I will add a label for the counterweight since it may be more obscure for a student compared to the pad.

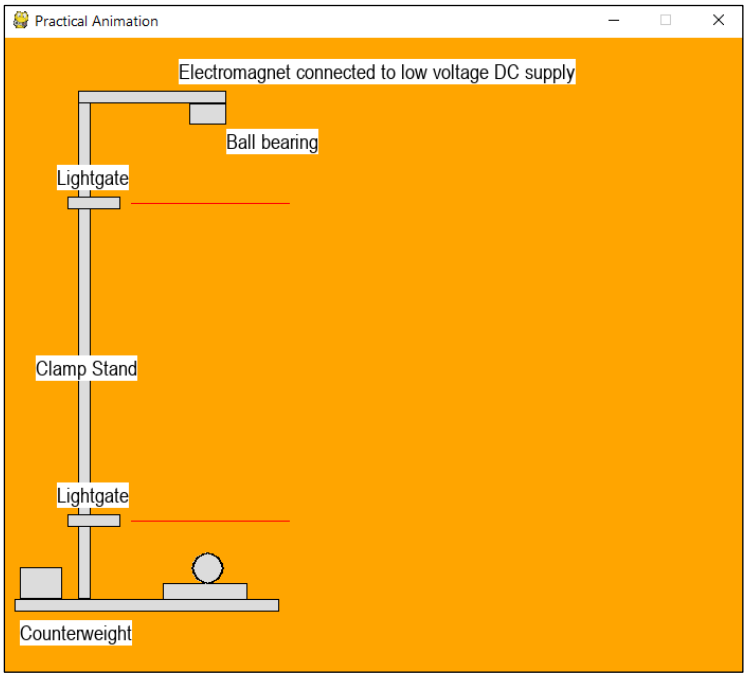
```
365     self.counterweight_label = pract.fontObj_small.render("Counterweight", True, pract.BLACK, pract.WHITE)
```



Moreover, I have changed the limit of the ball's movement so that it always lands on the pad.

```
481         if keys[pygame.K_SPACE] and mov.y_ball< 500: #--When the spacebar is pressed
```

Even though the y-coordinate of the pad is at 515, the ball's coordinates are measured at the centre rather than the top right like in rectangles. Therefore, I have subtracted 15 giving a limit of 500.



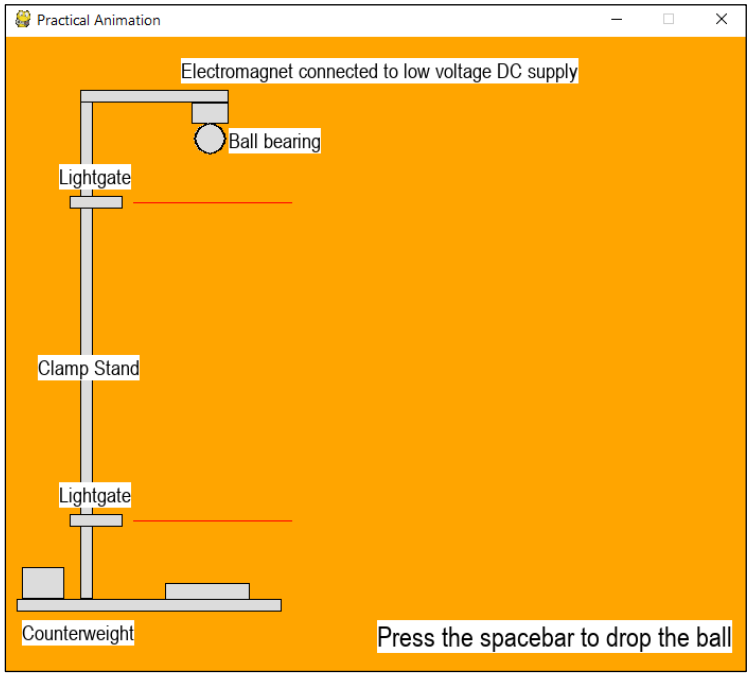
Finally, for the animation to work, the user must press the spacebar for the ball to drop. Therefore, I have created a new label which tells the user to do. This is a usability feature and makes sure that the program is easy to use. When the ball is dropped, I plan for the spacebar to be pressed again so the ball can be returned to the original position and so the experiment can be repeated.

```
150     self.press_spacebar_label = pract.fontObj_medium.render("Press the spacebar to drop the ball", True, pract.BLACK, pract.WHITE)
```

```
166         self.x_press_spacebarlabel = 350
167         self.y_press_spacebarlabel = 550
```

```
225         win.blit(mov.press_spacebar_label, (mov.x_press_spacebarlabel, mov.y_press_spacebarlabel))
```

This gives the screen:



Development Testing:

Play button	Valid input – button is pressed	Animation is played	User can watch the animation	4.1 Part 2

Now I have made several refinements to the falling motion of the ball, I will conduct a second test. I can confirm this falling motion of the ball through its coordinates in the Python shell.

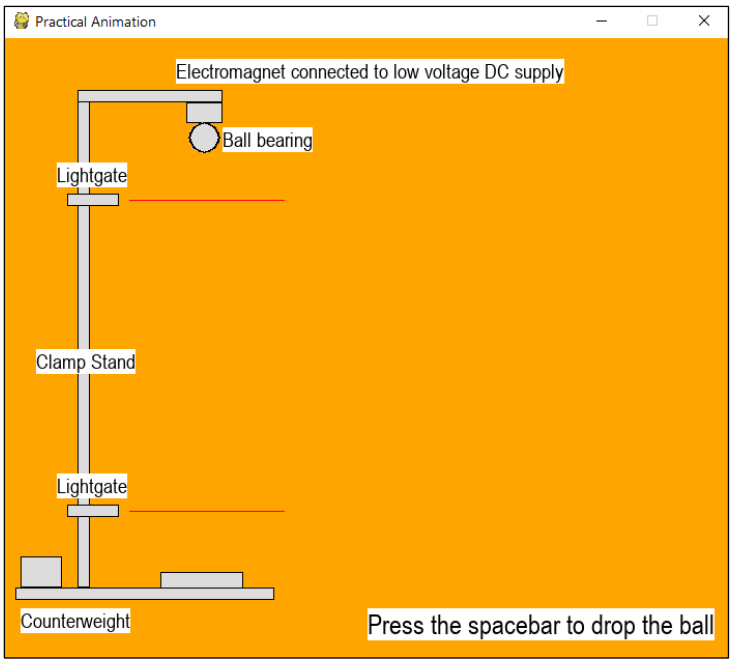
Start	End
99	447
102	450
105	453
108	456
111	459
114	462
117	465
120	468
123	471
126	474
129	477
132	480
135	483
138	486
141	489
144	492
147	495
150	498
153	501

Therefore, the motion of the ball increases linearly as it falls to the pad. This test was a success.

Development Testing:

If any buttons or sliders are not pressed	Valid input – no buttons are pressed	The animation will either stay static or will carry on without the prompted action that would have been initiated by the button	Allows the user the choice of whether to interact	4.6

When the spacebar is not pressed:



The above screen is displayed indefinitely and nothing occurs on the screen. This test is a success.

Acceleration of the ball:

To make the ball accelerate due to gravity, I will need to increase the value of the 'mov.acceleration' attribute through every loop so the ball gains larger values of speed as time increases:

```
525         mov.y_ball += mov.acceleration #--Moves the ball downwards
526         mov.acceleration += 1
527         ball_bearing = pygame.draw.circle(win, pract.GREY,(mov.x_ball, (mov.y_ball)), mov.radius_ball)
528         ball_bearing = pygame.draw.circle(win, pract.BLACK,(mov.x_ball, (mov.y_ball)), mov.radius_ball,1)
```

When testing this out, the ball instantly fell from its original position to the pad with no visible acceleration for a student to see. Therefore, I will reduce the value of 'mov.acceleration' to less than one.

```
525         mov.y_ball += mov.acceleration #--Moves the ball downwards
526         mov.acceleration += 0.005
527         ball_bearing = pygame.draw.circle(win, pract.GREY,(mov.x_ball, (mov.y_ball)), mov.radius_ball)
528         ball_bearing = pygame.draw.circle(win, pract.BLACK,(mov.x_ball, (mov.y_ball)), mov.radius_ball,1)
```

This resulted in the error below. Since Pygame doesn't like coordinates which are float values, I will convert the ball's coordinates to integers.

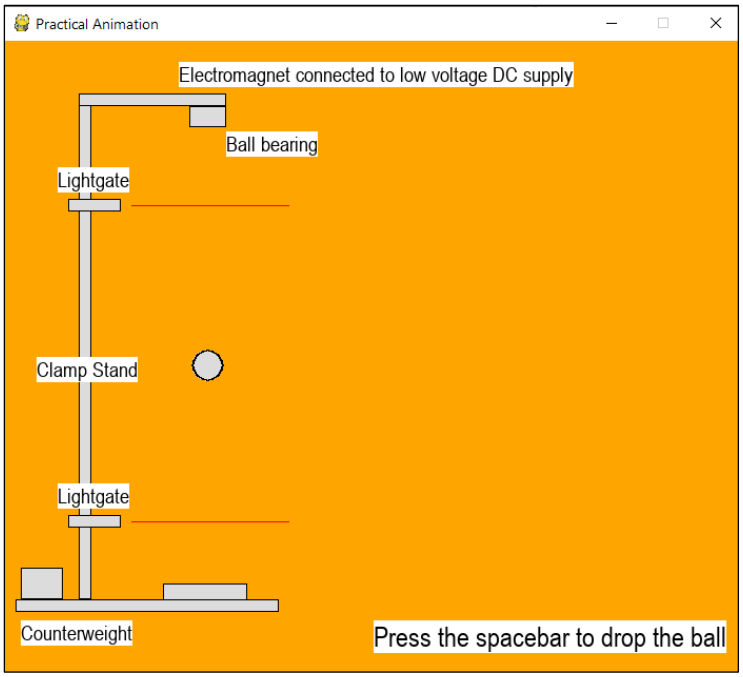
```
Traceback (most recent call last):
  File "C:\Users\Mayur Shankar\OneDrive\Documents\new\A Level Project\Programming Project - Final.py"
, line 581, in <module>
    command = PracticalAnimation() #--Allows the program to run
  File "C:\Users\Mayur Shankar\OneDrive\Documents\new\A Level Project\Programming Project - Final.py"
, line 460, in __init__
    command = self.AnimationInteraction() #--Moves onto the next method
  File "C:\Users\Mayur Shankar\OneDrive\Documents\new\A Level Project\Programming Project - Final.py"
, line 527, in AnimationInteraction
    ball_bearing = pygame.draw.circle(win, pract.GREY,(mov.x_ball, (mov.y_ball)), mov.radius_ball)
TypeError: integer argument expected, got float
>>>
```

```
525         mov.y_ball += mov.acceleration #--Moves the ball downwards
526         mov.acceleration += 0.005
527         ball_bearing = pygame.draw.circle(win, pract.GREY,(mov.x_ball, int(mov.y_ball)), mov.radius_ball)
528         ball_bearing = pygame.draw.circle(win, pract.BLACK,(mov.x_ball, int(mov.y_ball)), mov.radius_ball,1)
```

Development Testing:

Play button	Valid input – button is pressed	Animation is played	User can watch the animation	4.1 Part 3

The ball should now accelerate as it falls to the pad. When testing, I did not receive an error which was good however my ball didn't fall all the way to the pad.



Validation:

I have also tested for validation to ensure that the user entered the correct input as this could have caused the error.

Result	Valid input – Pressing the spacebar	Invalid input – Pressing a key that is not the spacebar	Boundary input – Holding down the spacebar rather than pressing it
Expected output?	No, the ball did not fall as expected	Yes, the ball did not fall	Yes, the ball fell as expected

While my invalid input test was as expected, my valid input test was unexpected. When pressing the spacebar, the ball only dropped part of the way to the bottom. Yet, when holding down the spacebar, the ball fell as normal.

I quickly realised that the ball's falling motion was dependent on how long I had pressed the spacebar for. This test was not a success since I wanted the user to press the spacebar once for the ball to drop rather than holding it down.

Since 0.005 is much smaller than 1, the ball moved far less in the same period of time when pressing the spacebar. To resolve this, I will alter my while loop and if/else statements so when the space bar is pressed, the ball will fall to the bottom indefinitely.

To begin with, I have introduced a new attribute which I have set to false. This is a Boolean attribute which will detect whether the space bar has been pressed.

```
385        self.pressed = False
```

Below is my while loop for the beginning of the 'AnimationInteraction' class. On line 481 I have used an 'if not' statement. Since I have set 'mov.pressed' to false, this statement will be fulfilled, and the

program will detect for any space bar presses. If there is a press, the Boolean attribute will now be set to True and the while loop iterates. On the next iteration, line 481 will not be fulfilled, therefore the falling ball motion can be simulated after line 483.

```
479     while True:
480       keys = pygame.key.get_pressed() #--Detects user's key inputs
481       if not (mov.pressed):
482         if keys[pygame.K_SPACE]:
483           mov.pressed = True
```

While the program is checking for any space bar presses, I have displayed the diagram of the practical in the else statement on line 485. There will be no motion in this until the user pressed the spacebar.

```
481     if not (mov.pressed):
482       if keys[pygame.K_SPACE]:
483         mov.pressed = True
484
485       else:
486         win.fill(pract.ORANGE)
487         stand = pygame.draw.rect(win, pract.GREY,(mov.x_stand, mov.y_stand, mov.width_stand, mov.height_stand))
488         #--An identical rectangle is created for the border
489         stand = pygame.draw.rect(win, pract.BLACK,(mov.x_stand, mov.y_stand, mov.width_stand, mov.height_stand), 1)
490         #--Rectangles for the base of the stand
491         base = pygame.draw.rect(win, pract.GREY,(mov.x_base, mov.y_base, mov.width_base, mov.height_base))
492         base = pygame.draw.rect(win, pract.BLACK,(mov.x_base, mov.y_base, mov.width_base, mov.height_base), 1)
493         #--Rectangles for the top of the screen
494         clamp = pygame.draw.rect(win, pract.GREY,(mov.x_clamp, mov.y_clamp, mov.width_clamp, mov.height_clamp))
495         clamp = pygame.draw.rect(win, pract.BLACK,(mov.x_clamp, mov.y_clamp, mov.width_clamp, mov.height_clamp), 1)
496         #--Rectangles for the pad and counterweight
497         pad = pygame.draw.rect(win, pract.GREY,(mov.x_pad, mov.y_pad, mov.width_pad, mov.height_pad))
498         pad = pygame.draw.rect(win, pract.BLACK,(mov.x_pad, mov.y_pad, mov.width_pad, mov.height_pad), 1)
499         counterweight = pygame.draw.rect(win, pract.GREY,(mov.x_counterweight, mov.y_counterweight, mov.width_counterweight, mov.height_counterweight))
500         counterweight = pygame.draw.rect(win, pract.BLACK,(mov.x_counterweight, mov.y_counterweight, mov.width_counterweight, mov.height_counterweight), 1)
501         #--Drawing lightgates as rectangles
502         lightgate = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate))
503         lightgate = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate), 1)
504         lightgate2 = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate2, mov.y_lightgate2, mov.width_lightgate2, mov.height_lightgate2))
505         lightgate2 = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate2, mov.y_lightgate2, mov.width_lightgate2, mov.height_lightgate2),1)
506         #--Treating these as rectangles but with very small heights
507         lightray = pygame.draw.rect(win, pract.RED,(mov.x_lightray, mov.y_lightray, mov.width_lightray, mov.height_lightray))
508         lightray2 = pygame.draw.rect(win, pract.RED,(mov.x_lightray2, mov.y_lightray2, mov.width_lightray2, mov.height_lightray2))
509         #--Drawing electromagnets as rectangles
510         electromagnet = pygame.draw.rect(win, pract.GREY,(mov.x_electromagnet, mov.y_electromagnet, mov.width_electromagnet,mov.height_electromagnet))
511         electromagnet = pygame.draw.rect(win, pract.BLACK,(mov.x_electromagnet, mov.y_electromagnet, mov.width_electromagnet,mov.height_electromagnet), 1)
512         #--Will be drawn as a circle
513         ball_bearing = pygame.draw.circle(win, pract.GREY,(mov.x_ball, mov.y_ball), mov.radius_ball)
514         ball_bearing = pygame.draw.circle(win, pract.BLACK,(mov.x_ball, mov.y_ball), mov.radius_ball,1)
515
516         #--Drawing to the screen the various labels at the specified x and y coordinates
517         win.blit(mov.clampStand_label, (mov.x_clampStandlabel, mov.y_clampStandlabel))
518         win.blit(mov.counterweight_label, (mov.x_counterweightlabel, mov.y_counterweightlabel))
519         win.blit(mov.lightgates_label, (mov.x_lightgateslabel, mov.y_lightgateslabel))
520         win.blit(mov.lightgates_label, (mov.x_lightgateslabel2, mov.y_lightgateslabel2))
521         win.blit(mov.magnet_label, (mov.x_magnetlabel, mov.y_magnetlabel))
522         win.blit(mov.bearings_label, (mov.x_bearingslabel, mov.y_bearingslabel))
523         win.blit(mov.press_spacebar_label, (mov.x_press_spacebarlabel, mov.y_press_spacebarlabel))
524
525         #--Will be drawn as a circle
526         ball_bearing = pygame.draw.circle(win, pract.GREY,(mov.x_ball, mov.y_ball), mov.radius_ball)
527         ball_bearing = pygame.draw.circle(win, pract.BLACK,(mov.x_ball, mov.y_ball), mov.radius_ball,1)
```

The screenshot below details the second else statement that is run when the spacebar is pressed (this bypasses the if not statement). Again, from lines 531 to 569, the same diagram of the practical is displayed but I have introduced a new if and else statement on lines 571 and 573. Line 573 checks whether the ball has landed on the pad – if so, the while loop is broken and a static display of the practical will be shown. Line 573 allows the ball's speed to increase over time – as the loop repeats, line 531 onwards redraws the entire diagram again with a new position of the ball.

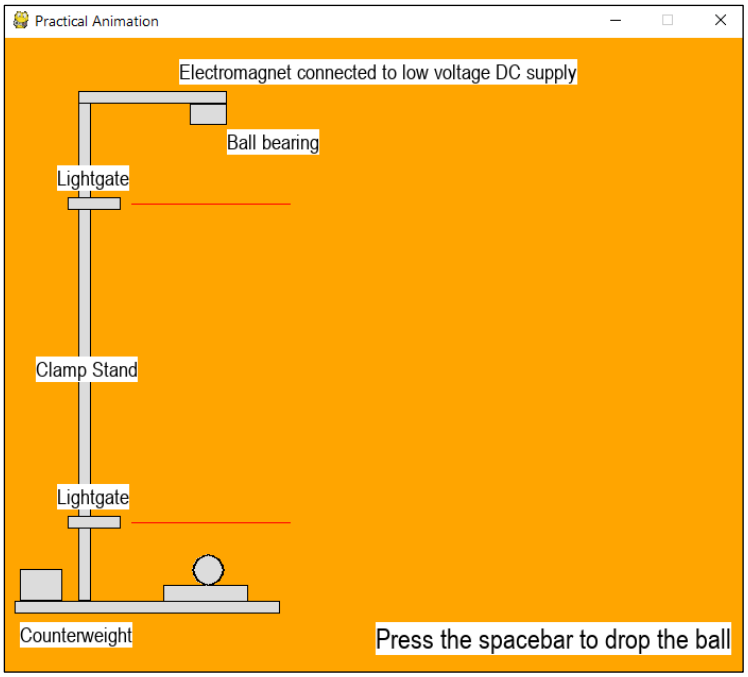
```
    else:
      win.fill(pract.ORANGE)
      #--Practical diagram is redrawn
      stand = pygame.draw.rect(win, pract.GREY,(mov.x_stand, mov.y_stand, mov.width_stand, mov.height_stand))
      #--An identical rectangle is created for the border
      stand = pygame.draw.rect(win, pract.BLACK,(mov.x_stand, mov.y_stand, mov.width_stand, mov.height_stand), 1)
      #--Rectangles for the base of the stand
      base = pygame.draw.rect(win, pract.GREY,(mov.x_base, mov.y_base, mov.width_base, mov.height_base))
      base = pygame.draw.rect(win, pract.BLACK,(mov.x_base, mov.y_base, mov.width_base, mov.height_base), 1)
      #--Rectangles for the top of the stand
      clamp = pygame.draw.rect(win, pract.GREY,(mov.x_clamp, mov.y_clamp, mov.width_clamp, mov.height_clamp))
      clamp = pygame.draw.rect(win, pract.BLACK,(mov.x_clamp, mov.y_clamp, mov.width_clamp, mov.height_clamp), 1)
      #--Rectangles for the pad and counterweight
      pad = pygame.draw.rect(win, pract.GREY,(mov.x_pad, mov.y_pad, mov.width_pad, mov.height_pad))
      pad = pygame.draw.rect(win, pract.BLACK,(mov.x_pad, mov.y_pad, mov.width_pad, mov.height_pad), 1)
      counterweight = pygame.draw.rect(win, pract.GREY,(mov.x_counterweight, mov.y_counterweight, mov.width_counterweight, mov.height_counterweight))
      counterweight = pygame.draw.rect(win, pract.BLACK,(mov.x_counterweight, mov.y_counterweight, mov.width_counterweight, mov.height_counterweight), 1)
      #--Drawing lightgates as rectangles
      lightgate = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate))
      lightgate = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate), 1)
      lightgate2 = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate2, mov.y_lightgate2, mov.width_lightgate2, mov.height_lightgate2))
      lightgate2 = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate2, mov.y_lightgate2, mov.width_lightgate2, mov.height_lightgate2), 1)
      #--Treating these as rectangles but with very small heights
      lightray = pygame.draw.rect(win, pract.RED,(mov.x_lightray, mov.y_lightray, mov.width_lightray, mov.height_lightray))
      lightray2 = pygame.draw.rect(win, pract.RED,(mov.x_lightray2, mov.y_lightray2, mov.width_lightray2, mov.height_lightray2))
      #--Drawing electromagnets as rectangles
      electromagnet = pygame.draw.rect(win, pract.GREY,(mov.x_electromagnet, mov.y_electromagnet, mov.width_electromagnet,mov.height_electromagnet))
      electromagnet = pygame.draw.rect(win, pract.BLACK,(mov.x_electromagnet, mov.y_electromagnet, mov.width_electromagnet,mov.height_electromagnet), 1)
      #--Will be drawn as a circle
      ball_bearing = pygame.draw.circle(win, pract.GREY,(mov.x_ball, int(mov.y_ball)), mov.radius_ball)
      ball_bearing = pygame.draw.circle(win, pract.BLACK,(mov.x_ball, int(mov.y_ball)), mov.radius_ball,1)

      #--Drawing to the screen the various labels at the specified x and y coordinates
      win.blit(mov.clampStand_label, (mov.x_clampStandlabel, mov.y_clampStandlabel))
      win.blit(mov.counterweight_label, (mov.x_counterweightlabel, mov.y_counterweightlabel))
      win.blit(mov.lightgates_label, (mov.x_lightgateslabel, mov.y_lightgateslabel))
      win.blit(mov.lightgates_label, (mov.x_lightgateslabel2, mov.y_lightgateslabel2))
      win.blit(mov.magnet_label, (mov.x_magnetlabel, mov.y_magnetlabel))
      win.blit(mov.bearings_label, (mov.x_bearingslabel, mov.y_bearingslabel))
      win.blit(mov.press_spacebar_label, (mov.x_press_spacebarlabel, mov.y_press_spacebarlabel))

      if mov.y_ball > 500:
        break
      else:
        mov.y_ball += mov.acceleration #--Moves the ball downwards
        mov.acceleration += 0.005
        ball_bearing = pygame.draw.circle(win, pract.GREY,(mov.x_ball, int(mov.y_ball)), mov.radius_ball)
        ball_bearing = pygame.draw.circle(win, pract.BLACK,(mov.x_ball, int(mov.y_ball)), mov.radius_ball,1)
```

Like with my other pieces of Pygame code, I have updated the display so the objects can be displayed onscreen.

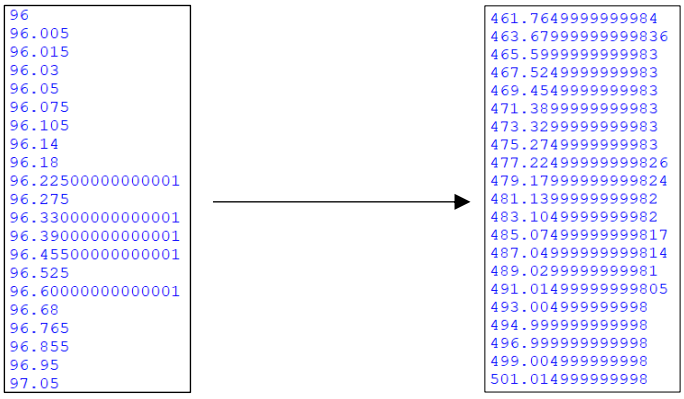
```
        #--While the program is running, checks whether the user is quitting the window
        for event in pygame.event.get():
          if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
        pygame.display.update()


      #--While the program is running, checks whether the user is quitting the window
      for event in pygame.event.get():
        if event.type == pygame.QUIT:
          pygame.quit()
          sys.exit()
      pygame.display.update()
```

Development Testing:

Play button	Valid input – button is pressed	Animation is played	User can watch the animation	4.1 Part 3



While it is hard for me to show the exact motion of the ball through screenshots, I can show this through the y-coordinates of the ball.



Since the increments are very small, I have decided to show coordinates of the ball at the beginning and the end. In the left-hand screenshot, the differences between each coordinate are very small whereas, in the right-hand screenshot, these differences have increased. This shows that the ball is accelerating.

Validation:

Result	Valid input – Pressing the spacebar	Invalid input – Pressing a key that is not the spacebar	Boundary input – Holding down the spacebar rather than pressing it
Expected output?	Yes, the ball fell as expected	Yes, the ball did not fall	Yes, the ball fell as expected

Pressing and holding the spacebar yielded the same result. Therefore, both my validation and development testing were a success.

Calculating the time taken for the ball to drop

To accurately measure the time taken for the ball to drop, I will need to tweak the ball's acceleration, so it follows acceleration due to gravity which is 9.8 m/s. To do this, I will assume that the distance between the two light gates is 1 metre. Using SUVAT equations, I can work out the time it should take for the ball fall.

This equation can be used to work out time it takes for the ball to fall: **$s = ut+1/2at^2$**

s = Distance (m)

u= Initial Speed (m/s)

v = Final Speed (m/s)

a = Acceleration due to Gravity (9.8m/s)

t = Time taken (s)

$1 = (0xt) +1/2 (9.8 \times t^2)$

Rearranging for t gives a time of 0.452 seconds or 452 milliseconds. This calculation doesn't take into account parameters such as air resistance which has an important effect on acceleration due to gravity. I will ignore such parameters as my program is an abstraction of real-life practicals.

Since the time taken for the ball to drop will be measured between the light gates, I will record the time when it passes both and then subtract the resulting times. The time will be displayed in milliseconds in the Python shell. Below is the code I have written to execute this.

```
574         if mov.y_ball == 156: #--When ball passes the first lightgate
575           self.lightgate_detect = pygame.time.get_ticks()
576
577         if mov.y_ball == 456: #--When ball passes the second lightgate
578           self.lightgate2_detect = pygame.time.get_ticks()
579           print(self.lightgate2_detect - self.lightgate_detect) #--Will print out the time
```

Development Testing:

Play button	Valid input – button is pressed	Animation is played	User can watch the animation	4.1 Part 4

When I ran the program, nothing was printed which confused me. I discovered that the ball may not always pass through the y-coordinates of 156 and 456. For example, when the ball is accelerating, it may move directly from the y-coordinate of 154 to 158, therefore, no time will be registered by line 575. To fix this, I reduced the value of the attribute 'mov.acceleration' to increase the chance that it will pass through the light rays. Additionally, I have initially set the starting value of this attribute to 2. Since I want my practical to simulate real-life conditions, as stated in my analysis, I have changed this value to 0 which represents the ball bearing's initial velocity – which should be 0. After using trial and improvement, I gave 'mov.acceleration' a starting value of 0 and a value of 0.00161 when incrementing. When repeating this test, the Python shell still did not print anything out. Since I was

using float values when incrementing the ball's coordinates, the chance of it passing 156 and 456 was small therefore I have introduced a small range of values like shown below:

```
574         if mov.y_ball > 155 and mov.y_ball < 157: #--When ball passes the first lightgate
575           self.lightgate_detect = pygame.time.get_ticks()
576
577         if mov.y_ball > 455 and mov.y_ball < 457: #--When ball passes the second lightgate
578           self.lightgate2_detect = pygame.time.get_ticks()
579           print(self.lightgate2_detect - self.lightgate_detect) #--Will print out the time
```

This is the result of the Python shell:

```
450
457
```

Thanks to trial and improvement, these times are very close to my calculated value of 452 milliseconds. Therefore, this test was successful.

There are two values printed since on lines 574 and 577 meaning the ball must have passed through the range twice. When repeating this test, I get different values which I found odd. This could be due to uncertainties to how Pygame records time or how I have used very small values when incrementing the 'mov.acceleration' attribute.

However, this could be very useful since it is representative of real-world conditions. While I was going to ignore discrepancies that occur in real-life, the fact that it is happening naturally in my program could be useful to a student so I will include it.

```
446
454
```

```
446
505
```

I get different values of time when I repeat the program.

To make code easier to write, I have stored this value as an attribute.

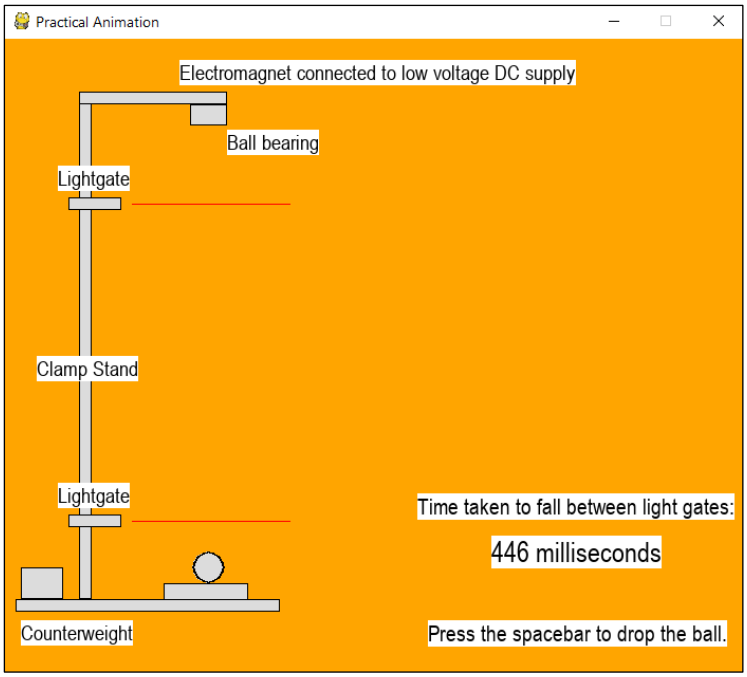
```
self.result_time = self.lightgate2_detect - self.lightgate_detect
```

After this, I will now work on a way for these recorded times to be displayed to the user on the Pygame window.

Displaying the times onscreen

Below, I have written the text to be displayed to the screen. On lines 589 and 590, I have written the character 'f' in front of the string. I have used this as it is much easier than dealing with the speech marks when writing a mixture of text and attributes in a single string This improves the readability of my code.

```
587         if mov.y_ball >= 500: #--If the ball begins to go past by the pad, the while loop will be broken
588           #--Displays the time taken for the ball to fall
589           self.display_timer = pract.fontObj_mediumsmall.render((f'Time taken to fall between light gates:'), True, pract.BLACK, pract.WHITE)
590           self.time_result = pract.fontObj_medium.render(str(f'{self.result_time} milliseconds'), True, pract.BLACK, pract.WHITE)
591           win.blit(self.display_timer, (mov.x_displaytimer,mov.y_displaytimer))
592           win.blit(self.time_result, (mov.x_timerresult,mov.y_timerresult))
```



I have made the font for the spacebar text slightly smaller so it matches the other text labels in the display.

Repeating the practical

When the ball has been dropped, I want the user to be able to re-press the spacebar so the experiment can be repeated. This fulfils my success criteria where the user should be able to watch the animation as many times as they want to.

Once the ball bearing has reached the pad, the if statement will check whether the spacebar is being pressed again. If so, the acceleration and the y-coordinate of the ball will be reset. Since I have not ended the while loop, the program will iterate but with the ball in its original position.

```
595            if keys[pygame.K_SPACE]: #--When space is pressed, the ball will be returned to its original position
596              mov.acceleration = 0
597              mov.y_ball = 96
```

To let the user know that they can repeat the practical, I will create a new text label that will only be displayed when the ball bearing has reached the pad.

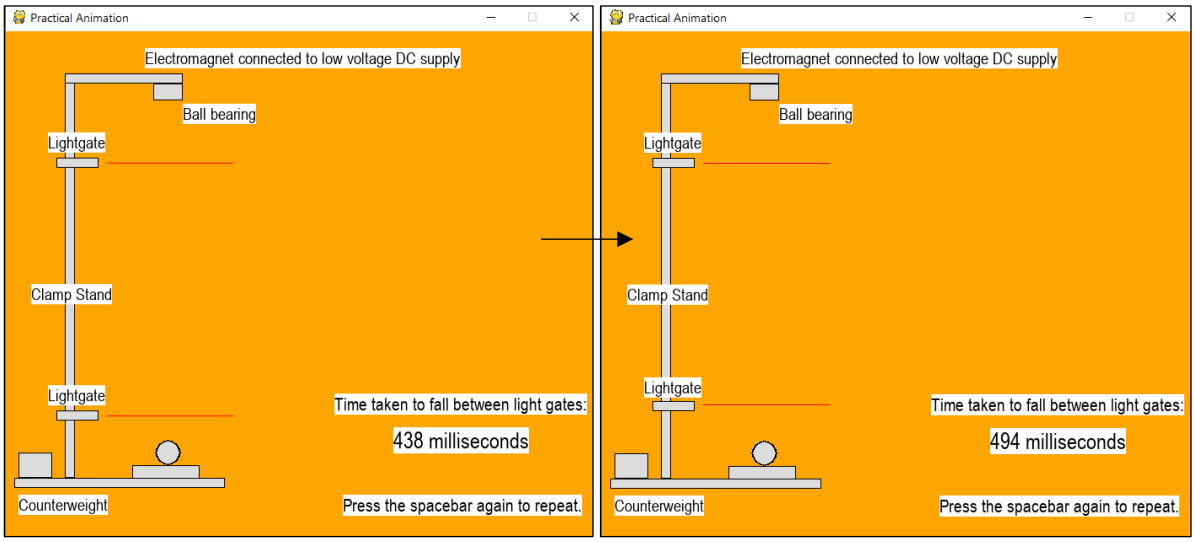
```
371    self.repeat_spacebar_label = pract.fontObj_mediumsmall.render("Press the spacebar again to repeat.", True, pract.BLACK, pract.WHITE)
```

```
593            #--Prompts the user to repeat the practical
594            win.blit(mov.repeat_spacebar_label, (mov.x_press_spacebarlabel, mov.y_press_spacebarlabel))
595            if keys[pygame.K_SPACE]: #--When space is pressed, the ball will be returned to its original position
596              mov.acceleration = 0
597              mov.y_ball = 96
```

Development Testing:

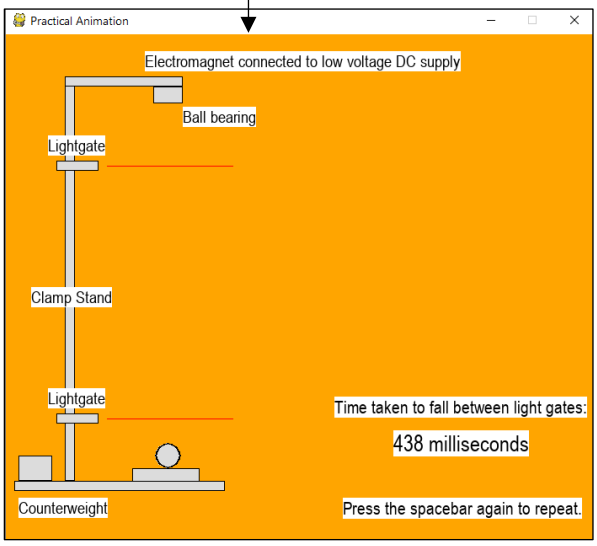
Repeat button	Valid input – button is pressed	Repeats the animation from the start	Allows the user to watch the animation as many times as they want	4.4

If any buttons or sliders are not pressed	Valid input – no buttons are pressed	The animation will either stay static or will carry on without the prompted action that would have been initiated by the button	Allows the user the choice of whether to interact	4.6



The user has the option to repeat the practical

The user then presses the spacebar repeating the practical



If the user doesn't press the spacebar, the diagram will remain static

Below, I have printed to the Python shell whether the practical has been repeated.

```
596         if keys[pygame.K_SPACE]: #--When space is pressed, the ball will be returned to its original position
597           mov.acceleration = 0
598           mov.y_ball = 96
599           print("Practical Repeated")
```

```
Practical Repeated
```

However, in my success criteria, I wanted my user to be able to repeat the practical as many times as they want to. To test this, I will repeatedly press the spacebar and see if any errors occur. This will be an example of a boundary test as I will testing the limits of the program.

```
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
```

After many repeats, the ball successfully dropped every time shown by the result in the Python shell. Therefore, test 4.1 and 4.6 have been a success and shows that my program is robust.

Changing positions of light gates:

I want the user to be able to change the position of the light gates to be able to see how the time it takes for the ball to fall depends on height.

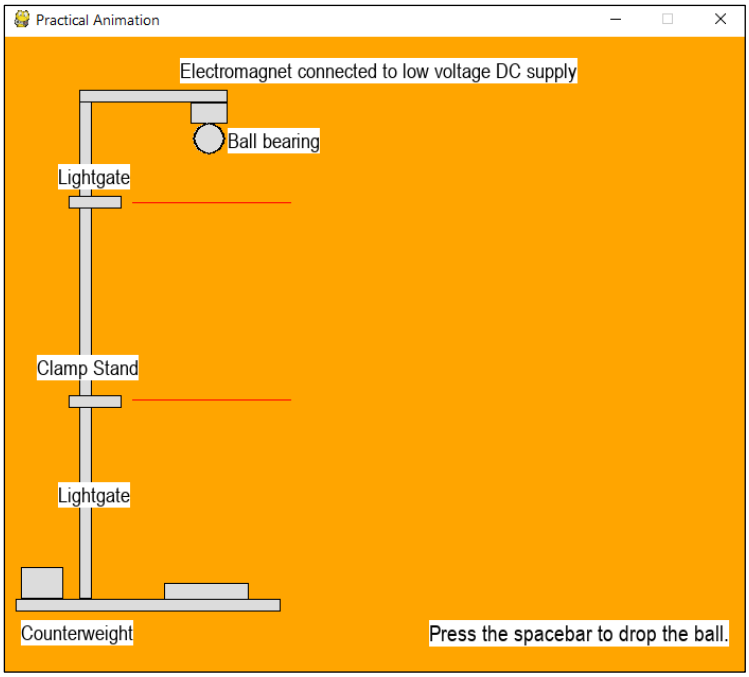
Below are some new coordinates for the light gates and light rays. Since I will be just changing the height, I can ignore the x coordinate and just change the y coordinate. Moreover, I will just be changing the coordinates of the lower light gate. Since my animation relates to real-life practicals, I have done this as it reduces percentage uncertainties when measuring height.

To begin with, I shall start with a height of 0.75m. Therefore, during the practical, the user will have the option to click a button that will change the height of the light gates to 0.75m from the default which is 1m. Once, the student can change the height to 0.75m, I can repeat this for other heights such as 0.5m and 0.25m.

Below, are some new coordinates for these light gates and light rays. These are three-quarters of my original coordinates which represented 1 metre.

```
    self.y_lightgate2_075m = 338
    self.y_lightray2_075m = 342
```

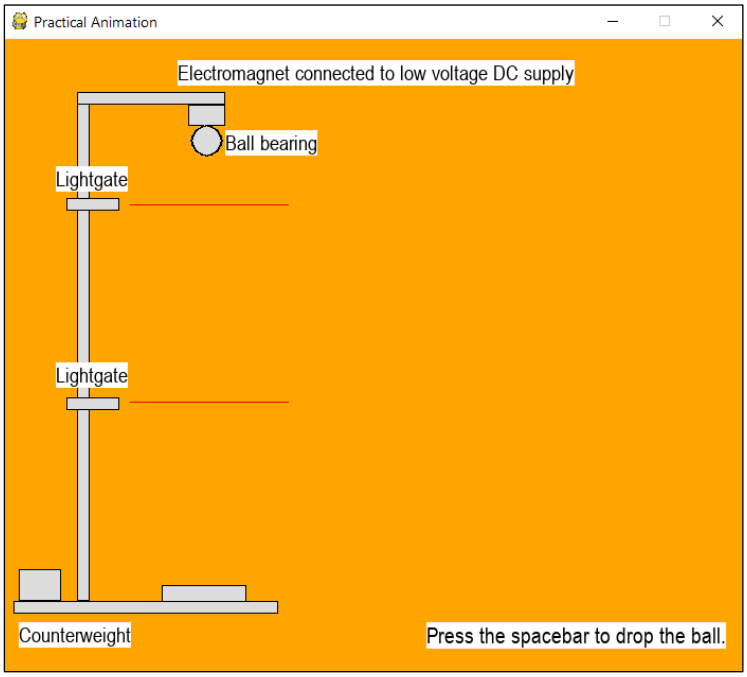
This is what the screen will look like with the new positions of the light gates and light rays:



I will need to move other widgets that are onscreen such as the light gate and clamp stand labels.

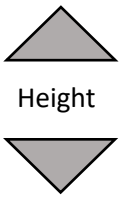
Below, is the new coordinate of the light gate label:

```
389     self.y_lightgateslabel2_075m = 305
```



I decided to remove the clamp stand text as I felt it cluttered the look of the program.

Next, I will create two up and down buttons in the shape of triangles. Compared to my screen designs, I think that this implementation would be much simpler than using a slider while still maintaining the engagement that a user will have when changing the height. I plan to have a piece of text in between the buttons which will change depending on whether they have been pressed. I have shown an example below.



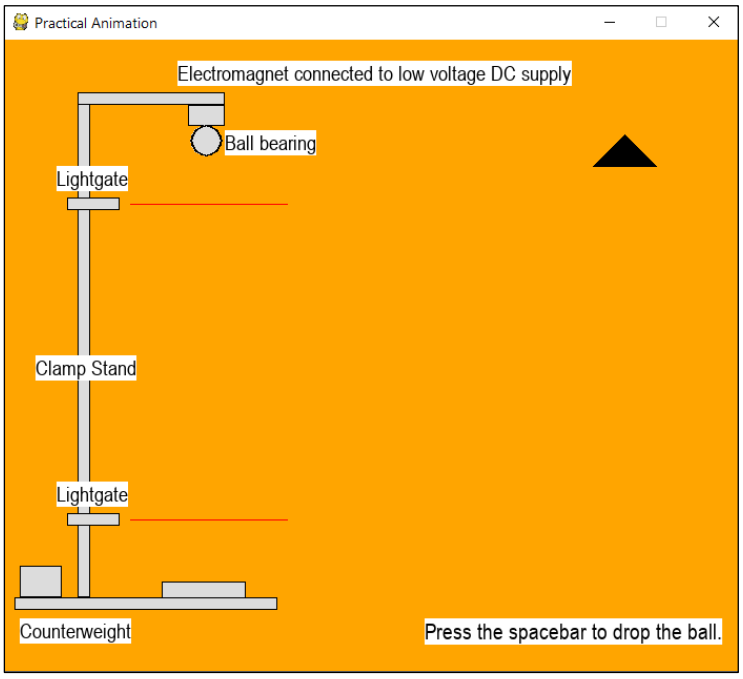
Below, I have used the pygame draw polygon method to draw a triangle.

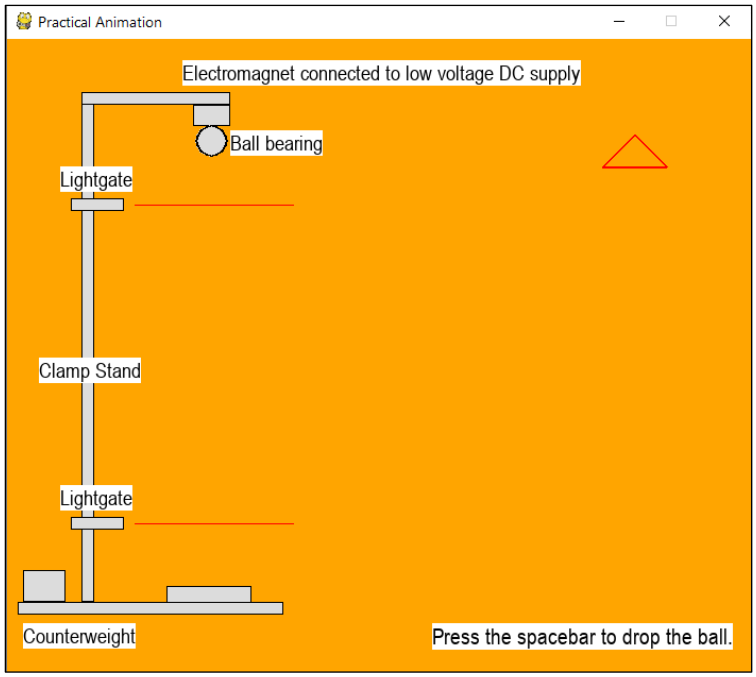
```
557            increaseheightbutton = pygame.draw.polygon(win, pract.BLACK, mov.coordinates_upbutton)
```

Below are the coordinates that were instantiated in the 'MovingAnimation' class.

```
415        self.coordinates_upbutton = ((560,120),(590,90), (620,120))
```

This is what the screen now looks like:





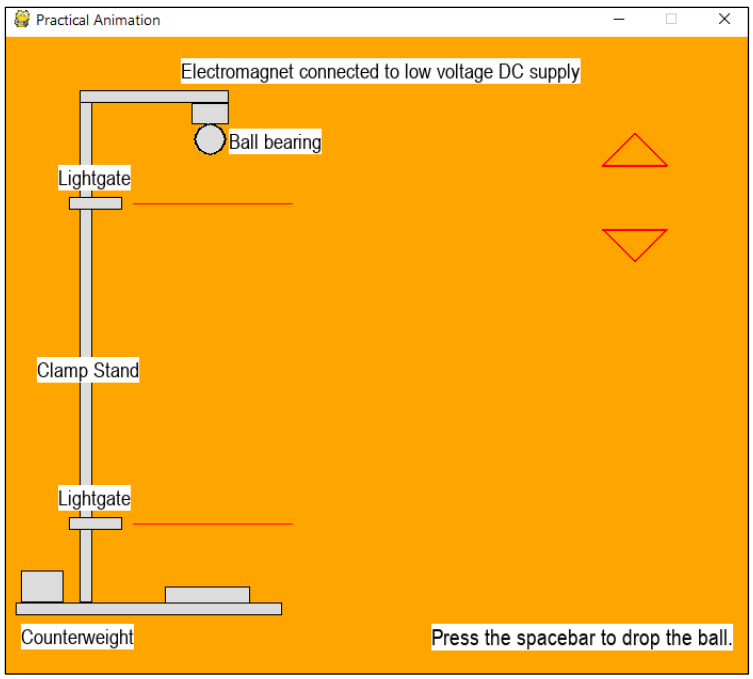

```
557          increaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_upbutton,2)
```

Shown in the above screenshot, I have changed the colour of the button and not filled it in to make it more aesthetically pleasing.

Below, is the screen and code for the down arrow. I have kept the x-coordinates the same but changed the y-coordinates:

```
559          decreaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_downbutton,2)
```

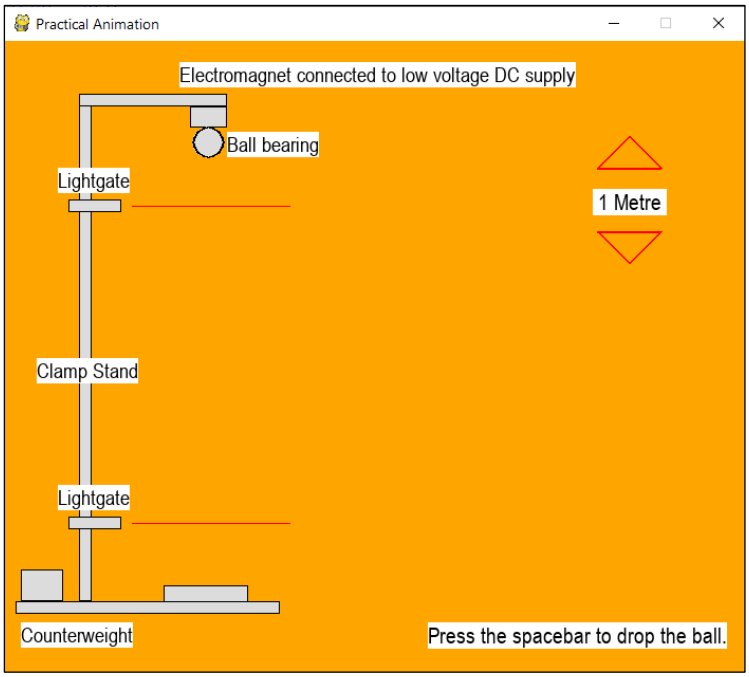
```
416      self.coordinates_downbutton = ((560,180),(590,210), (620,180))
```



Now, I will create the label which will be placed in between the two arrows.

```
560         height_1m_text = pract.fontObj_mediumsmall.render(' 1 Metre ', True, pract.BLACK, pract.WHITE)
561         win.blit(height_1m_text, (mov.x_height_1m_text, mov.y_height_1m_text))
```

```
412     self.x_height_1m_text = 556
413     self.y_height_1m_text = 140
```



Next, I will need to detect whether these buttons have been pressed. If so, the new positions of the light gate and light ray can be displayed.

After displaying the diagram, I have incorporated a new while loop that detects whether the user has pressed either button. Since 0.75m is less than 1m, I will be focusing on just on the bottom arrow for now.

```
565         #--Detects whether this button has been pressed
566         while pract.condition_lightgatebutton == True:
567           keys = pygame.key.get_pressed() #--Receives any key press
568           pos = pygame.mouse.get_pos() #--Receives any mouse press
569           print(pos)
570           #--Boundaries the user's mouse position has to be in for the bottom button to be pressed
571           if pos[0] > 560 and pos[0]< 620  and pos[1] > 180 and pos[1]< 210:
572             event = pygame.event.poll()
573             #--The mouse button has to also be pressed
574             if event.type == pygame.MOUSEBUTTONDOWN:
575               pract.condition_lightgatebutton == False
```

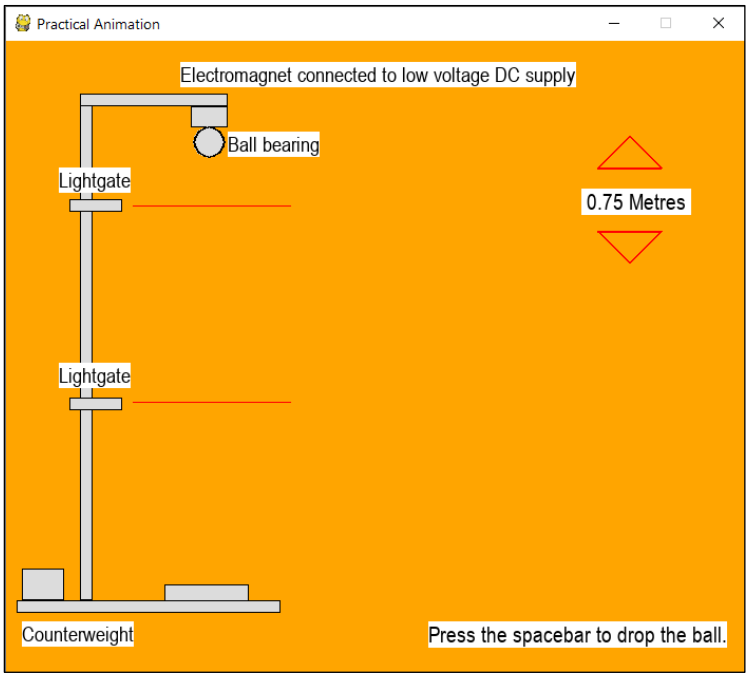
Above is the while loop which detects whether the cursor is in the correct boundaries. Although the button is a triangle, I have made my boundary in the shape of a rectangle to improve usability where there is a larger area for the user to press.

To display the new position of the new light gate, I will have to redisplay all the objects in the previous screen. Lines 611 to 622, show the new objects that have been displayed.

```
                win.fill(pract.ORANGE)
                stand = pygame.draw.rect(win, pract.GREY,(mov.x_stand, mov.y_stand, mov.width_stand, mov.height_stand))
                #--An identical rectangle is created for the border
                stand = pygame.draw.rect(win, pract.BLACK,(mov.x_stand, mov.y_stand, mov.width_stand, mov.height_stand), 1)
                #--Rectangles for the base of the stand
                base = pygame.draw.rect(win, pract.GREY,(mov.x_base, mov.y_base, mov.width_base, mov.height_base))
                base = pygame.draw.rect(win, pract.BLACK,(mov.x_base, mov.y_base, mov.width_base, mov.height_base), 1)
                #--Rectangles for the top of the screen
                clamp = pygame.draw.rect(win, pract.GREY,(mov.x_clamp, mov.y_clamp, mov.width_clamp, mov.height_clamp))
                clamp = pygame.draw.rect(win, pract.BLACK,(mov.x_clamp, mov.y_clamp, mov.width_clamp, mov.height_clamp), 1)
                #--Rectangles for the pad and counterweight
                pad = pygame.draw.rect(win, pract.GREY,(mov.x_pad, mov.y_pad, mov.width_pad, mov.height_pad))
                pad = pygame.draw.rect(win, pract.BLACK,(mov.x_pad, mov.y_pad, mov.width_pad, mov.height_pad), 1)
                counterweight = pygame.draw.rect(win, pract.GREY,(mov.x_counterweight, mov.y_counterweight, mov.width_counterweight, mov.height_counterweight))
                counterweight = pygame.draw.rect(win, pract.BLACK,(mov.x_counterweight, mov.y_counterweight, mov.width_counterweight, mov.height_counterweight), 1)
                #--Drawing electromagnets as rectangles
                electromagnet = pygame.draw.rect(win, pract.GREY,(mov.x_electromagnet, mov.y_electromagnet, mov.width_electromagnet,mov.height_electromagnet))
                electromagnet = pygame.draw.rect(win, pract.BLACK,(mov.x_electromagnet, mov.y_electromagnet, mov.width_electromagnet,mov.height_electromagnet), 1)
                #--Will be drawn as a circle
                ball_bearing = pygame.draw.circle(win, pract.GREY,(mov.x_ball, mov.y_ball), mov.radius_ball)
                ball_bearing = pygame.draw.circle(win, pract.BLACK,(mov.x_ball, mov.y_ball), mov.radius_ball,1)

                #--Drawing to the screen the various labels at the specified x and y coordinates
                win.blit(mov.counterweight_label, (mov.x_counterweightlabel, mov.y_counterweightlabel))
                win.blit(mov.lightgates_label, (mov.x_lightgateslabel, mov.y_lightgateslabel))
                win.blit(mov.lightgates_label, (mov.x_lightgateslabel2, mov.y_lightgateslabel2_075m))
                win.blit(mov.magnet_label, (mov.x_magnetlabel, mov.y_magnetlabel))
                win.blit(mov.bearings_label, (mov.x_bearingslabel, mov.y_bearingslabel))
                win.blit(mov.press_spacebar_label, (mov.x_press_spacebarlabel, mov.y_press_spacebarlabel))

                lightgate = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate))
                lightgate = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate), 1)
                lightgate2_075m = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate2, mov.y_lightgate2_075m, mov.width_lightgate2, mov.height_lightgate2))
                lightgate2_075m= pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate2, mov.y_lightgate2_075m, mov.width_lightgate2, mov.height_lightgate2), 1)

                #--Treating these as rectangles but with very small heights
                lightray = pygame.draw.rect(win, pract.RED,(mov.x_lightray, mov.y_lightray, mov.width_lightray, mov.height_lightray))
                lightray2_075m = pygame.draw.rect(win, pract.RED,(mov.x_lightray2, mov.y_lightray2_075m, mov.width_lightray2, mov.height_lightray2))

                #--Buttons the user can press to change the position of the light gates
                increaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_upbutton,2)
                decreaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_downbutton,2)
                height_075m_text = pract.fontObj_mediumsmall.render(' 0.75 Metres ', True, pract.BLACK, pract.WHITE)
                win.blit(height_075m_text, (mov.x_height_075m_text, mov.y_height_075m_text))
```

Development Testing:

Slider	Valid input – slider is moved	Allows the user to change an aspect of the practical such as height	Allows the user to interact with the practical	4.2



I have also printed to the Python shell to confirm that the height of 0.75 metres is being displayed.

```
0.75 metres will now be displayed
```

When the button is pressed, the screen successfully changes showing the new objects. This means that this test was a success.

Validation:

Result	Valid input – Clicking the down button	Invalid input – Clicking anything that isn't the down button	Boundary input – Holding down the mouse button when clicking
Expected output?	Yes, the screen changed	Yes, the screen did not change	Yes, the screen changed

I have validated that the user has performed the right input and all the outcomes were as expected.

To fulfil my success criteria, when the space bar is pressed, the ball should fall. Currently, nothing occurs. I realised that in the while loop, the program is forced to continually check whether the button has been pressed. To solve this, I have introduced a new if condition which checks whether the space bar has been pressed. If so, this while loop will be exited.

```
624             if keys[pygame.K_SPACE]: #--Checks whether the space bar has been pressed
625               pract.condition_lightgatebutton = False #--If so, this while loop will be ended
626
627             #--While the program is running, checks whether the user is quitting the window
628             for event in pygame.event.get():
629               if event.type == pygame.QUIT:
630                 pygame.quit()
631                 sys.exit()
632             pygame.display.update()
```

When the ball falls, I will need to change the screen based on which height the user has chosen. I have introduced a new attribute which keeps track of which height the user has chosen.

```
409     self.chooseheight = 0
```

Depending on what height the user has chosen, this attribute's value will change allowing the program to decide whether time needs to be measured between the light gates of distance 1 metre or 0.75 metres

When the down button is pressed, this attribute will decrease by a value of 1

```
577                 mov.chooseheight -=1
```

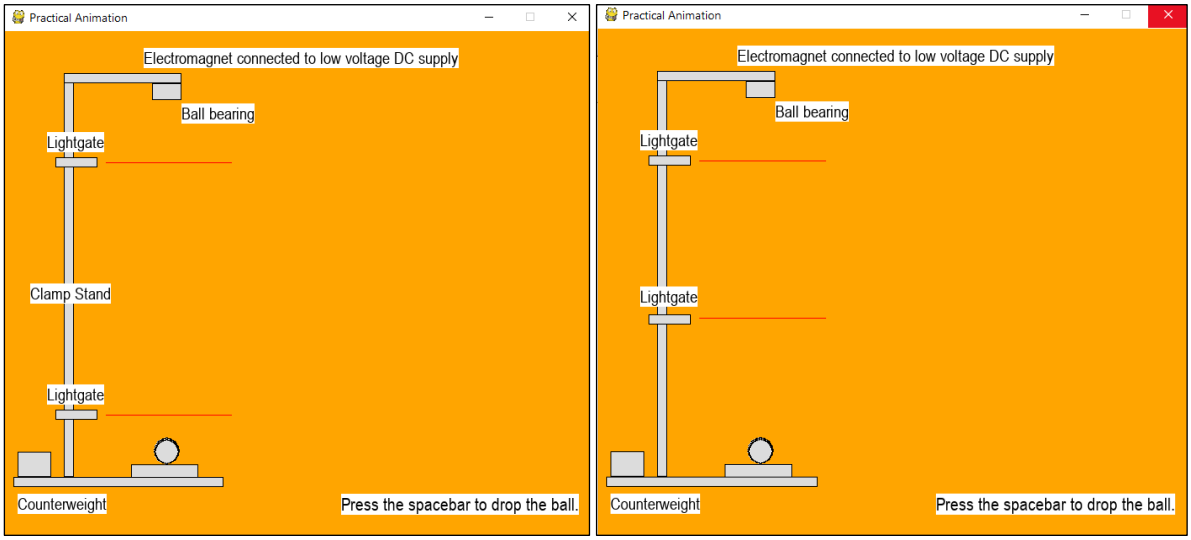
The code below shows what happens when the down button is pressed and then the spacebar is pressed after. This is part of the else statement which increments the ball's y-coordinate.

```
704         if mov.chooseheight == 0: #--When the up and down buttons are not pressed
705           #--Drawing the original light gates
706           lightgate = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate))
707           lightgate = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate), 1)
708           lightgate2 = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate2, mov.y_lightgate2, mov.width_lightgate2, mov.height_lightgate2))
709           lightgate2 = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate2, mov.y_lightgate2, mov.width_lightgate2, mov.height_lightgate2),1)
710           #--Treating these as rectangles but with very small heights
711           lightray = pygame.draw.rect(win, pract.RED,(mov.x_lightray, mov.y_lightray, mov.width_lightray, mov.height_lightray))
712           lightray2 = pygame.draw.rect(win, pract.RED,(mov.x_lightray2, mov.y_lightray2, mov.width_lightray2, mov.height_lightray2))
713           #--Drawing light gate label
714           win.blit(mov.lightgates_label, (mov.x_lightgateslabel2, mov.y_lightgateslabel2))
715           #--Drawing clamp stand label
716           win.blit(mov.clampStand_label, (mov.x_clampStandlabel, mov.y_clampStandlabel))
717
718         if mov.chooseheight == -1: #--When the down button is pressed once
719           #--Drawing the new light gates corresponding to a height of 0.75 metres
720           lightgate = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate))
721           lightgate = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate), 1)
722           lightgate2_075m = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate2, mov.y_lightgate2_075m, mov.width_lightgate2, mov.height_lightgate2))
723           lightgate2_075m = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate2, mov.y_lightgate2_075m, mov.width_lightgate2, mov.height_lightgate2), 1)
724           #--Treating these as rectangles but with very small heights
725           lightray = pygame.draw.rect(win, pract.RED,(mov.x_lightray, mov.y_lightray, mov.width_lightray, mov.height_lightray))
726           lightray2_075m = pygame.draw.rect(win, pract.RED,(mov.x_lightray2, mov.y_lightray2_075m, mov.width_lightray2, mov.height_lightray2))
727           #--Drawing light gate label
728           win.blit(mov.lightgates_label, (mov.x_lightgateslabel2, mov.y_lightgateslabel2_075m))
```

This allows the ball to fall for both positions of light gates and light rays.

Development Testing:

Play button	Valid input – button is pressed	Animation is played	User can watch the animation	4.1 Part 5 – Ball falls for both heights



1 metre 0.75 metres

The ball successfully falls for both heights with the acceleration that I described in previous tests. This test is a success.

Displaying the time taken to fall for both heights

I will move on to displaying the time it takes for the ball to fall for both the heights and allowing the user to press the spacebar again so that they can repeat the experiment.

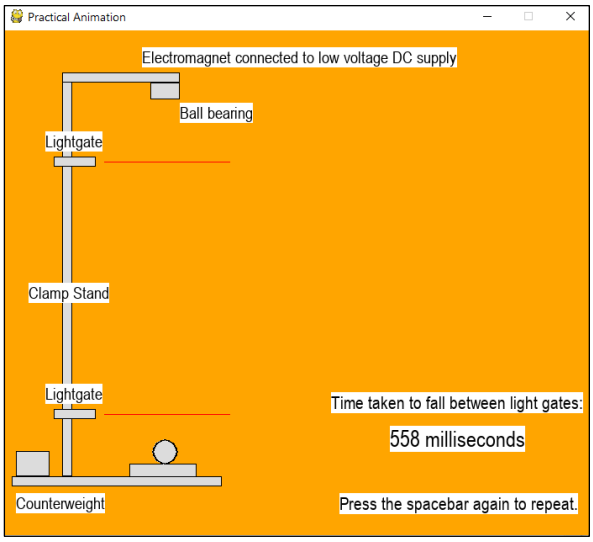
```
693         if mov.chooseheight == 0: #--If the light gates are in their original positions
694           if mov.y_ball > mov.y_lightray - 1 and mov.y_ball < mov.y_lightray + 1: #--When ball passes the first light gate
695             self.lightgate_detect = pygame.time.get_ticks() #--Will record time
696
697           if mov.y_ball > mov.y_lightray2 - 1 and mov.y_ball < mov.y_lightray2 + 1: #--When ball passes the second light gate
698             self.lightgate2_detect = pygame.time.get_ticks()
699             self.result_time = self.lightgate2_detect - self.lightgate_detect #--Stores the calculated time
700             print(self.result_time) #--Will print out the time in the shell for confirmation
701
702         if mov.chooseheight == -1: #--If the light gates are separated by a distance of 0.75 metres
703           if mov.y_ball > mov.y_lightray - 1 and mov.y_ball < mov.y_lightray + 1: #--When ball passes the first light gate
704             self.lightgate_detect = pygame.time.get_ticks() #--Will record time
705
706           if mov.y_ball > mov.y_lightray2_075m - 1 and mov.y_ball < mov.y_lightray2_075m + 1: #--When ball passes the second light gate
707             self.lightgate2_detect = pygame.time.get_ticks()
708             self.result_time = self.lightgate2_detect - self.lightgate_detect #--Stores the calculated time
709             print(self.result_time) #--Will print out the time in the shell for confirmation
```

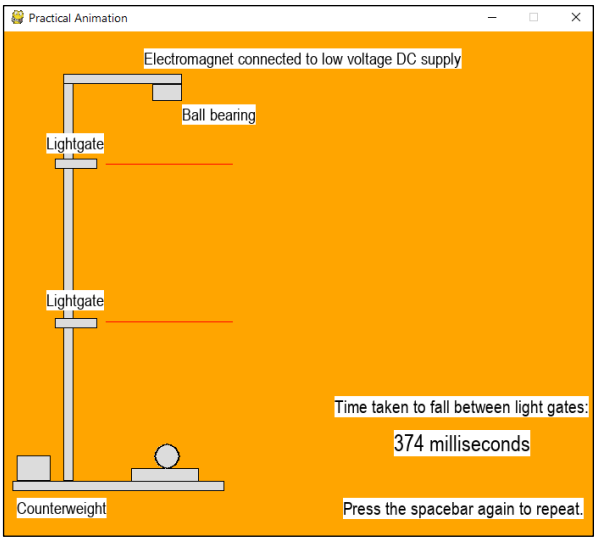
On line 702, I have added a new condition which detects whether the light gates have been moved by the user. When mov.chooseheight equals -1, the program knows that time must be measured for a height of 0.75 metres.

Development Testing:

Any other buttons or key presses	Valid input – button is pressed	There is a change onscreen	Allows the user to interact with the practical	Test 4.3 The time taken should be displayed for both heights

When the user presses the spacebar, the ball should accelerate towards the pad and the correct time taken should be displayed for either the height of 1 metre or 0.75 metres.





1 metre 0.75 metres

This test was a success.

Repeating the practical

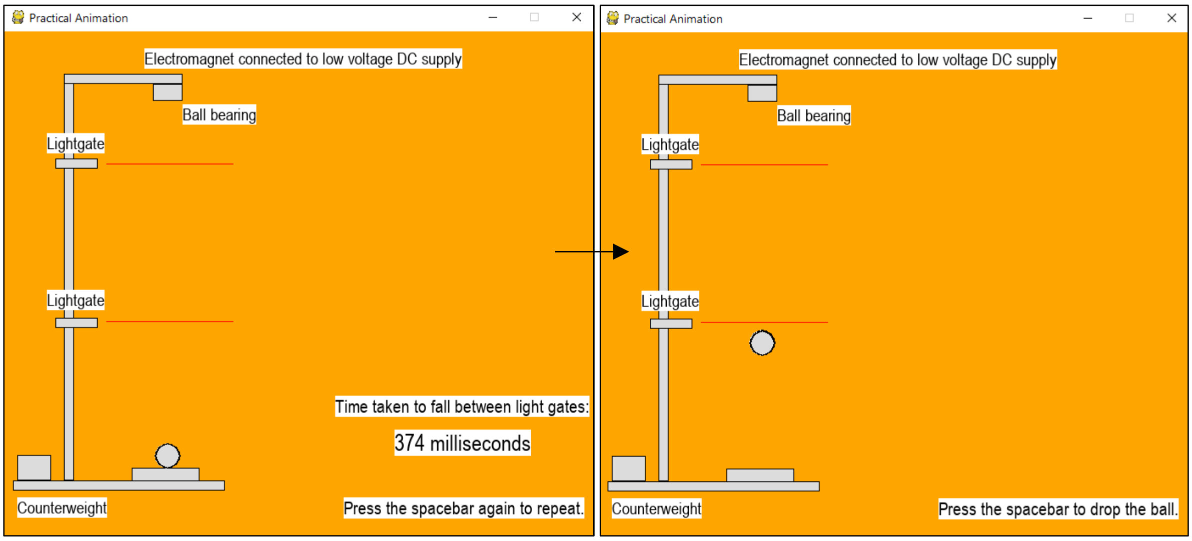
When the ball reaches the pad, the user can press the spacebar again to reset the ball's position.

```
720             if keys[pygame.K_SPACE]:
721                 mov.acceleration = 0
722                 mov.y_ball = 96
```

Development Testing:

Repeat button	Valid input – button is pressed	Repeats the animation from the start	Allows the user to watch the animation as many times as they want	4.4

For either of the heights, the user must be able to press the spacebar to repeat the practical.



The user has the option to repeat the practical

The user then presses the spacebar repeating the practical

I have also tested for boundary inputs when the user repeats the practical as many times as they want to for both heights.

```
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
Practical Repeated
```

This test was a success as I received no errors and the program ran as expected.

Changing the height between light gates

To improve the usability of the program, I will now let the user change the heights as many times as they want to. Currently, the user can only change the height from 1 metre to 0.75 metres and are stuck with this height. I want the user to be able to click the up button so they can choose 1 metre again.

Below are the boundaries that the mouse cursor has to be in for the up arrow to be pressed. If so, the mov.chooseheight attribute will increase by 1 which will return the user to the original height.

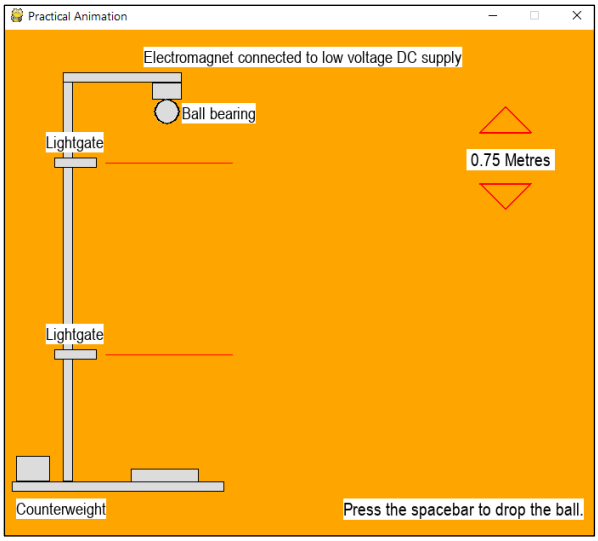
```
624             if pos[0] > 560 and pos[0]< 620  and pos[1] > 90 and pos[1]< 120: #--Boundaries for the up button
625               event = pygame.event.poll()
626               #--The mouse button has to also be pressed
627               if event.type == pygame.MOUSEBUTTONDOWN:
628                 mov.chooseheight += 1 #--Will increment this attribute by 1
```

Below, is the code for the screen for a height of 1 metre when the up button is pressed.

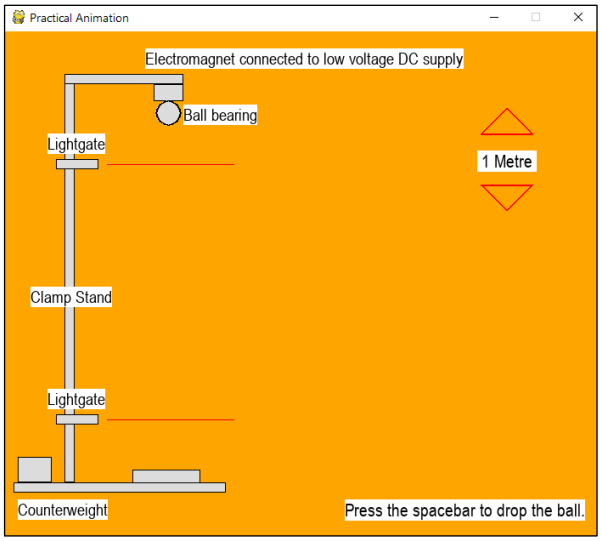
```
625             #--Displays window with new objects
626             win.fill(pract.ORANGE)
627             stand = pygame.draw.rect(win, pract.GREY,(mov.x_stand, mov.y_stand, mov.width_stand, mov.height_stand))
628             #--An identical rectangle is created for the border
629             stand = pygame.draw.rect(win, pract.BLACK,(mov.x_stand, mov.y_stand, mov.width_stand, mov.height_stand), 1)
630             #--Rectangles for the base of the stand
631             base = pygame.draw.rect(win, pract.GREY,(mov.x_base, mov.y_base, mov.width_base, mov.height_base))
632             base = pygame.draw.rect(win, pract.BLACK,(mov.x_base, mov.y_base, mov.width_base, mov.height_base), 1)
633             #--Rectangles for the top of the screen
634             clamp = pygame.draw.rect(win, pract.GREY,(mov.x_clamp, mov.y_clamp, mov.width_clamp, mov.height_clamp))
635             clamp = pygame.draw.rect(win, pract.BLACK,(mov.x_clamp, mov.y_clamp, mov.width_clamp, mov.height_clamp), 1)
636             #--Rectangles for the pad and counterweight
637             pad = pygame.draw.rect(win, pract.GREY,(mov.x_pad, mov.y_pad, mov.width_pad, mov.height_pad))
638             pad = pygame.draw.rect(win, pract.BLACK,(mov.x_pad, mov.y_pad, mov.width_pad, mov.height_pad), 1)
639             counterweight = pygame.draw.rect(win, pract.GREY,(mov.x_counterweight, mov.y_counterweight, mov.width_counterweight, mov.height_counterweight))
640             counterweight = pygame.draw.rect(win, pract.BLACK,(mov.x_counterweight, mov.y_counterweight, mov.width_counterweight, mov.height_counterweight), 1)
641             #--Drawing light gates as rectangles
642             lightgate = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate))
643             lightgate = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate), 1)
644             lightgate2 = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate2, mov.y_lightgate2, mov.width_lightgate2, mov.height_lightgate2))
645             lightgate2 = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate2, mov.y_lightgate2, mov.width_lightgate2, mov.height_lightgate2),1)
646             #--Treating these as rectangles but with very small heights
647             lightray = pygame.draw.rect(win, pract.RED,(mov.x_lightray, mov.y_lightray, mov.width_lightray, mov.height_lightray))
648             lightray2 = pygame.draw.rect(win, pract.RED,(mov.x_lightray2, mov.y_lightray2, mov.width_lightray2, mov.height_lightray2))
649             #--Drawing electromagnets as rectangles
650             electromagnet = pygame.draw.rect(win, pract.GREY,(mov.x_electromagnet, mov.y_electromagnet, mov.width_electromagnet,mov.height_electromagnet))
651             electromagnet = pygame.draw.rect(win, pract.BLACK,(mov.x_electromagnet, mov.y_electromagnet, mov.width_electromagnet,mov.height_electromagnet), 1)
652
653             #--Drawing to the screen the various labels at the specified x and y coordinates
654             win.blit(mov.clampStand_label, (mov.x_clampStandlabel, mov.y_clampStandlabel))
655             win.blit(mov.counterweight_label, (mov.x_counterweightlabel, mov.y_counterweightlabel))
656             win.blit(mov.lightgates_label, (mov.x_lightgateslabel, mov.y_lightgateslabel))
657             win.blit(mov.lightgates_label, (mov.x_lightgateslabel2, mov.y_lightgateslabel2))
658             win.blit(mov.magnet_label, (mov.x_magnetlabel, mov.y_magnetlabel))
659             win.blit(mov.bearings_label, (mov.x_bearingslabel, mov.y_bearingslabel))
660             win.blit(mov.press_spacebar_label, (mov.x_press_spacebarlabel, mov.y_press_spacebarlabel))
661
662             #--Will be drawn as a circle
663             ball_bearing = pygame.draw.circle(win, pract.GREY,(mov.x_ball, mov.y_ball), mov.radius_ball)
664             ball_bearing = pygame.draw.circle(win, pract.BLACK,(mov.x_ball, mov.y_ball), mov.radius_ball,1)
665
666             #--Buttons the user can press to change the position of the light gates
667             increaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_upbutton,2)
668             decreaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_downbutton,2)
669             height_1m_text = pract.fontObj_mediumsmall.render(' 1 Metre ', True, pract.BLACK, pract.WHITE)
670             win.blit(height_1m_text, (mov.x_height_1m_text, mov.y_height_1m_text))
671
```

Development Testing:

Slider	Valid input – slider is moved Invalid input – slider is moved beyond its limits	Allows the user to change an aspect of the practical such as height	Allows the user to interact with the practical	4.2 Switching between heights



Height of 0.75 metres



Height of 1 metre

```
0.75m metre
1 metre
```

Additionally, I have printed to the Python shell when each height was chosen. Moreover, I have tested my program for robustness as I repeatedly clicked the buttons to change the height.

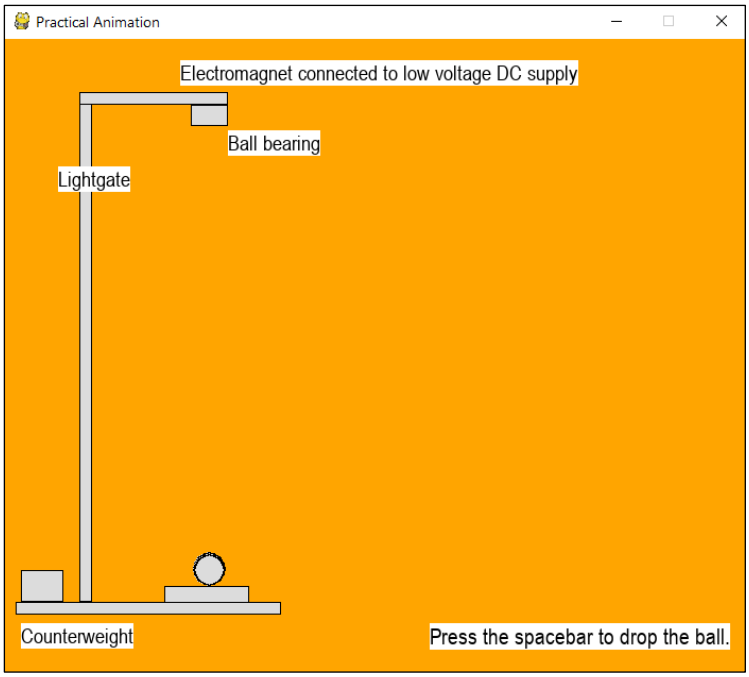
```
0.75m metre
1 metre
0.75m metre
1 metre
0.75m metre
1 metre
0.75m metre
1 metre
0.75m metre
1 metre
0.75m metre
1 metre
0.75m metre
1 metre
0.75m metre
1 metre
0.75m metre
1 metre
0.75m metre
1 metre
0.75m metre
1 metre
0.75m metre
1 metre
0.75m metre
1 metre
0.75m metre
1 metre
0.75m metre
1 metre
0.75m metre
1 metre
0.75m metre
1 metre
0.75m metre
1 metre
```

I encountered no errors so this test was a success.

Validation:

Result	Valid input – Clicking the down button or up button once	Invalid input – Clicking anything that isn't the down or up button	Boundary input – Repeatedly clicking the down or up buttons
Expected output?	Yes, the screen changed and the user could drop the ball	Yes, the screen did not change	No, while the screen changed, dropping the ball was not successful

While my valid and invalid inputs were as expected, my boundary input was not.



My boundary test is slightly different from my robustness testing. For robustness, I am clicking the buttons in an alternating order – I don't press the same button twice in a row. While for validation, I am clicking the same button many times in a row.

When doing this, the screen changed correctly. However, when dropping the ball, the diagram wasn't displayed properly. I think this is because whenever the button is pressed, the 'mov.chooseheight' attribute increments or decrements. For a height of 1 metre to be displayed, this attribute must equal 0 but if the button is pressed more than once, the attribute will be greater than 0.

Solution:

The code below protects against repeated clicking of the up button as the max height that the user can choose is 1 metre. If the program recognises that the attribute is greater than 0, it will be reset to 0 so the maximum height cannot be exceeded.

```
619             if pos[0] > 560 and pos[0]< 620  and pos[1] > 90 and pos[1]< 120: #--Boundaries for the up button
620               event = pygame.event.poll()
621               #--The mouse button has to also be pressed
622               if event.type == pygame.MOUSEBUTTONDOWN:
623                 mov.chooseheight += 1 #--Will increment this attribute by 1
624                 if mov.chooseheight > 0:
625                   mov.chooseheight = 0 #--Protects against repeated clicking of the up button
```

I will do the same for the down button once I have confirmed how many heights I want to include.

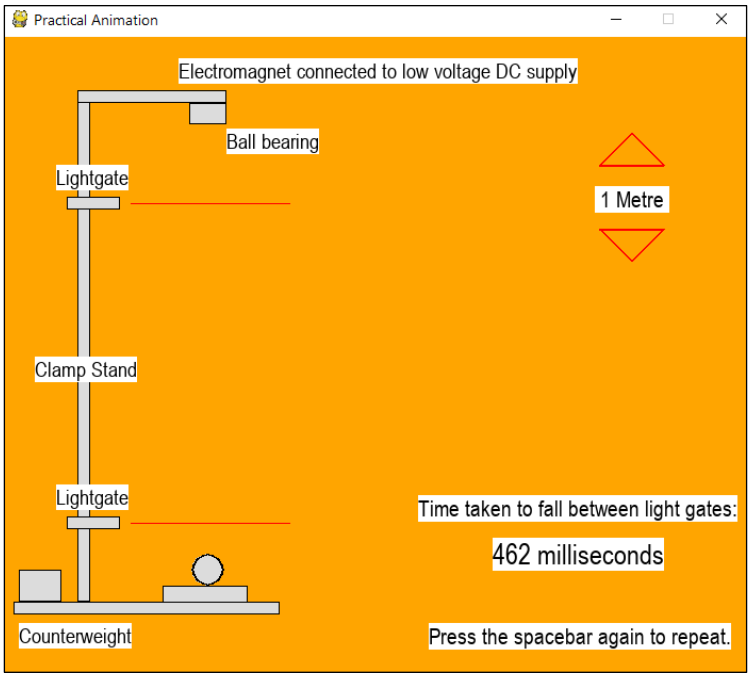
Changing heights when the ball has fallen

I want there to be a way for the user to change the height once the ball has already fallen. Currently, this can only be done at the start of the animation.

Therefore, after the ball has fallen, I have displayed the buttons and current height.

```
            #--So the user can change the heights again, once the ball had fallen
            increaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_upbutton,2)
            decreaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_downbutton,2)
            #--Displays the current height between the light gates
            if mov.chooseheight == 0:
              height_1m_text = pract.fontObj_mediumsmall.render(' 1 Metre ', True, pract.BLACK, pract.WHITE)
              win.blit(height_1m_text, (mov.x_height_1m_text, mov.y_height_1m_text))
            elif mov.chooseheight == -1:
              height_075m_text = pract.fontObj_mediumsmall.render(' 0.75 Metres ', True, pract.BLACK, pract.WHITE)
              win.blit(height_075m_text, (mov.x_height_075m_text, mov.y_height_075m_text))
```

When the ball has fallen:



After this, I will implement another while loop to allow the user to click the arrow buttons to change the height.

```
            #--Detects whether the up or down button has been pressed
            while pract.condition_lightgatebutton2 == True:
              keys = pygame.key.get_pressed() #--Receives any key press
              pos = pygame.mouse.get_pos() #--Receives any mouse press
              #--Boundaries the user's mouse position has to be in for the bottom button to be pressed
              if pos[0] > 560 and pos[0]< 620  and pos[1] > 180 and pos[1]< 210:
                event = pygame.event.poll()
                #--The mouse button has to also be pressed
                if event.type == pygame.MOUSEBUTTONDOWN:
                  mov.chooseheight -=1

                  #--Displaying the buttons and text again
                  increaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_upbutton,2)
                  decreaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_downbutton,2)
                  height_075m_text = pract.fontObj_mediumsmall.render(' 0.75 Metres ', True, pract.BLACK, pract.WHITE)
                  win.blit(height_075m_text, (mov.x_height_075m_text, mov.y_height_075m_text))

              if pos[0] > 560 and pos[0]< 620  and pos[1] > 90 and pos[1]< 120: #--Boundaries for the up button
                event = pygame.event.poll()
                #--The mouse button has to also be pressed
                if event.type == pygame.MOUSEBUTTONDOWN:
                  mov.chooseheight += 1 #--Will increment this attribute by 1
                  if mov.chooseheight > 0:
                    mov.chooseheight = 0 #--Protects against repeated clicking of the up button
                  else:

                    #--Displaying the buttons and text again
                    increaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_upbutton,2)
                    decreaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_downbutton,2)
                    height_1m_text = pract.fontObj_mediumsmall.render(' 1 Metre ', True, pract.BLACK, pract.WHITE)
                    win.blit(height_1m_text, (mov.x_height_1m_text, mov.y_height_1m_text))

              if keys[pygame.K_SPACE]: #--Checks whether the space bar has been pressed
                pract.condition_lightgatebutton2 = False #--If so, this while loop will be ended
                mov.acceleration = 0 #--Resets acceleration and position of the ball
                mov.y_ball = 96

              #--While the program is running, checks whether the user is quitting the window
              for event in pygame.event.get():
                if event.type == pygame.QUIT:
                  pygame.quit()
                  sys.exit()
              pygame.display.update() #--Updates the display
```

The above while loop is similar to the one I have implemented when the user could first change the heights of the light gates. When the up or down button is pressed, the attribute on line 794 and 806 are updated. This means that when the spacebar is pressed on line 817, the correct screen will be displayed.

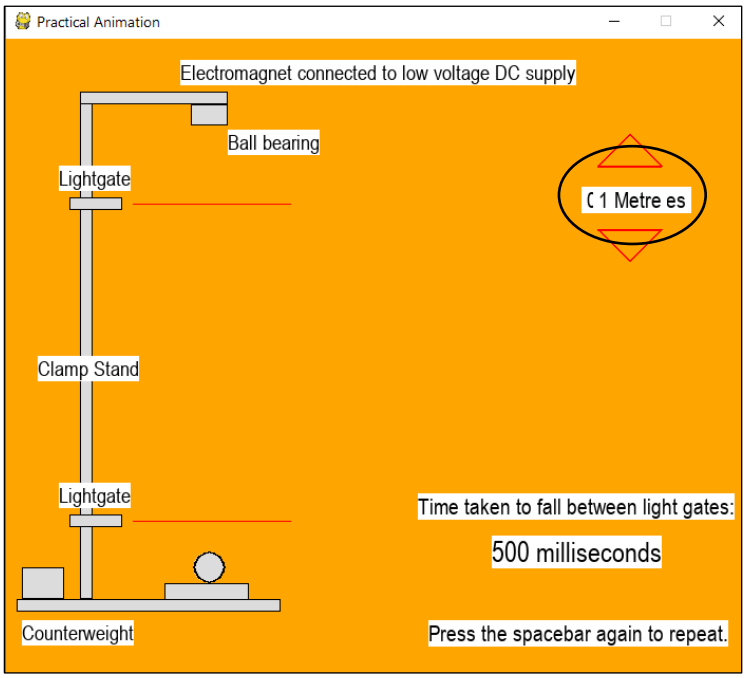
However, the user will only be able to repeat the practical if they have already pressed the up or down button since the code detecting spacebar presses is contained in while loop (lines 817 to 820). Therefore, I will include another if condition outside the while loop. This is shown below from lines 832 to 835. This allows the user to repeat the practical without changing the height.

```
819             if keys[pygame.K_SPACE]: #--Checks whether the space bar has been pressed
820               pract.condition_lightgatebutton2 = False #--If so, this while loop will be ended
821               mov.acceleration = 0 #--Resets acceleration and position of the ball
822               mov.y_ball = 96
823
824             #--While the program is running, checks whether the user is quitting the window
825             for event in pygame.event.get():
826               if event.type == pygame.QUIT:
827                 pygame.quit()
828                 sys.exit()
829             pygame.display.update() #--Updates the display
830
831
832           if keys[pygame.K_SPACE]: #--Checks whether the space bar has been pressed (only when the up or down button has not been pressed)
833             pract.condition_lightgatebutton2 = False #--If so, this while loop will be ended
834             mov.acceleration = 0 #--Resets acceleration and position of the ball
835             mov.y_ball = 96
```

Development Testing:

Slider	Valid input – slider is moved Invalid input – slider is moved beyond its limits	Allows the user to change an aspect of the practical such as height	Allows the user to interact with the practical	4.2 Changing heights once the ball has been dropped

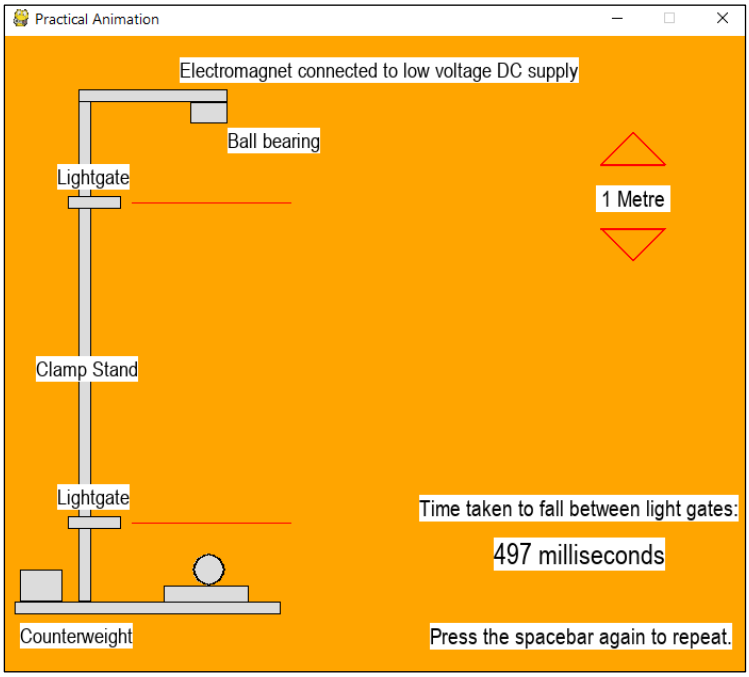
When testing my program, I faced two problems meaning my test was not successful.



When switching between heights, the text was not been displayed properly as shown in the screenshot above. This is due to Pygame displaying new objects on top of old ones. I will have to redisplay all the objects in the window to fix this.

Solution:

```
                win.fill(pract.ORANGE)
                stand = pygame.draw.rect(win, pract.GREY,(mov.x_stand, mov.y_stand, mov.width_stand, mov.height_stand))
                #--An identical rectangle is created for the border
                stand = pygame.draw.rect(win, pract.BLACK,(mov.x_stand, mov.y_stand, mov.width_stand, mov.height_stand), 1)
                #--Rectangles for the base of the stand
                base = pygame.draw.rect(win, pract.GREY,(mov.x_base, mov.y_base, mov.width_base, mov.height_base))
                base = pygame.draw.rect(win, pract.BLACK,(mov.x_base, mov.y_base, mov.width_base, mov.height_base), 1)
                #--Rectangles for the top of the screen
                clamp = pygame.draw.rect(win, pract.GREY,(mov.x_clamp, mov.y_clamp, mov.width_clamp, mov.height_clamp))
                clamp = pygame.draw.rect(win, pract.BLACK,(mov.x_clamp, mov.y_clamp, mov.width_clamp, mov.height_clamp), 1)
                #--Rectangles for the pad and counterweight
                pad = pygame.draw.rect(win, pract.GREY,(mov.x_pad, mov.y_pad, mov.width_pad, mov.height_pad))
                pad = pygame.draw.rect(win, pract.BLACK,(mov.x_pad, mov.y_pad, mov.width_pad, mov.height_pad), 1)
                counterweight = pygame.draw.rect(win, pract.GREY,(mov.x_counterweight, mov.y_counterweight, mov.width_counterweight, mov.height_counterweight))
                counterweight = pygame.draw.rect(win, pract.BLACK,(mov.x_counterweight, mov.y_counterweight, mov.width_counterweight, mov.height_counterweight), 1)
                #--Drawing light gates as rectangles
                lightgate = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate))
                lightgate = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate), 1)
                lightgate2 = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate2, mov.y_lightgate2, mov.width_lightgate2, mov.height_lightgate2))
                lightgate2 = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate2, mov.y_lightgate2, mov.width_lightgate2, mov.height_lightgate2),1)
                #--Treating these as rectangles but with very small heights
                lightray = pygame.draw.rect(win, pract.RED,(mov.x_lightray, mov.y_lightray, mov.width_lightray, mov.height_lightray))
                lightray2 = pygame.draw.rect(win, pract.RED,(mov.x_lightray2, mov.y_lightray2, mov.width_lightray2, mov.height_lightray2))
                #--Drawing electromagnets as rectangles
                electromagnet = pygame.draw.rect(win, pract.GREY,(mov.x_electromagnet, mov.y_electromagnet, mov.width_electromagnet,mov.height_electromagnet))
                electromagnet = pygame.draw.rect(win, pract.BLACK,(mov.x_electromagnet, mov.y_electromagnet, mov.width_electromagnet,mov.height_electromagnet), 1)
                #--Drawing to the screen the various labels at the specified x and y coordinates
                win.blit(mov.clampStand_label, (mov.x_clampStandlabel, mov.y_clampStandlabel))
                win.blit(mov.counterweight_label, (mov.x_counterweightlabel, mov.y_counterweightlabel))
                win.blit(mov.lightgates_label, (mov.x_lightgateslabel, mov.y_lightgateslabel))
                win.blit(mov.lightgates_label, (mov.x_lightgateslabel2, mov.y_lightgateslabel2))
                win.blit(mov.magnet_label, (mov.x_magnetlabel, mov.y_magnetlabel))
                win.blit(mov.bearings_label, (mov.x_bearingslabel, mov.y_bearingslabel))
                win.blit(mov.press_spacebar_label, (mov.x_press_spacebarlabel, mov.y_press_spacebarlabel))

                win.blit(self.display_timer, (mov.x_displaytimer,mov.y_displaytimer))
                win.blit(self.time_result, (mov.x_timerresult,mov.y_timerresult))
                #--Prompts the user to repeat the practical
                win.blit(mov.repeat_spacebar_label, (mov.x_press_spacebarlabel, mov.y_press_spacebarlabel))
                #--Draws the ball bearing onscreen again
                ball_bearing = pygame.draw.circle(win, pract.GREY,(mov.x_ball, int(mov.y_ball)), mov.radius_ball)
                ball_bearing = pygame.draw.circle(win, pract.BLACK,(mov.x_ball, int(mov.y_ball)), mov.radius_ball,1)

                #--Displaying the buttons and text again
                increaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_upbutton,2)
                decreaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_downbutton,2)
                height_1m_text = pract.fontObj_mediumsmall.render(' 1 Metre ', True, pract.BLACK, pract.WHITE)
                win.blit(height_1m_text, (mov.x_height_1m_text, mov.y_height_1m_text))
```



The second problem I faced was that I could not change the heights once the ball had dropped after I had repeated the animation. The cause of the problem was that when the user presses the spacebar to repeat the practical, I have set the while loop condition to false. This means when the program is repeating, that condition is now false, so the loop will not run. I have solved this by setting this condition to true just before the while loop begins so that it will always run.

```
785             #--Sets this condition to true so the while loop can always run
786             pract.condition_lightgatebutton2 = True
787             #--Detects whether the up or down button has been pressed
788             while pract.condition_lightgatebutton2 == True:
```

After fixing these issues, my test was successful.

Adding new heights

Before I begin to add these heights, I discovered that I calculated the distance between the light gates incorrectly. For the height of 0.75m, to position the bottom light gate, I multiplied its y-coordinate by 0.75. However, here, I am assuming that the upper light gate is at a y-coordinate of 0. Since this y-coordinate is at 150, I will need to multiply the height between the light gates by 0.75 instead. Due to my implementation of classes, this should be an easy fix since I will only have to change this value once.

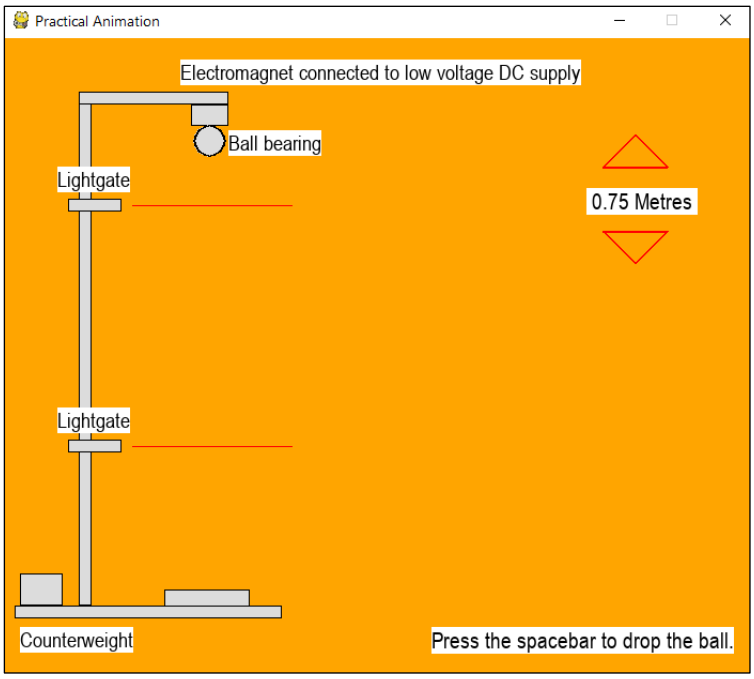
These are the new coordinates:

```
329     self.y_lightgate2_075m = 375
330     self.y_lightray2_075m = 381
```

I will also change the coordinates of the light gate label so that appears above the light gate.

```
389     self.y_lightgateslabel2_075m = 345
```

The screen looks like this:



Now, I will add more heights that the user can experiment with. I will start with heights of 0.5 and 0.25m so the user has a range that they can experiment with. Ideally, there should be smaller increments of 0.2m or 0.1m however due to time constraints I will have a total of 4 heights.

These are the new coordinates for the heights:

```
332         self.y_lightgate2_05m = 300
333         self.y_lightray2_05m = 306
334
335         self.y_lightgate2_025m = 225
336         self.y_lightray2_025m = 231
```

These are the coordinates for the new labels:

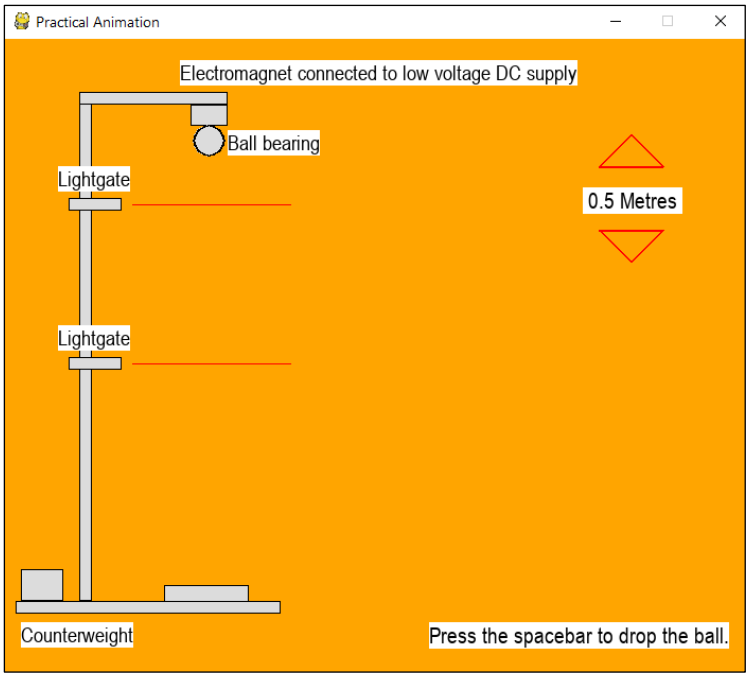
```
397         self.y_lightgateslabel2_050m = 270
398         self.y_lightgateslabel2_025m = 190
```

0.5 metres:

When the down button is pressed twice, the mov.chooseheight attribute will reduce in value by 2. I have introduced an if statement which when met, will display a new screen. On lines 669,670,674,680 and 681 are the new objects that will be displayed.

```
634             #--If the button is pressed twice, the diagram for a height of 0.50m will be displayed.
635             if mov.chooseheight == -2:
636               win.fill(pract.ORANGE)
637               stand = pygame.draw.rect(win, pract.GREY,(mov.x_stand, mov.y_stand, mov.width_stand, mov.height_stand))
638               #--An identical rectangle is created for the border
639               stand = pygame.draw.rect(win, pract.BLACK,(mov.x_stand, mov.y_stand, mov.width_stand, mov.height_stand), 1)
640               #--Rectangles for the base of the stand
641               base = pygame.draw.rect(win, pract.GREY,(mov.x_base, mov.y_base, mov.width_base, mov.height_base))
642               base = pygame.draw.rect(win, pract.BLACK,(mov.x_base, mov.y_base, mov.width_base, mov.height_base), 1)
643               #--Rectangles for the top of the screen
644               clamp = pygame.draw.rect(win, pract.GREY,(mov.x_clamp, mov.y_clamp, mov.width_clamp, mov.height_clamp))
645               clamp = pygame.draw.rect(win, pract.BLACK,(mov.x_clamp, mov.y_clamp, mov.width_clamp, mov.height_clamp), 1)
646               #--Rectangles for the pad and counterweight
647               pad = pygame.draw.rect(win, pract.GREY,(mov.x_pad, mov.y_pad, mov.width_pad, mov.height_pad))
648               pad = pygame.draw.rect(win, pract.BLACK,(mov.x_pad, mov.y_pad, mov.width_pad, mov.height_pad), 1)
649               counterweight = pygame.draw.rect(win, pract.GREY,(mov.x_counterweight, mov.y_counterweight, mov.width_counterweight, mov.height_counterweight))
650               counterweight = pygame.draw.rect(win, pract.BLACK,(mov.x_counterweight, mov.y_counterweight, mov.width_counterweight, mov.height_counterweight), 1)
651               #--Drawing electromagnets as rectangles
652               electromagnet = pygame.draw.rect(win, pract.GREY,(mov.x_electromagnet, mov.y_electromagnet, mov.width_electromagnet,mov.height_electromagnet))
653               electromagnet = pygame.draw.rect(win, pract.BLACK,(mov.x_electromagnet, mov.y_electromagnet, mov.width_electromagnet,mov.height_electromagnet), 1)
654               #--Will be drawn as a circle
655               ball_bearing = pygame.draw.circle(win, pract.GREY,(mov.x_ball, mov.y_ball), mov.radius_ball)
656               ball_bearing = pygame.draw.circle(win, pract.BLACK,(mov.x_ball, mov.y_ball), mov.radius_ball,1)
657               #--Drawing to the screen the various labels at the specified x and y coordinates
658               win.blit(mov.counterweight_label, (mov.x_counterweightlabel, mov.y_counterweightlabel))
659               win.blit(mov.lightgates_label, (mov.x_lightgateslabel, mov.y_lightgateslabel))
660               #--New position of the light gate label
661               win.blit(mov.lightgates_label, (mov.x_lightgateslabel2, mov.y_lightgateslabel2_050m))
662               win.blit(mov.magnet_label, (mov.x_magnetlabel, mov.y_magnetlabel))
663               win.blit(mov.bearings_label, (mov.x_bearingslabel, mov.y_bearingslabel))
664               win.blit(mov.press_spacebar_label, (mov.x_press_spacebarlabel, mov.y_press_spacebarlabel))
665
666               lightgate = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate))
667               lightgate = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate), 1)
668               #--New coordinates for the light gates
669               lightgate2_050m = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate2, mov.y_lightgate2_050m, mov.width_lightgate2, mov.height_lightgate2))
670               lightgate2_050m= pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate2, mov.y_lightgate2_050m, mov.width_lightgate2, mov.height_lightgate2), 1)
671
672               #--Treating these as rectangles but with very small heights
673               lightray = pygame.draw.rect(win, pract.RED,(mov.x_lightray, mov.y_lightray, mov.width_lightray, mov.height_lightray))
674               lightray2_050m = pygame.draw.rect(win, pract.RED,(mov.x_lightray2, mov.y_lightray2_050m, mov.width_lightray2, mov.height_lightray2))
675
676               #--Buttons the user can press to change the position of the light gates
677               increaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_upbutton,2)
678               decreaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_downbutton,2)
679               #--New label for the height of 0.5 metres
680               height_050m_text = pract.fontObj_mediumsmall.render(' 0.5 Metres ', True, pract.BLACK, pract.WHITE)
681               win.blit(height_050m_text, (mov.x_height_050m_text, mov.y_height_050m_text))
```

When the down button is pressed twice, this is the resulting screen:



When clicking on the buttons, I had difficulty in pressing them. To increase usability, I decided to make the area where the click registers larger.

For the ball to drop, I have added another if statement which when met will redraw the screen for the height of 0.5m.

```
795         if mov.chooseheight == -2: #--When the down button is pressed twice
796           #--Drawing the new light gates corresponding to a height of 0.5 metres
797           lightgate = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate))
798           lightgate = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate), 1)
799           lightgate2_050m = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate2, mov.y_lightgate2_050m, mov.width_lightgate2, mov.height_lightgate2))
800           lightgate2_050m = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate2, mov.y_lightgate2_050m, mov.width_lightgate2, mov.height_lightgate2), 1)
801           #--Treating these as rectangles but with very small heights
802           lightray = pygame.draw.rect(win, pract.RED,(mov.x_lightray, mov.y_lightray, mov.width_lightray, mov.height_lightray))
803           lightray2_050m = pygame.draw.rect(win, pract.RED,(mov.x_lightray2, mov.y_lightray2_050m, mov.width_lightray2, mov.height_lightray2))
804           #--Drawing light gate label
805           win.blit(mov.lightgates_label, (mov.x_lightgateslabel2, mov.y_lightgateslabel2_050m))
```

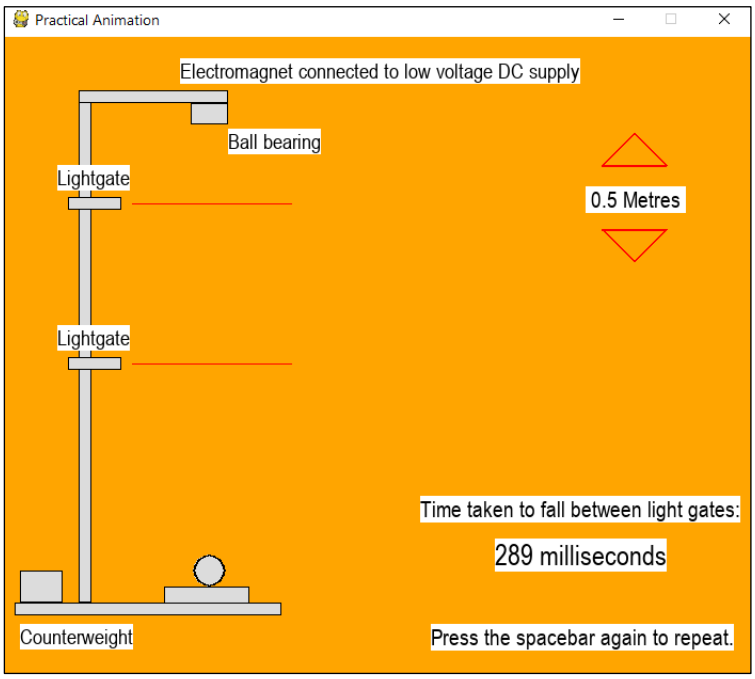
To detect the time it takes for the ball to fall, I have added an if statement which when met will record time between the light gates for the height of 0.5m.

```
843         if mov.chooseheight == -2: #--If the light gates are separated by a distance of 0.5 metres
844           if mov.y_ball > mov.y_lightray - 1 and mov.y_ball < mov.y_lightray + 1: #--When ball passes the first light gate
845             self.lightgate_detect = pygame.time.get_ticks() #--Will record time
846
847           if mov.y_ball > mov.y_lightray2_050m - 1 and mov.y_ball < mov.y_lightray2_050m + 1: #--When ball passes the second light gate
848             self.lightgate2_detect = pygame.time.get_ticks()
849             self.result_time = self.lightgate2_detect - self.lightgate_detect #--Stores the calculated time
850             print(self.result_time) #--Will print out the time in the shell for confirmation
```

Below is the code which allows the user to change the height once the ball has dropped. On line 895, if the mov.chooseheight attribute is equal to -2 then this new height will be displayed and the ball can be dropped again.

```
872            elif mov.chooseheight == -2:
873              height_050m_text = pract.fontObj_mediumsmall.render(' 0.5 Metres ', True, pract.BLACK, pract.WHITE)
874              win.blit(height_050m_text, (mov.x_height_050m_text, mov.y_height_050m_text))
875
876            #--Sets this condition to true so the while loop can always run
877            pract.condition_lightgatebutton2 = True
878            #--Detects whether the up or down button has been pressed
879            while pract.condition_lightgatebutton2 == True:
880              keys = pygame.key.get_pressed() #--Receives any key press
881              pos = pygame.mouse.get_pos() #--Receives any mouse press
882              #--Boundaries the user's mouse position has to be in for the bottom button to be pressed
883              if pos[0] > 560 and pos[0]< 620  and pos[1] > 180 and pos[1]< 210:
884                event = pygame.event.poll()
885                #--The mouse button has to also be pressed
886                if event.type == pygame.MOUSEBUTTONDOWN:
887                  mov.chooseheight -=1
888                  if mov.chooseheight == -1:
889                    #--Displaying the buttons and text again
890                    increaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_upbutton,2)
891                    decreaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_downbutton,2)
892                    height_075m_text = pract.fontObj_mediumsmall.render(' 0.75 Metres ', True, pract.BLACK, pract.WHITE)
893                    win.blit(height_075m_text, (mov.x_height_075m_text, mov.y_height_075m_text))
894
895                  if mov.chooseheight == -2:
896                    #--Displaying the buttons and text again
897                    increaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_upbutton,2)
898                    decreaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_downbutton,2)
899                    height_050m_text = pract.fontObj_mediumsmall.render(' 0.5 Metres   ', True, pract.BLACK, pract.WHITE)
900                    win.blit(height_050m_text, (mov.x_height_050m_text, mov.y_height_050m_text))
```

When running my program, clicking the down button twice and then pressing space gave this result:

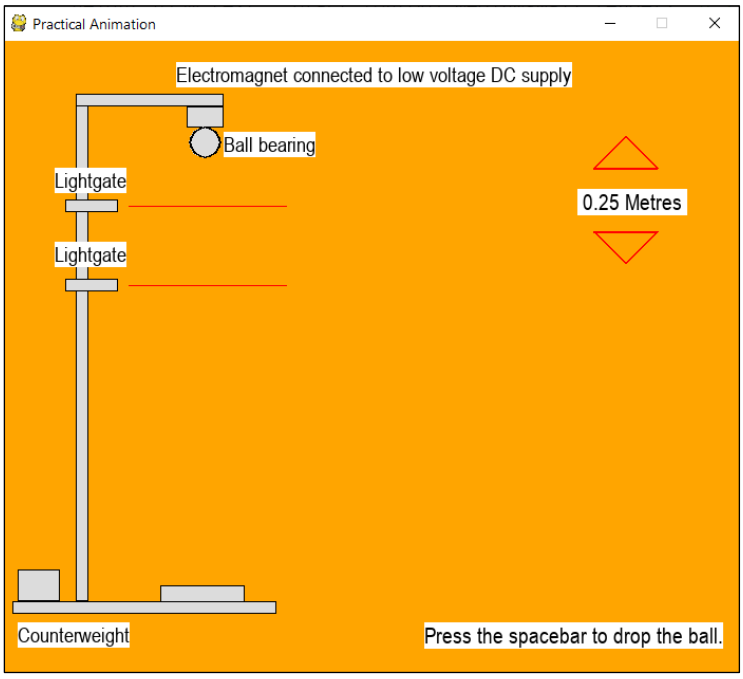


The user can return the height to either 0.75 or 1 metre and repeat the program.

0.25 metres:

When the down arrow button is pressed three times, the light gates will move to display a height of 0.25m. Below is the code for this screen.

```
            #--If the button is pressed thrice, the diagram for a height of 0.25m will be displayed.
            if mov.chooseheight == -3:
              win.fill(pract.ORANGE)
              stand = pygame.draw.rect(win, pract.GREY,(mov.x_stand, mov.y_stand, mov.width_stand, mov.height_stand))
              #--An identical rectangle is created for the border
              stand = pygame.draw.rect(win, pract.BLACK,(mov.x_stand, mov.y_stand, mov.width_stand, mov.height_stand), 1)
              #--Rectangles for the base of the stand
              base = pygame.draw.rect(win, pract.GREY,(mov.x_base, mov.y_base, mov.width_base, mov.height_base))
              base = pygame.draw.rect(win, pract.BLACK,(mov.x_base, mov.y_base, mov.width_base, mov.height_base), 1)
              #--Rectangles for the top of the screen
              clamp = pygame.draw.rect(win, pract.GREY,(mov.x_clamp, mov.y_clamp, mov.width_clamp, mov.height_clamp))
              clamp = pygame.draw.rect(win, pract.BLACK,(mov.x_clamp, mov.y_clamp, mov.width_clamp, mov.height_clamp), 1)
              #--Rectangles for the pad and counterweight
              pad = pygame.draw.rect(win, pract.GREY,(mov.x_pad, mov.y_pad, mov.width_pad, mov.height_pad))
              pad = pygame.draw.rect(win, pract.BLACK,(mov.x_pad, mov.y_pad, mov.width_pad, mov.height_pad), 1)
              counterweight = pygame.draw.rect(win, pract.GREY,(mov.x_counterweight, mov.y_counterweight, mov.width_counterweight, mov.height_counterweight))
              counterweight = pygame.draw.rect(win, pract.BLACK,(mov.x_counterweight, mov.y_counterweight, mov.width_counterweight, mov.height_counterweight), 1)
              #--Drawing electromagnets as rectangles
              electromagnet = pygame.draw.rect(win, pract.GREY,(mov.x_electromagnet, mov.y_electromagnet, mov.width_electromagnet,mov.height_electromagnet))
              electromagnet = pygame.draw.rect(win, pract.BLACK,(mov.x_electromagnet, mov.y_electromagnet, mov.width_electromagnet,mov.height_electromagnet), 1)
              #--Will be drawn as a circle
              ball_bearing = pygame.draw.circle(win, pract.GREY,(mov.x_ball, mov.y_ball), mov.radius_ball)
              ball_bearing = pygame.draw.circle(win, pract.BLACK,(mov.x_ball, mov.y_ball), mov.radius_ball,1)
              #--Drawing to the screen the various labels at the specified x and y coordinates
              win.blit(mov.counterweight_label, (mov.x_counterweightlabel, mov.y_counterweightlabel))
              win.blit(mov.lightgates_label, (mov.x_lightgateslabel, mov.y_lightgateslabel))
              #--New position of the light gate label
              win.blit(mov.lightgates_label, (mov.x_lightgateslabel2, mov.y_lightgateslabel2_025m))
              win.blit(mov.magnet_label, (mov.x_magnetlabel, mov.y_magnetlabel))
              win.blit(mov.bearings_label, (mov.x_bearingslabel, mov.y_bearingslabel))
              win.blit(mov.press_spacebar_label, (mov.x_press_spacebarlabel, mov.y_press_spacebarlabel))

              lightgate = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate))
              lightgate = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate), 1)
              #--New coordinates for the light gate
              lightgate2_025m = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate2, mov.y_lightgate2_025m, mov.width_lightgate2, mov.height_lightgate2))
              lightgate2_025m= pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate2, mov.y_lightgate2_025m, mov.width_lightgate2, mov.height_lightgate2), 1)
              #--Treating these as rectangles but with very small heights
              lightray = pygame.draw.rect(win, pract.RED,(mov.x_lightray, mov.y_lightray, mov.width_lightray, mov.height_lightray))
              lightray2_025m = pygame.draw.rect(win, pract.RED,(mov.x_lightray2, mov.y_lightray2_025m, mov.width_lightray2, mov.height_lightray2))

              #--Buttons the user can press to change the position of the light gates
              increaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_upbutton,2)
              decreaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_downbutton,2)
              #--New label for the height of 0.25 metres
              height_025m_text = pract.fontObj_mediumsmall.render(' 0.25 Metres ', True, pract.BLACK, pract.WHITE)
              win.blit(height_025m_text, (mov.x_height_025m_text, mov.y_height_025m_text))
```



To prevent repeated clicking of the down button, if the mov.chooseheight attribute has a value of less than -3, I have set it equal to -3 so the lowest height can be displayed.

```
587                 #--Protects against repeated clicking of the button
588                 if mov.chooseheight < -3:
589                   mov.chooseheight = -3
```

This displays the correct screen when the ball is falling:

```
858         if mov.chooseheight == -3: #--When the down button is pressed thrice
859           #--Drawing the new light gates corresponding to a height of 0.25 metres
860           lightgate = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate))
861           lightgate = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate), 1)
862           lightgate2_025m = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate2, mov.y_lightgate2_025m, mov.width_lightgate2, mov.height_lightgate2))
863           lightgate2_025m = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate2, mov.y_lightgate2_025m, mov.width_lightgate2, mov.height_lightgate2), 1)
864           #--Treating these as rectangles but with very small heights
865           lightray = pygame.draw.rect(win, pract.RED,(mov.x_lightray, mov.y_lightray, mov.width_lightray, mov.height_lightray))
866           lightray2_025m = pygame.draw.rect(win, pract.RED,(mov.x_lightray2, mov.y_lightray2_025m, mov.width_lightray2, mov.height_lightray2))
867           #--Drawing light gate label
868           win.blit(mov.lightgates_label, (mov.x_lightgateslabel2, mov.y_lightgateslabel2_025m))
```

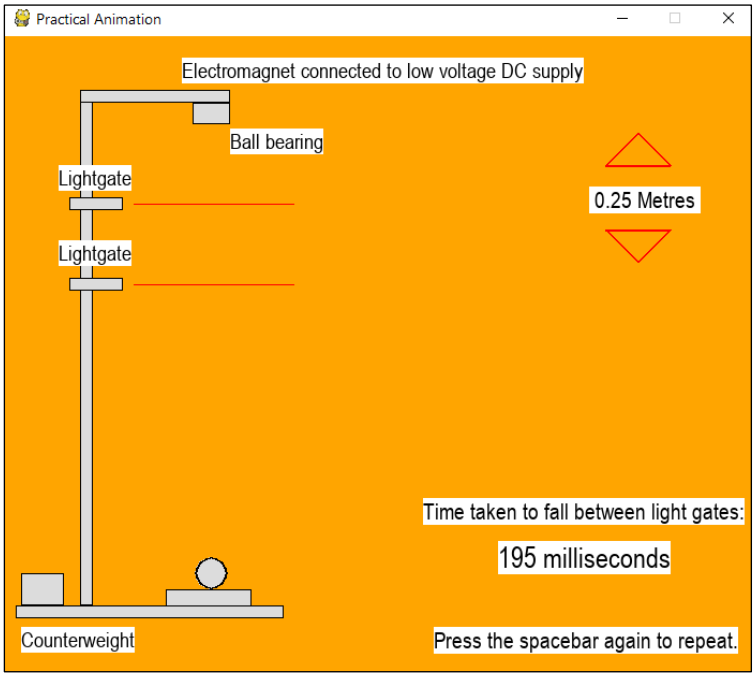
This calculates the correct time between the light gates.

```
        if mov.chooseheight == -3: #--If the light gates are separated by a distance of 0.25 metres
          if mov.y_ball > mov.y_lightray - 1 and mov.y_ball < mov.y_lightray + 1: #--When ball passes the first light gate
            self.lightgate_detect = pygame.time.get_ticks() #--Will record time

          if mov.y_ball > mov.y_lightray2_025m - 1 and mov.y_ball < mov.y_lightray2_025m + 1: #--When ball passes the second light gate
            self.lightgate2_detect = pygame.time.get_ticks()
            self.result_time = self.lightgate2_detect - self.lightgate_detect #--Stores the calculated time
            print(self.result_time) #--Will print out the time in the shell for confirmation
```

This is so the heights can be changed by the user after the ball has completed its fall.

```
          elif mov.chooseheight == -3:
            height_050m_text = pract.fontObj_mediumsmall.render(' 0.25 Metres ', True, pract.BLACK, pract.WHITE)
            win.blit(height_050m_text, (mov.x_height_025m_text, mov.y_height_025m_text))



          #--Sets this condition to true so the while loop can always run
          pract.condition_lightgatebutton2 = True
          #--Detects whether the up or down button has been pressed
          while pract.condition_lightgatebutton2 == True:
            keys = pygame.key.get_pressed() #--Receives any key press
            pos = pygame.mouse.get_pos() #--Receives any mouse press
            #--Boundaries the user's mouse position has to be in for the bottom button to be pressed
            if pos[0] > 560 and pos[0]< 620  and pos[1] > 160 and pos[1]< 230:
              event = pygame.event.poll()
              #--The mouse button has to also be pressed
              if event.type == pygame.MOUSEBUTTONDOWN:
                mov.chooseheight -=1
                if mov.chooseheight < -3:
                  mov.chooseheight = -3
                if mov.chooseheight == -1:
                  #--Displaying the buttons and text again
                  increaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_upbutton,2)
                  decreaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_downbutton,2)
                  height_075m_text = pract.fontObj_mediumsmall.render(' 0.75 Metres ', True, pract.BLACK, pract.WHITE)
                  win.blit(height_075m_text, (mov.x_height_075m_text, mov.y_height_075m_text))

                if mov.chooseheight == -2:
                  #--Displaying the buttons and text again
                  increaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_upbutton,2)
                  decreaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_downbutton,2)
                  height_050m_text = pract.fontObj_mediumsmall.render(' 0.5 Metres   ', True, pract.BLACK, pract.WHITE)
                  win.blit(height_050m_text, (mov.x_height_050m_text, mov.y_height_050m_text))

                if mov.chooseheight == -3:
                  #--Displaying the buttons and text again
                  increaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_upbutton,2)
                  decreaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_downbutton,2)
                  height_050m_text = pract.fontObj_mediumsmall.render(' 0.25 Metres   ', True, pract.BLACK, pract.WHITE)
                  win.blit(height_025m_text, (mov.x_height_025m_text, mov.y_height_025m_text))
```

Below is the resulting screen for the ball when it has fallen

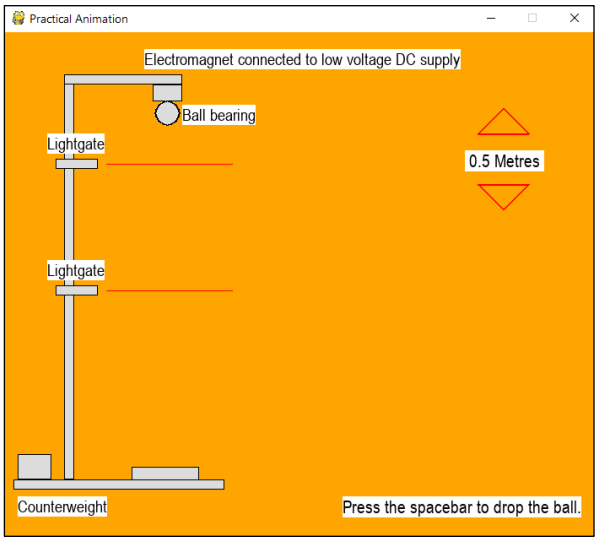


Development Testing:

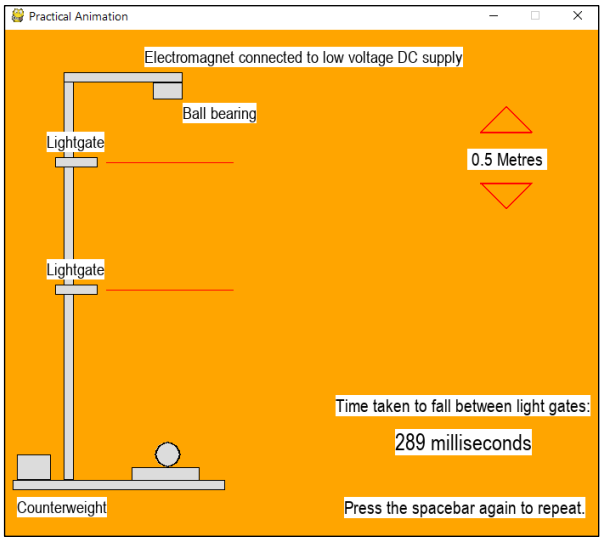
Slider	Valid input – slider is moved Invalid input – slider is moved beyond its limits	Allows the user to change an aspect of the practical such as height	Allows the user to interact with the practical	4.2 Displaying heights of 0.5 and 0.25 metres

I will test whether the program displays the correct screens for the heights and whether the ball can be dropped for these heights.

0.5m

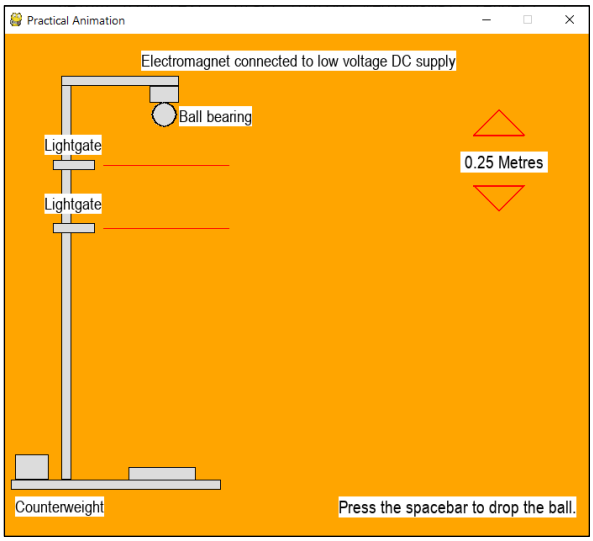


When the down arrow is pressed

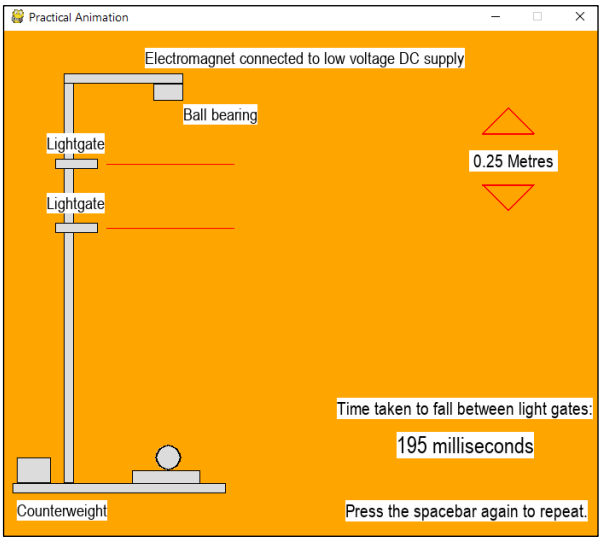


When the spacebar is pressed

0.25m



When the down arrow is pressed



When the spacebar is pressed

I have repeated these tests several times and all of them resulted in no errors. This means that this test was successful.

CREATING A GRAPH OF RESULTS:

The final stage in my practical animation is for the user to view a graph of the recorded times which will be plotted against each corresponding height. To do this, I will create a scatter diagram using Matplotlib and then will draw a line of best fit.

Firstly, I need a way for the program to record the results of the practical. I have chosen lists to do this.

```
434     #--Recording times and heights
435     self.graphTime = []
436     self.graphHeight = []
```

Next, I will import Matplotlib:

```
7 import matplotlib.pyplot as plt
```

Whenever the ball falls, the time taken to fall and corresponding height will be appended to these lists. This is shown on lines 918, 930, 942 and 954.

```
909         if mov.chooseheight == 0: #--If the light gates are in their original positions
910           if mov.y_ball > mov.y_lightray - 1 and mov.y_ball < mov.y_lightray + 1: #--When ball passes the first light gate
911             self.lightgate_detect = pygame.time.get_ticks() #--Will record time
912
913           if mov.y_ball > mov.y_lightray2 - 1 and mov.y_ball < mov.y_lightray2 + 1: #--When ball passes the second light gate
914             self.lightgate2_detect = pygame.time.get_ticks()
915             self.result_time = self.lightgate2_detect - self.lightgate_detect #--Stores the calculated time
916             print(self.result_time) #--Will print out the time in the shell for confirmation
917             #--Appending time and height to lists
918             mov.graphTime.append(((self.result_time)**2)/1000**2)
919             mov.graphHeight.append(1)
920
921         if mov.chooseheight == -1: #--If the light gates are separated by a distance of 0.75 metres
922           if mov.y_ball > mov.y_lightray - 1 and mov.y_ball < mov.y_lightray + 1: #--When ball passes the first light gate
923             self.lightgate_detect = pygame.time.get_ticks() #--Will record time
924
925           if mov.y_ball > mov.y_lightray2_075m - 1 and mov.y_ball < mov.y_lightray2_075m + 1: #--When ball passes the second light gate
926             self.lightgate2_detect = pygame.time.get_ticks()
927             self.result_time = self.lightgate2_detect - self.lightgate_detect #--Stores the calculated time
928             print(self.result_time) #--Will print out the time in the shell for confirmation
929             #--Appending time and height to lists
930             mov.graphTime.append(((self.result_time)**2)/1000**2)
931             mov.graphHeight.append(0.75)
932
933         if mov.chooseheight == -2: #--If the light gates are separated by a distance of 0.5 metres
934           if mov.y_ball > mov.y_lightray - 1 and mov.y_ball < mov.y_lightray + 1: #--When ball passes the first light gate
935             self.lightgate_detect = pygame.time.get_ticks() #--Will record time
936
937           if mov.y_ball > mov.y_lightray2_050m - 1 and mov.y_ball < mov.y_lightray2_050m + 1: #--When ball passes the second light gate
938             self.lightgate2_detect = pygame.time.get_ticks()
939             self.result_time = self.lightgate2_detect - self.lightgate_detect #--Stores the calculated time
940             print(self.result_time) #--Will print out the time in the shell for confirmation
941             #--Appending time and height to lists
942             mov.graphTime.append(((self.result_time)**2)/1000**2)
943             mov.graphHeight.append(0.5)
944
945         if mov.chooseheight == -3: #--If the light gates are separated by a distance of 0.25 metres
946           if mov.y_ball > mov.y_lightray - 1 and mov.y_ball < mov.y_lightray + 1: #--When ball passes the first light gate
947             self.lightgate_detect = pygame.time.get_ticks() #--Will record time
948
949           if mov.y_ball > mov.y_lightray2_025m - 1 and mov.y_ball < mov.y_lightray2_025m + 1: #--When ball passes the second light gate
950             self.lightgate2_detect = pygame.time.get_ticks()
951             self.result_time = self.lightgate2_detect - self.lightgate_detect #--Stores the calculated time
952             print(self.result_time) #--Will print out the time in the shell for confirmation
953             #--Appending time and height to lists
954             mov.graphTime.append(((self.result_time)**2)/1000**2)
955             mov.graphHeight.append(0.25)
956
```

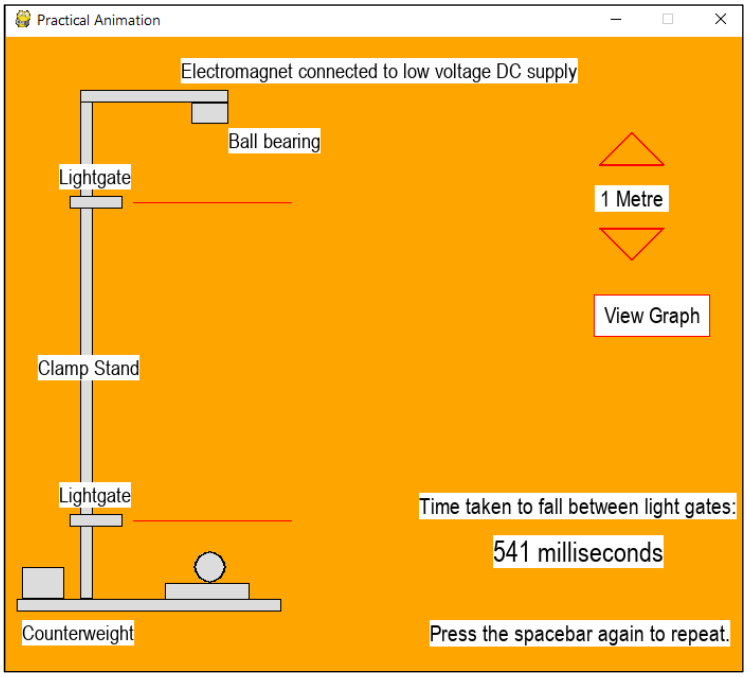
I have squared time since a graph of time squared against height can give a gradient which represents the value of ½ of 'g'. Since I have measured time in milliseconds I have divided this by 1000^2 to get this in seconds. Using seconds will be clearer for a student as this is the unit that exams typically use.

```
914                 mov.graphTime.append(((self.result_time)**2)/1000**2)
```

Next, I will create a button that can be clicked by the user which will display the Matplotlib graph:

```
441     self.x_graphButton = 555
442     self.y_graphButton = 243
443     self.x_graphButtonText = 565
444     self.y_graphButtonText = 250
445     self.width_graphButton = 110
446     self.height_graphButton = 40
447
448     self.graphButtonText = pract.fontObj_mediumsmall.render("View Graph", True, pract.BLACK, pract.WHITE)
```

```
965         #--So the user can open the graph of results
966         graphButton = pygame.draw.rect(win, pract.WHITE, (mov.x_graphButton, mov.y_graphButton, mov.width_graphButton, mov.height_graphButton))
967         graphButton = pygame.draw.rect(win, pract.RED, (mov.x_graphButton, mov.y_graphButton, mov.width_graphButton, mov.height_graphButton),1)
968         win.blit(mov.graphButtonText,(mov.x_graphButtonText, mov.y_graphButtonText))
```



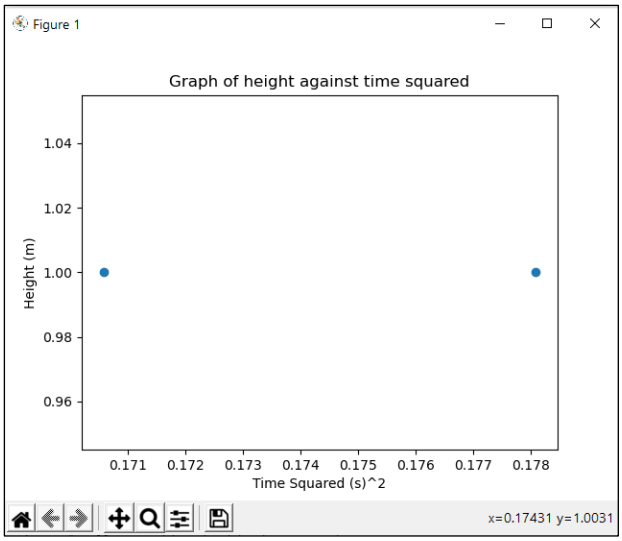
These are the boundaries for the graph button:

```
1081            #--Boundaries for the graph button
1082            if pos[0] > 555 and pos[0]< 665 and pos[1] > 243 and pos[1]< 283:
1083              event = pygame.event.poll()
1084              #--The mouse button has to also be pressed
1085              if event.type == pygame.MOUSEBUTTONDOWN:
```

When the mouse button is pressed, a simple scatter diagram will be displayed:

```
1097                    plt.scatter(mov.graphTime,mov.graphHeight)
1098                    plt.title('Graph of height against time squared')
1099                    plt.xlabel('Time Squared (s)^2')
1100                    plt.ylabel('Height (m)')
1101                    plt.show()
```

This is the first Matplotlib screen:



While the matplotlib screen was displayed successfully, I noticed that there were two values for time even though the ball was only dropped once. This is due to the ball passing each light ray twice. Since it is a scatter plot, having multiple values of time at the same height is not that bad.

Next, I will validate that the user has repeated the practical enough times so a graph of enough data points is produced. I will introduce a new attribute which keeps track of how many times the animation has been played and whether it has been done for at least three of the heights.

```
453        self.countRepeats = 0
```

Whenever the ball falls, this attribute will increment by 1:

```
975                mov.countRepeats += 1
```

```
1095                        if mov.countRepeats > 3:
```

If the graph button is pressed before this if statement is fulfilled, I have added a piece of text which prompts the user to repeat the practical.

```
454    self.repeatText = pract.fontObj_mediumsmall.render(" Repeat the experiment with different heights! ", True, pract.BLACK, pract.WHITE)
455    self.x_repeatText = 330
456    self.y_repeatText = 370
```

This is what the entire code looks like. The print statements will be used for validation which I will talk about now.

```
1091              #--Boundaries for the graph button
1092              if pos[0] > 555 and pos[0]< 665 and pos[1] > 243 and pos[1]< 283:
1093                event = pygame.event.poll()
1094                #--The mouse button has to also be pressed
1095                if event.type == pygame.MOUSEBUTTONDOWN:
1096                  #--So enough data points are plotted on the graph
1097                  if mov.countRepeats > 3:
1098                    print(mov.countRepeats)
1099                    print("Graph will be displayed")
1100                    #--Plots graph
1101                    plt.scatter(mov.graphTime,mov.graphHeight)
1102                    plt.title('Graph of height against time squared')
1103                    plt.xlabel('Time Squared (s)^2')
1104                    plt.ylabel('Height (m)')
1105                    plt.show()
1106                  else: #--User is prompted to repeat the animation with more heights
1107                    print(mov.countRepeats)
1108                    print("Graph will not be displayed")
1109                    win.blit(mov.repeatText,(mov.x_repeatText,mov.y_repeatText))
```

Validation:

Result	Valid input – Clicking the graph button and the animation has been repeated enough times	Invalid input – Clicking anything that isn't the graph button or not repeating the animation enough times	Boundary input – Clicking the graph button and repeating the practical at the same time
Expected output?	Yes, the graph was displayed	Yes, the graph wasn't displayed	Yes, the graph was displayed

My validation was successful as under the right inputs, the graph was displayed. I have included evidence of this below:

Invalid:

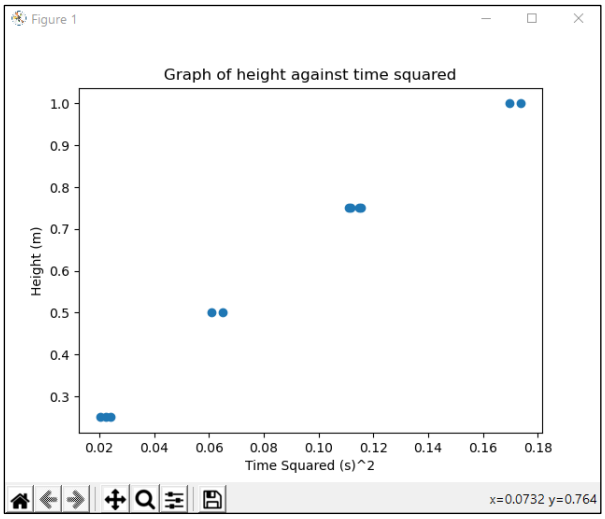
```
2
Graph will not be displayed
```

Valid and Boundary:

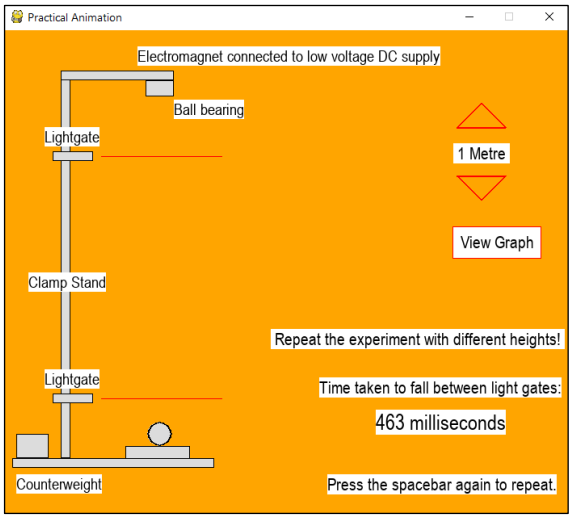
```
4
Graph will be displayed
```

The number 2 is less than than the minimum times needed for the graph to be displayed (3). Whereas with the valid and boundary tests, 4 is greater than 3.

Valid and Boundary inputs:



Invalid input:



Drawing a line of best fit

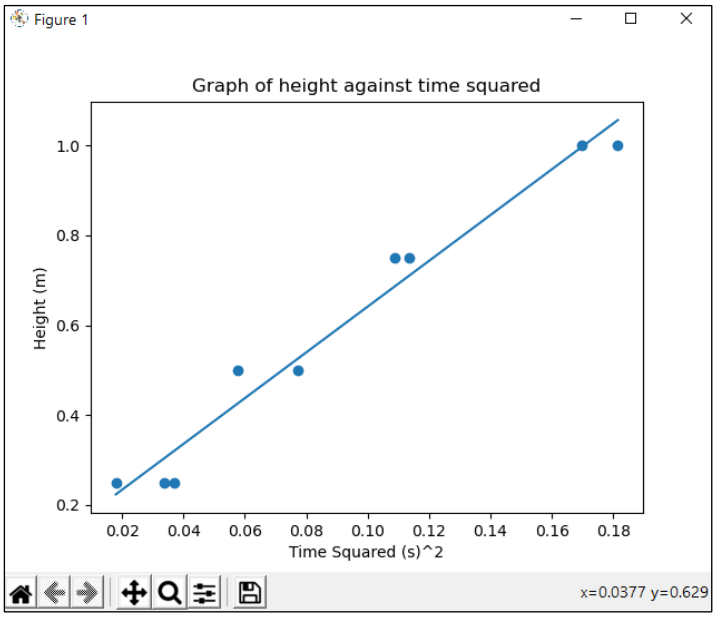
Next, I will attempt to draw a line of best fit for the data. This is particularly difficult since a line of best fit is normally done by eye in A level Physics. So for my graph, I will need to employ a formula to plot it. After conducting some research, I have imported new modules called NumPy and Polyfit which will enable me to draw the line of best fit.

```
8 import numpy as np
9 from numpy.polynomial.polynomial import polyfit
```

The line below allows the line of best fit to be drawn by inputting the data values for time and height:

```
1103                plt.plot(np.unique(mov.graphTime), np.poly1d(np.polyfit(mov.graphTime, mov.graphHeight, 1))
1104                                                                    (np.unique(mov.graphTime)))
```

This is the resulting screen:



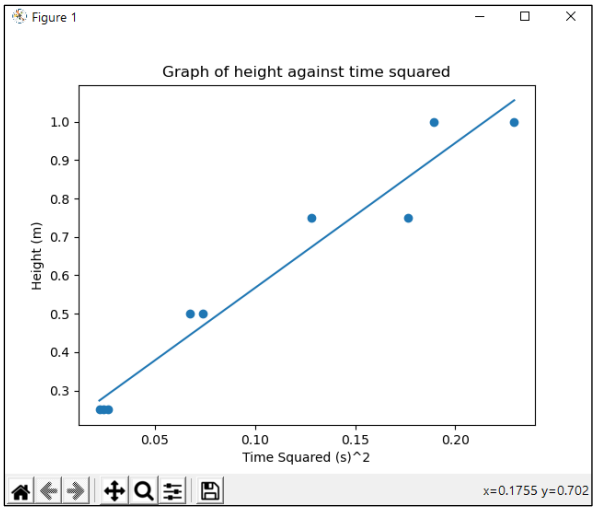
Development Testing:

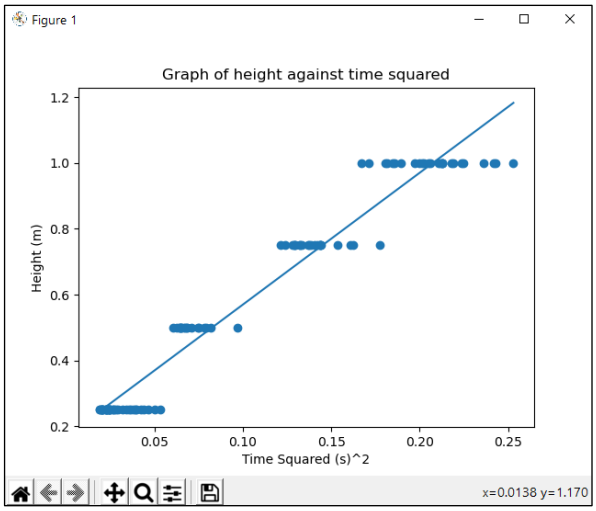
Graph button	Valid input – button is pressed	Displays graph of recorded results	Allows the user to view a graph	4.7

When the animation is repeated enough times, the graph has been successfully drawn. However, this is only for 4 data points. To test robustness, I will have my program deal with many more data points.

```
4
Graph will be displayed
```

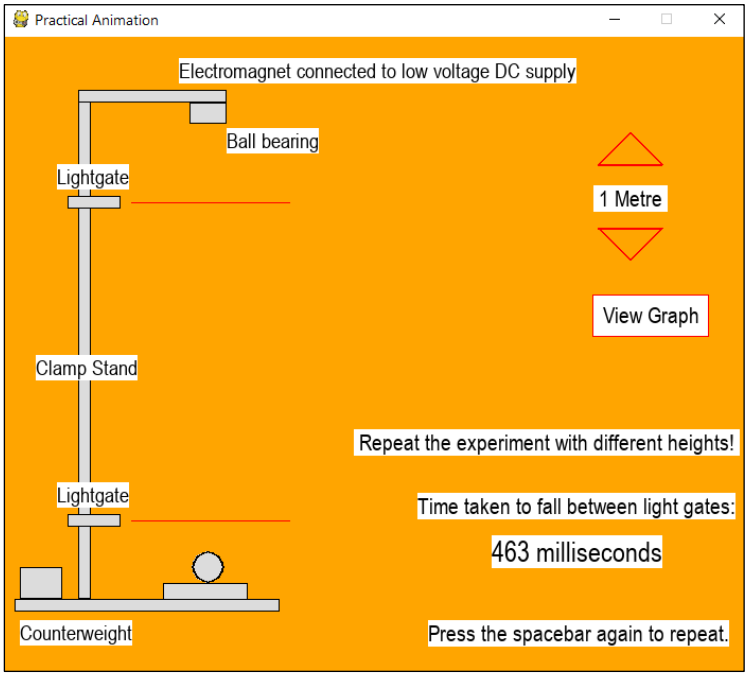
```
45
Graph will be displayed
```





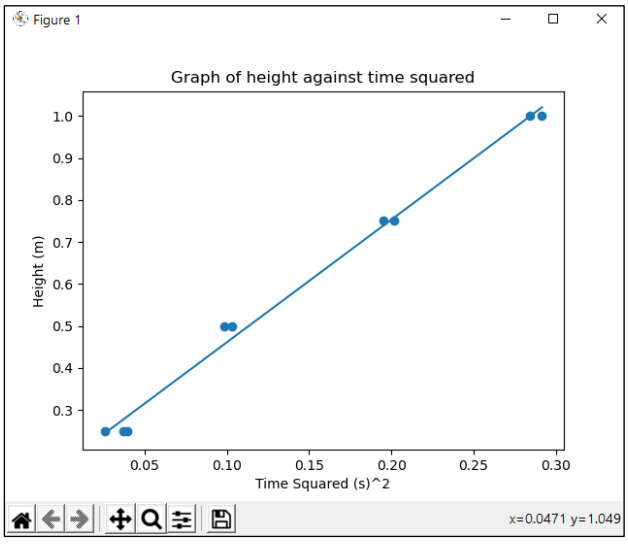
As shown above, my program demonstrated that it was robust as all 45 data points were successfully plotted and an accurate line of best fit was plotted.

If the number of times that the practical should be repeated for is not met, this is the resulting screen:



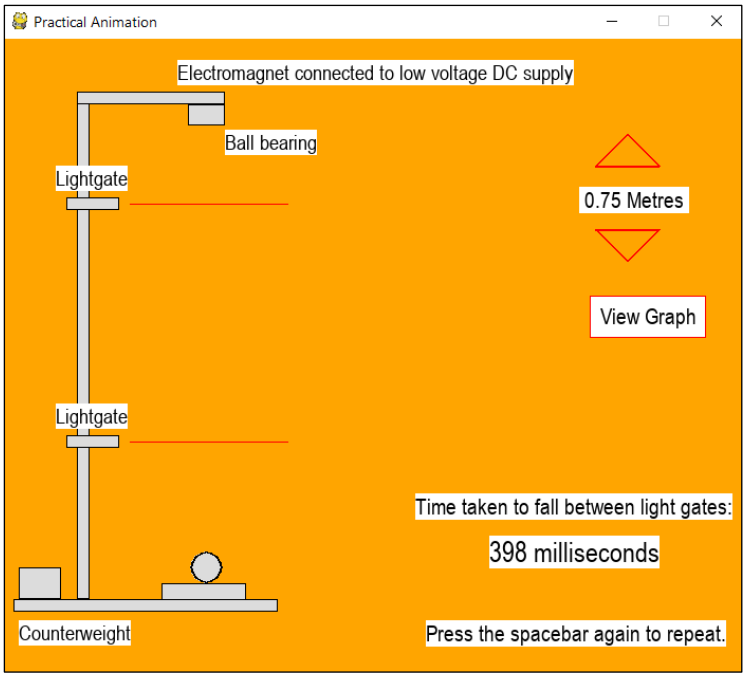
To improve the usability of the program, I want the user to be able to continue repeating the practical after they have viewed the graph. I will test whether this can be done.

Below is what happens when the graph button is pressed for the first time.

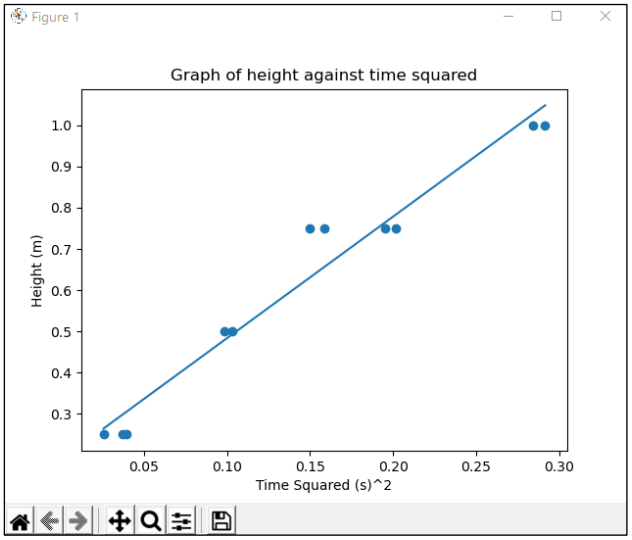



```
5
Graph will be displayed
```

When the graph window is closed, the user can repeat the practical as shown below.



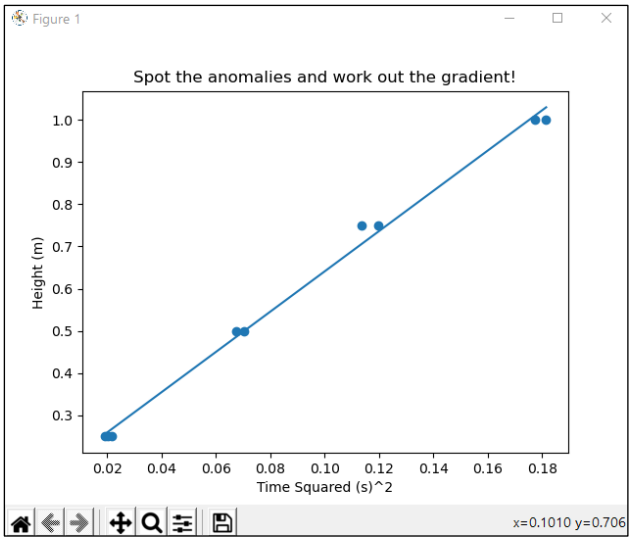
The user can even press the graph button again which will open the graph window with the latest data points. In this case, a height of 0.75 metres and a time of 398 milliseconds will be plotted.




```
6
Graph will be displayed
```

To conclude, my development testing was successful having tested for valid, invalid and boundary inputs.

While I am happy with how my program displays the graph, I want to make a change when reflecting on my success criteria. The program needs to explain to the student how they can use the graph to help them with their revision. Below, I have changed the title of the graph to prompt the student to engage with the graph.



Exiting the animation

The final stage in my Pygame animation is for the user to access the quiz. To do this, I will need the user to exit the animation window. I will include an exit button in the animation window to that this can be done.

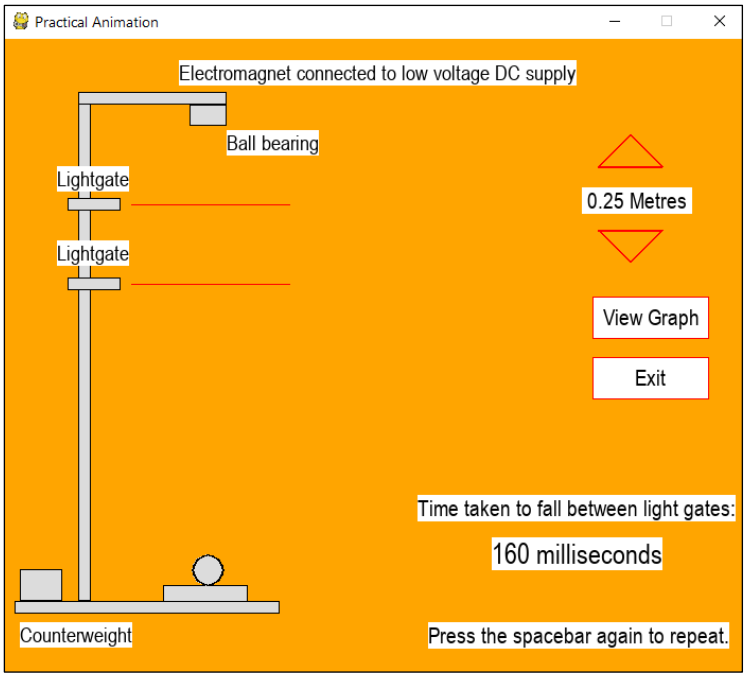
Attributes for the exit button:

```
460     self.x_exitButton = 555
461     self.y_exitButton = 300
462     self.x_exitButtonText = 595
463     self.y_exitButtonText = 307
464     self.width_exitButton = 110
465     self.height_exitButton = 40
466
467     self.exitButtonText = pract.fontObj_mediumsmall.render("Exit", True, pract.BLACK, pract.WHITE)
```

Drawing the exit button onscreen:

```
991         #--Allows the user to exit the Pygame window
992         exitButton = pygame.draw.rect(win, pract.WHITE, (mov.x_exitButton, mov.y_exitButton, mov.width_exitButton, mov.height_exitButton))
993         exitButton = pygame.draw.rect(win, pract.RED, (mov.x_exitButton, mov.y_exitButton, mov.width_exitButton, mov.height_exitButton),1)
994         win.blit(mov.exitButtonText,(mov.x_exitButtonText, mov.y_exitButtonText))
```

Resulting screen:



Like in my screen designs, the graph and exit buttons will always be displayed so the user has complete control over the animation, improving usability.

When the exit button is pressed:

```
1127            #--Boundaries for the exit button
1128            if pos[0] > 555 and pos[0]< 665 and pos[1] > 300 and pos[1]< 340:
1129              event = pygame.event.poll()
1130              #--The mouse button has to also be pressed
1131              if event.type == pygame.MOUSEBUTTONDOWN:
1132                #--Exits the window
1133                pygame.quit()
1134                sys.exit()
```

I have also imported the sys module:

```
6 import sys
```

Development Testing:

Exit button	Valid input – button is pressed	Closes the animation window	Allows the user to quit the animation and return to the main window	4.8

When the exit button was pressed, the window closed correctly. To provide evidence, I have printed the below statement in the Python shell.

```
Window closed
>>>
```

Validation:

Result	Valid input – Clicking the exit button	Invalid input – Clicking anything that isn't the exit button	Boundary input – Clicking the exit button when the graph window is open
Expected output?	Yes, the animation window closed	Yes, the animation window did not close	No, the animation window did not close

My boundary input test was not successful as the window did not close and nothing was printed in the Python shell. However, closing the graph window and then clicking the exit button was successful. Initially, I thought the while loop was preventing me from clicking the button but ending the loop once the graph was displayed didn't work. My conclusion is that when the Matplotlib window is open, it makes the Pygame window unresponsive to any inputs.

While I could not resolve this issue, I feel that it doesn't affect the usability of my program too much. If the user closes the graph window beforehand, then the animation can be exited successfully.

CREATING THE QUIZ

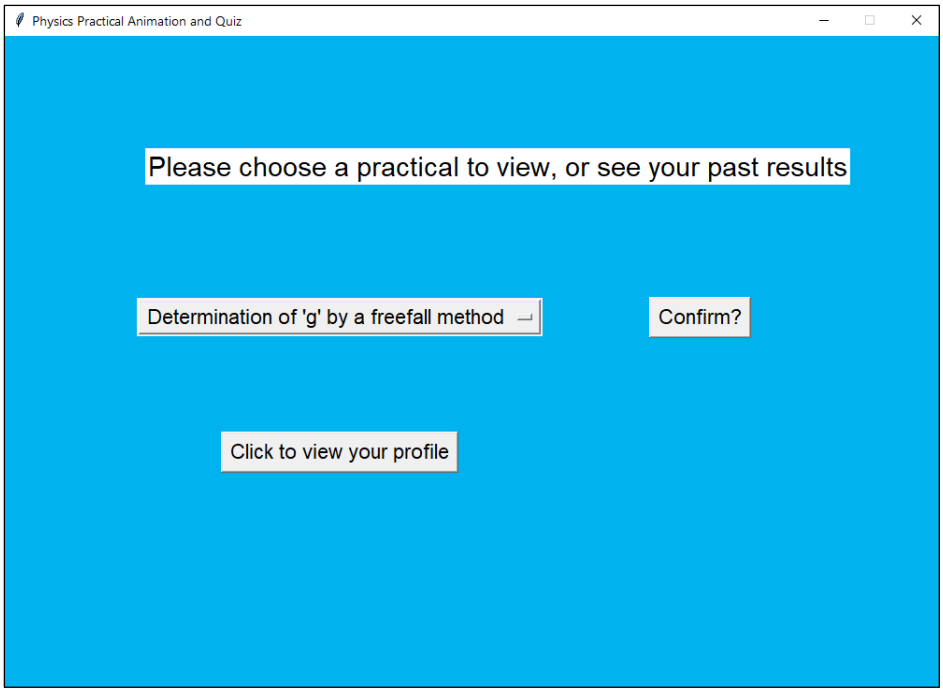
The penultimate stage in my development is to create a quiz so the user can answer questions on aspects of the practicals, improving their exam skills. As stated in my analysis and design, I will ask between 11 to 16 multiple choice questions where the user can choose from four answers, only one of which is correct. If they get this correct, they will gain a mark, and after the quiz finishes, they will get a final score.

This is the order in which I will approach the quiz:

1. Creating the screens for the quiz. To make this easier, I plan to have a template for every screen. For each screen, all I will have to do is changing the displayed questions and answers.
2. Next, I will create an Excel file where I will input all the questions and answers. This is a goal of my success criteria as I wanted CSV files to be used.
3. I will then implement a scoring system where the student will gain a mark for every question that they get right.
4. My final step is to implement a countdown timer which was a goal of my success criteria.

Creating the screens:

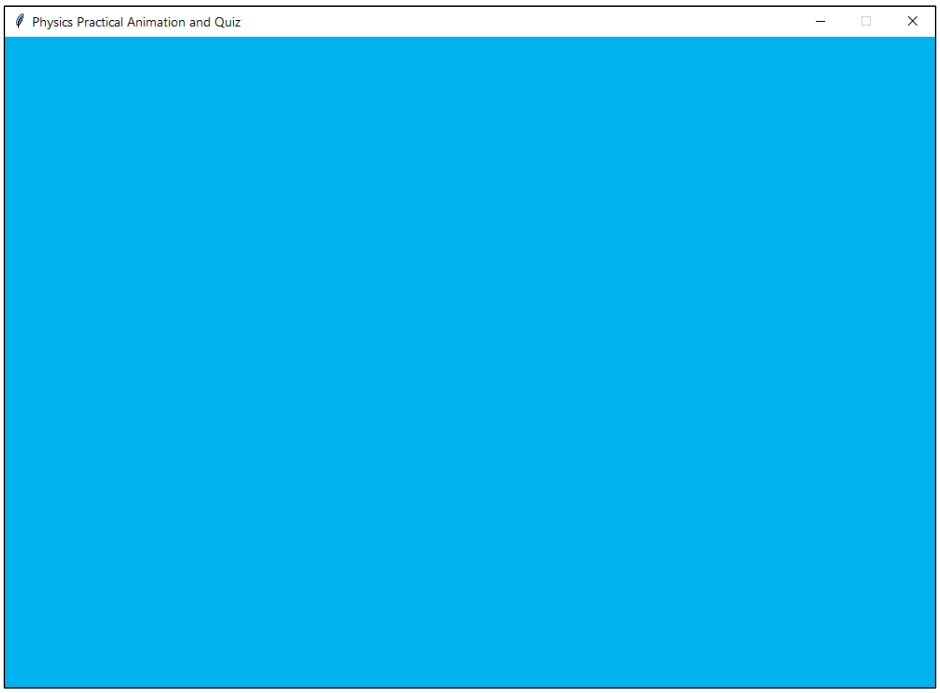
When the user first opens the animation window the Tkinter window which contains the drop-down menu remained open therefore I can use that window for my quiz. I will need to remove all the widgets in it to begin.

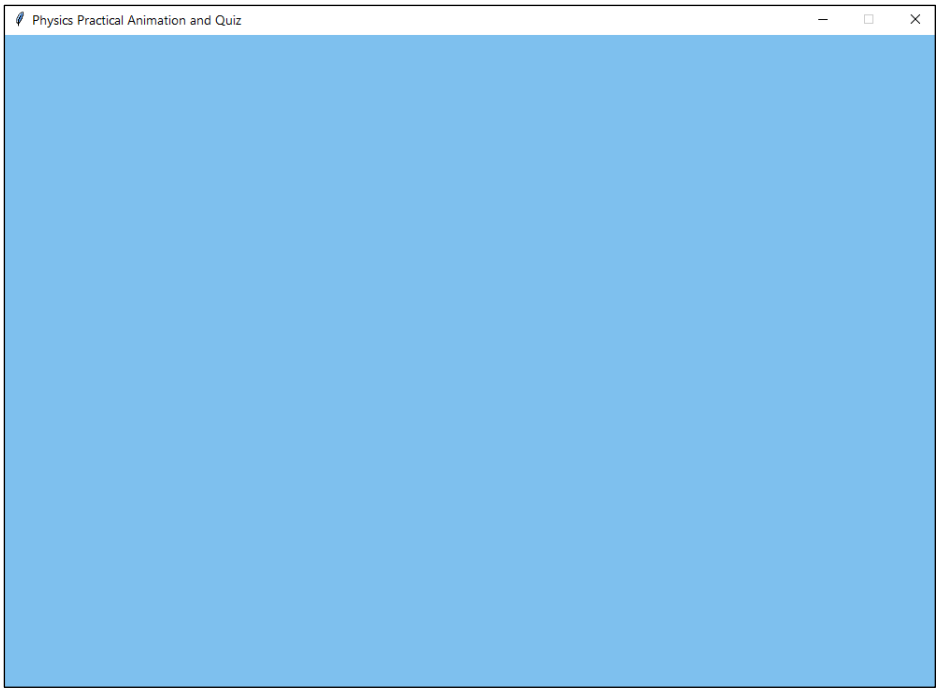


The code below removes these widgets before the animation window is first opened.

```
226   def practical(self): #--Instantiates the PracticalAnimation class which creates a new window
227     #--Removing all widgets in the screen so the user can use the quiz after viewing the animation
228     self.headingText.grid_forget()
229     self.confirm_Button2.grid_forget()
230     self.studentProfile_Button.grid_forget()
231     self.dropDown.grid_forget()
232     command = PracticalAnimation()
```

Resulting in the below screen:





I have decided to use a lighter shade of blue as I felt the other shade was too saturated. Once I finish the quiz, I will change the background colour of my login screen to this.

I have introduced a new class called Quiz where I will include all the attributes needed for the quiz screens.

```
625 class Quiz(): #--New class which will be used for the quiz
626   def __init__(self):
```

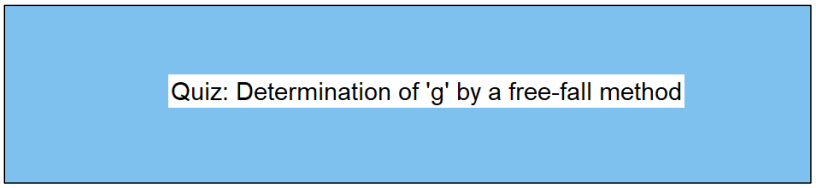
When the animation is closed by the user, this class will be instantiated allowing the user to experience the quiz.

```
1128            #--Boundaries for the exit button
1129            if pos[0] > 555 and pos[0]< 665 and pos[1] > 300 and pos[1]< 340:
1130              event = pygame.event.poll()
1131              #--The mouse button has to also be pressed
1132              if event.type == pygame.MOUSEBUTTONDOWN:
1133                #--Exits the window
1134                print("Window closed")
1135                pygame.quit()
1136                sys.exit()
1137                command = Quiz()
```

Adding new widgets

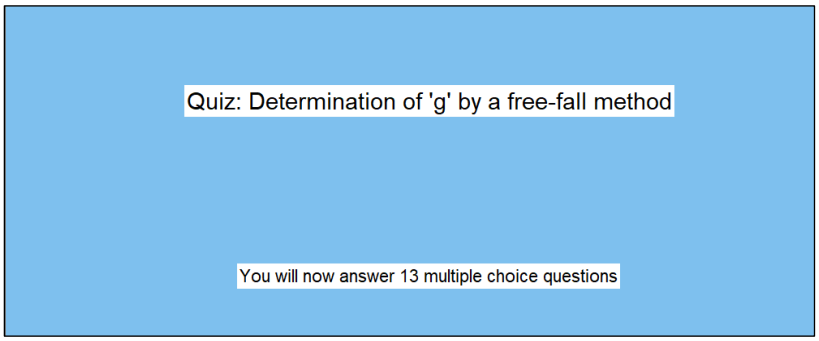
This is my first widget in the new screen, using attributes from the Format class. Similar to my login screen, since this is the main widget in the display, I have given it a column span of 9 allowing me to fit other widgets beneath it.

```
627     #--Initial screen which will allow the user to click a button to begin the practical
628     self.quizMaintext = Label(window, text="Quiz: Determination of 'g' by a free-fall method",
629                               fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
630     self.quizMaintext.grid(row=0, columnspan=9, pady = 100, padx = 220) #--Main widget in the window
```



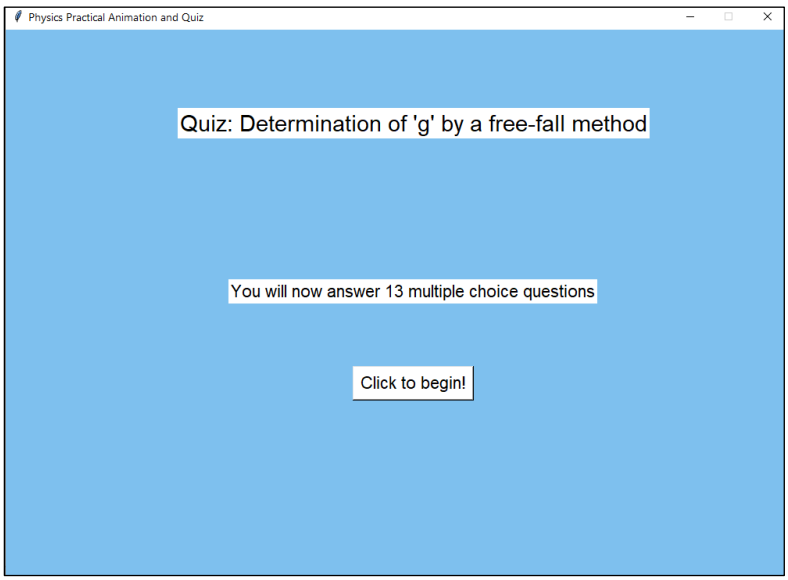
The next widget in the display:

```
627     #--Initial screen which will allow the user to click a button to begin the practical
628     self.quizMaintext = Label(window, text="Quiz: Determination of 'g' by a free-fall method",
629                               fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
630
631     self.startText = Label(window, text="You will now answer 13 multiple choice questions",
632                               fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium))
633
634
635     self.quizMaintext.grid(row=0, columnspan=9, pady = 100, padx = 220) #--Main widget in the window
636     self.startText.grid(row=1, column=4, pady = 80)
```



The third widget will be a button that the user can press to begin the quiz. If I have enough time, pressing this button would also start the timer.

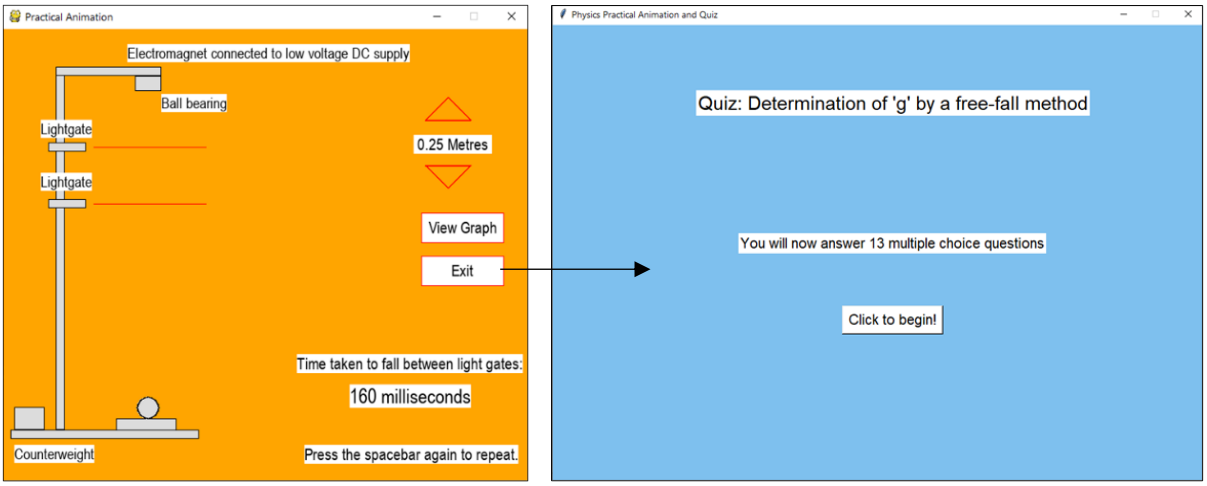
```
627     #--Initial screen which will allow the user to click a button to begin the practical
628     self.quizMaintext = Label(window, text="Quiz: Determination of 'g' by a free-fall method",
629                               fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
630
631     self.startText = Label(window, text="You will now answer 13 multiple choice questions",
632                               fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium))
633
634     self.startButton = Button(window, text="Click to begin!", fg=form.black,bg=form.white,
635                               font=(form.font, form.fontSizeMedium), command = self.LoadQuestions)
636
637     self.quizMaintext.grid(row=0, columnspan=9, pady = 100, padx = 220) #--Main widget in the window
638     self.startText.grid(row=1, column=2, columnspan = 5,pady = 80)
639     self.startButton.grid(row=2, column=4)
```



Development Testing:

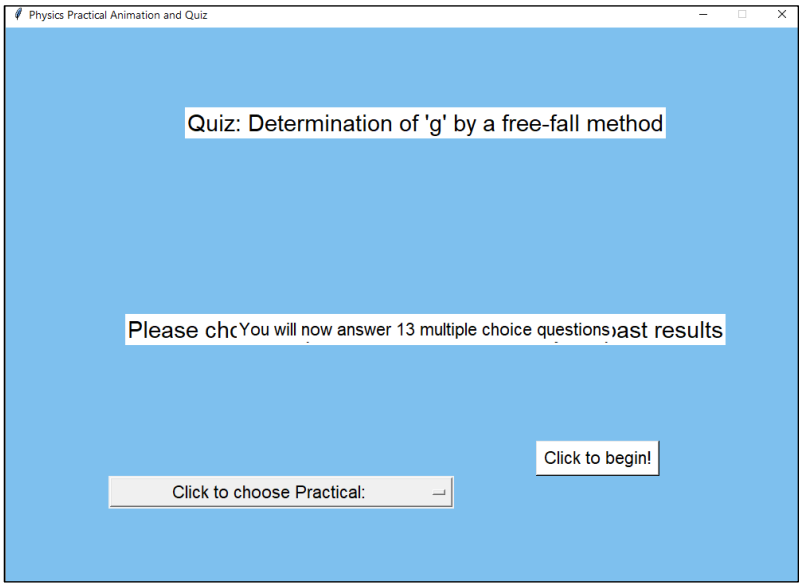
Quiz Button	Valid input – button is pressed	Finishes the animation and allows the user to move onto the quiz	Allows the user to complete the quiz	5.0

When the exit was pressed, the user was successfully transitioned to the new quiz screen. I have even printed a statement in the Python shell to ensure that this was happening.




```
Screen Transitioned
```

To test the robustness of my program, I have repeated this test. When doing so, this transition didn't always work as expected. Below is an example of the quiz screen. For some reason, the widgets from the previous Tkinter screen which displayed the drop-down menu were not removed meaning the new widgets could not be displayed properly. My statement was still being printed so I knew the quiz class was being instantiated but I could not figure out my problem. Therefore, this test was unsuccessful.



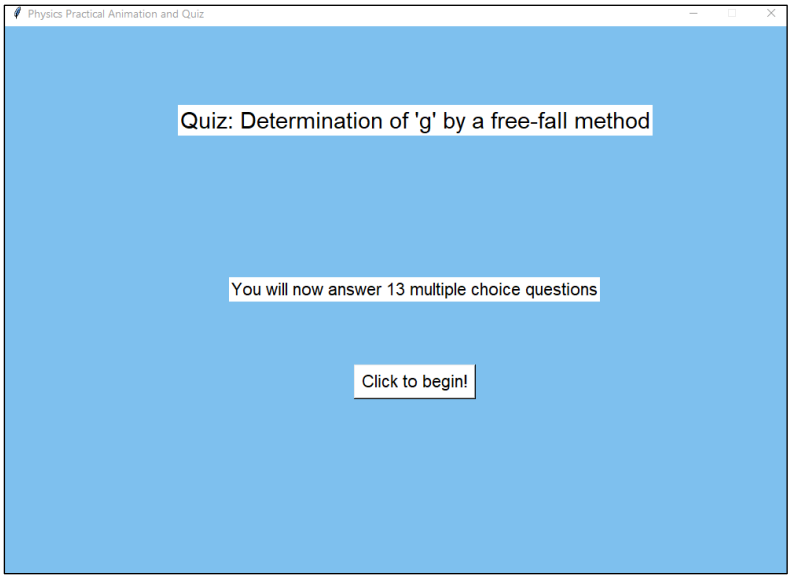

```
Screen Transitioned
```

Solution:

To fix this, I decided to destroy the old window used for the login screen and use a new window exclusively for the quiz. This is not ideal as it makes it harder to code the process of returning the user to the login screen once they finish the program.

The code below destroys the original Tkinter window and creates an identical new one but with a different name. For the new widgets to be displayed, I won't have to do anything as there is only one window that can be displayed too.

```
621 class Quiz(): #--New class which will be used for the quiz
622   def __init__(self):
623     window.destroy() #--Destroying the old window
624     window2 = Tk()
625     #--Creating the new window which will be identical
626     window2.title("Physics Practical Animation and Quiz")#---Name of the window
627     window2.geometry("1000x700") #---Sets the height and width of the tkinter window
628     window2.configure(bg="SkyBlue2") #---Sets the background colour of the window
629     window2.resizable(False, False) #--Tkinter window is not resizable
```



Screen Transitioned

When testing this for robustness, the screen transition always worked. Therefore, this test was a success.

Storing and loading the questions and answers

These are the questions that will be asked to the user. There will be a total of four answers to choose from, only one of which will be correct.

Question Number	Question	Correct Answer	Incorrect Answer 1	Incorrect Answer 2	Incorrect Answer 3
1.	Describe the relationship of the drawn graph	Directly Proportional	Proportional	Inversely Proportional	Indirectly Proportional
2.	Click on the closest value to the gradient of the graph	4.9	3	6	9.8

3.	What is the physical significance of the gradient?	½ Acceleration due to gravity	Acceleration due to gravity	Speed of the ball bearing	Gravitational Field Strength
4.	What is an independent variable in this experiment?	Height between light gates	Speed of the ball	Time it takes for the ball to fall	Acceleration of the ball
5.	What is a dependant variable in this experiment?	Time it takes for the ball to fall	Height between light gates	Acceleration of the ball	Speed of the ball
6.	State a control variable	Same surface area of the ball	Time it takes for the ball to fall	Height between light gates	Acceleration of the ball
7.	What is an advantage of using an automatic timer compared to a stopwatch?	More accurate results due to the elimination of human reaction time	More precise results due to the elimination of zero errors	Less likely to fail compared to a stopwatch	Less work needed to operate it
8.	Which of the following do **not** influence the ball bearing's acceleration?	Mass	Air Resistance	Both	Neither
9.	Why is the pad used?	To prevent a trip accident if the ball falls onto the floor	To prevent it from falling on the floor	So the ball bearing doesn't bounce on impact	To quieten the impact of the ball
10.	What is the disadvantage of using an electromagnet?	There could be a small delay between it becoming demagnetised and the ball dropping	The ball may not drop at all and still be attracted to the electromagnet	The electromagnet may not always switch off properly	Electromagnets are more expensive
11.	What is the problem with using small heights?	Increases percentage uncertainty when measuring time	Ball bearing will not accelerate that much	Harder to measure smaller changes in time	Will give erroneous results for the value of g
12.	Why must the distance between the first light gate and the ball be kept the same?	Reduces percentage uncertainty when measuring height	So that the ball enters the first light gate with the same speed	Easier to measure height	Distance doesn't actually matter
13.	Why must the ball be dense?	To mitigate the effect of air resistance	To ensure the ball falls in a straight line	So the ball falls faster	So the ball doesn't bounce off the pad

I will add these questions and answers to an Excel file which will be saved in the CSV format.

	B	C	D	E	F
1	Question	Correct Answer	Incorrect Answer 1	Incorrect Answer 2	Incorrect Answer 3
2	Describe the relationship of the drawn graph	Directly Proportional	Proportional	Inversely Proportional	Indirectly Proportional
3	Click on the closest value to the gradient of the g	4.9	3	6	9.8
4	Therefore, what is the physical significance of th	1/2 Acceleration due to gravity	Acceleration due to gravity	Speed of the ball bearing	Gravitational Field Strength
5	What is the independent variable in this experim	Height between light gates	Speed of the ball	Time it takes for the ball to fall	Acceleration of the ball
6	What is the dependant variable in this experime	Time it takes for the ball to fall	Height between light gates	Acceleration of the ball	Speed of the ball
7	State a control variable	Same surface area of the ball	Time it takes for the ball to fall	Height between light gates	Acceleration of the ball
8	What is an advantage of using an automatic tim	More accurate results due to the elimation of human	More precise results due to the elimination of	Less likely to fail compared to a stopw	Less work needed to operate it
9	Which of the following do not influence the ball	Mass	Air Resistance	Both	Neither
10	Why is the pad used?	To prevent a trip accident if the ball falls onto the flo	To prevent it from falling on the floor	So the ball bearing doesn't bounce on	To quieten the impact of the ball
11	What is a disadvantage of using an electromagn	There could be a small delay between it becoming de	The ball may not drop at all and still be attract	The electromagnet may not always sw	Electromagnets are more expensive
12	What is the problem with using small heights?	Increases percentage uncertainty when measuring tir	Ball bearing will not accelerate that much	Harder to measure smaller changes in	Will give erroneous results for the value of g
13	Why must the distance between the first light ga	Reduces percentage uncertainty when measuring hei	So that the ball enters the first light gate with	Easier to measure height	Distance doesn't actually matter
14	Why must the ball be dense?	To mitigate the effect of air resistance	To ensure the ball falls in a straight line	So the ball falls faster	So the ball doesn't bounce off the pad

On line 635, when the start button is pressed, the LoadQuestions method will be run.

```
634     self.startButton = Button(window, text="Click to begin!", fg=form.black,bg=form.white,
635                               font=(form.font, form.fontSizeMedium), command = self.LoadQuestions)
```

The screenshot below shows the method of appending the contents from the Excel file to a 2D array. This is similar to my login screen where I appended the student's details from a file to an array.

```
642     self.quizArray = [] #--Array of questions and answers that will be used
643   def LoadQuestions(self):
644     with open('Quiz_Questions&Answers.csv', 'r') as Quiz_QA:
645         reader = csv.reader(Quiz_QA)
646         for row in reader: #--File contents are appended as a 2d array
647           self.quizArray.append(row)
648     print(self.quizArray)
```

Development Testing

Displaying questions from CSV file	Valid input – button to start the quiz is pressed	Displays the questions	So the user can attempt the quiz	5.2

This is the result of the Python shell. I can confirm that appending questions from a CSV file to a 2D array was successful.

```
>>> [['Question Number', 'Question', 'Correct Answer', 'Incorrect Answer 1', 'Incorrect Answer 2', 'Incorrect Answer 3']
, ['1', 'Describe the relationship of the drawn graph', 'Directly Proportional', 'Proportional', 'Inversely Proportional
', 'Indirectly Proportional'], ['2', 'Click on the closest value to the gradient of the graph', '4.9', '3', '6', '9.8'],
['3', 'Therefore, what is the physical significance of the gradient?', '1/2 Acceleration due to gravity', 'Acceleration
due to gravity', 'Speed of the ball bearing', 'Gravitational Field Strength'], ['4', 'What is the independent variable i
n this experiment?', 'Height between light gates', 'Speed of the ball', 'Time it takes for the ball to fall', 'Accelerat
ion of the ball'], ['5', 'What is the dependant variable in this experiment?', 'Time it takes for the ball to fall', 'He
ight between light gates', 'Acceleration of the ball', 'Speed of the ball'], ['6', 'State a control variable', 'Same sur
face area of the ball', 'Time it takes for the ball to fall', 'Height between light gates', 'Acceleration of the ball'],
['7', 'What is an advantage of using an automatic timer compared to a stopwatch?', 'More accurate results due to the eli
mation of human reaction time', 'More precise results due to the elimination of zero errors', 'Less likely to fail compa
red to a stopwatch', 'Less work needed to operate it'], ['8', "Which of the following do not influence the ball bearing'
s acceleration?", 'Mass', 'Air Resistance ', 'Both', 'Neither'], ['9', 'Why is the pad used?', 'To prevent a trip accide
nt if the ball falls onto the floor', 'To prevent it from falling on the floor', "So the ball bearing doesn't bounce on
impact", 'To quieten the impact of the ball'], ['10', 'What is a disadvantage of using an electromagnet?', 'There could
be a small delay between it becoming demagnetised and the ball dropping', 'The ball may not drop at all and still be att
racted to the electromagnet', 'The electromagnet may not always switch off properly', 'Electromagnets are more expensive
'], ['11', 'What is the problem with using small heights?', 'Increases percentage uncertainty when measuring time', 'Bal
l bearing will not accelerate that much', 'Harder to measure smaller changes in time', 'Will give erroneous results for
the value of g'], ['12', 'Why must the distance between the first light gate and the ball be kept the same?', 'Reduces p
ercentage uncertainty when measuring height', 'So that the ball enters the first light gate with the same speed', 'Easie
r to measure height ', 'Distance doesn't actually matter'], ['13', 'Why must the ball be dense?', 'To mitigate the effec
t of air resistance ', 'To ensure the ball falls in a straight line', 'So the ball falls faster', 'So the ball doesn't b
ounce off the pad']]
```

Displaying the questions and answers onscreen:

Next, I will create the screens for displaying the questions and answers. I will create a template that I can use for all 13 questions.

```
659     command = self.DisplayQuiz()
660
661   def DisplayQuiz(self):
662     #--Removes widgets in the window
663     self.quizMaintext.grid_forget()
664     self.startText.grid_forget()
665     self.startButton.grid_forget()
666
667     #--First Question
668     self.QuestionNumber = Label(text=f' Question {self.quizArray[1][0]}: ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
669     self.Question = Label(text=f'{self.quizArray[1][1]} ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
670     self.CorrectAnswer = Button(text=f'  {self.quizArray[1][2]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),command = self.CorrectAnswer)
671     self.IncorrectAnswer1 = Button(text=f'  {self.quizArray[1][3]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),command = self.IncorrectAnswer)
672     self.IncorrectAnswer2 = Button(text=f'  {self.quizArray[1][4]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),command = self.IncorrectAnswer)
673     self.IncorrectAnswer3 = Button(text=f'  {self.quizArray[1][5]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),command = self.IncorrectAnswer)
674     #--Determines positioning of widgets
675     self.QuestionNumber.grid(row = 1, column = 1, columnspan = 1,padx = 400, pady = 50)
676     self.Question.grid(row = 2, column = 1, columnspan = 1,pady = 70)
677     self.CorrectAnswer.grid(row = 3, column = 1, padx = 100,pady = 20)
678     self.IncorrectAnswer1.grid(row = 4, column = 1, padx = 100,pady = 20)
679     self.IncorrectAnswer2.grid(row = 5, column = 1, pady = 20)
680     self.IncorrectAnswer3.grid(row = 6, column = 1, pady = 20)
```

Once all the questions and answers have been loaded into the 2D array, I have created a new method which will display the questions and answers. From lines 662 to 665, I have cleared the screen. From lines 668 to 673, I have displayed the first question and corresponding answers from the array. From lines 675 to 680, I have used the grid packet manager to arrange these widgets.

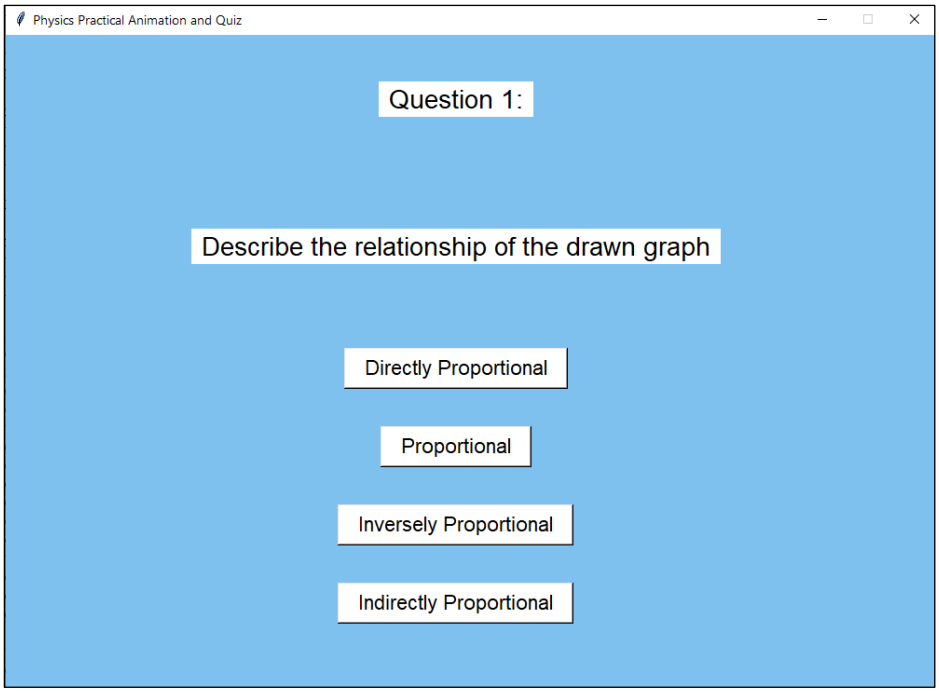
I have written 1 instead of 0 within the double brackets since the first items in the array are headings designed to improve the readability of the Excel file.

```
1264        self.Question = Label(text=f' {self.quizArray[1][1]} '
```

Development Testing:

Displaying questions from CSV file	Valid input – button to start the quiz is pressed	Displays the questions	So the user can attempt the quiz	5.2

This is a continuation from my previous test where I test whether the questions can be displayed.



The first question and answers are displayed successfully, therefore, this test was a success

Detecting which answers the user clicked

Next, I will work on detecting which button is clicked and determining whether this answer is correct or not.

In the below screenshot, when each button is clicked, it will initiate a new method that will tell the user whether they have gotten the question correct or not. If they press the correct answer, the CorrectAnswer method will be run but if they press the wrong answer the IncorrectAnswer method will be run. This is shown by lines 670 to 673.

```
659     command = self.DisplayQuiz()
660
661   def DisplayQuiz(self):
662     #--Removes widgets in the window
663     self.quizMaintext.grid_forget()
664     self.startText.grid_forget()
665     self.startButton.grid_forget()
666
667     #--First Question
668     self.QuestionNumber = Label(text=f' Question {self.quizArray[1][0]}: ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
669     self.Question = Label(text=f'{self.quizArray[1][1]} ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
670     self.CorrectAnswer = Button(text=f'  {self.quizArray[1][2]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),command = self.CorrectAnswer)
671     self.IncorrectAnswer1 = Button(text=f'  {self.quizArray[1][3]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),command = self.IncorrectAnswer)
672     self.IncorrectAnswer2 = Button(text=f'  {self.quizArray[1][4]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),command = self.IncorrectAnswer)
673     self.IncorrectAnswer3 = Button(text=f'  {self.quizArray[1][5]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),command = self.IncorrectAnswer)
674     #--Determines positioning of widgets
675     self.QuestionNumber.grid(row = 1, column = 1, columnspan = 1,padx = 400, pady = 50)
676     self.Question.grid(row = 2, column = 1, columnspan = 1,pady = 70)
677     self.CorrectAnswer.grid(row = 3, column = 1, padx = 100,pady = 20)
678     self.IncorrectAnswer1.grid(row = 4, column = 1, padx = 100,pady = 20)
679     self.IncorrectAnswer2.grid(row = 5, column = 1, pady = 20)
680     self.IncorrectAnswer3.grid(row = 6, column = 1, pady = 20)
```

If the user gets the answer correct

If the user presses the correct answer, this new method detailed below will run. It will remove all the widgets in the window and then repaint the background colour green indicating to the user that they have got the question right. This links back to my secondary research on the learning tool Memrise since I liked how it used colours to show whether the student got the question right or wrong.

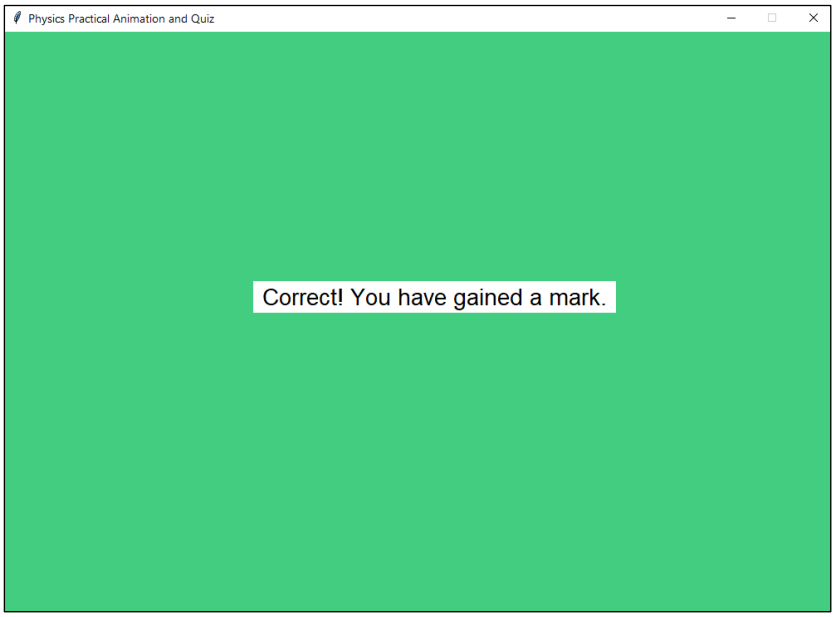
```
683   #--If the first question is answered correctly
684   def CorrectAnswer(self):
685     #--Removes all widgets in the window
686     self.QuestionNumber.grid_forget()
687     self.Question.grid_forget()
688     self.CorrectAnswer.grid_forget()
689     self.IncorrectAnswer1.grid_forget()
690     self.IncorrectAnswer2.grid_forget()
691     self.IncorrectAnswer3.grid_forget()
692     #--Paints the background green
693     self.window2.configure(bg=form.green)
694     self.correctAnswerText = Label(text=f' Correct! You have gained a mark. ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
695     self.correctAnswerText.grid(padx = 300, pady = 300)
696
697     #--Increases the score by 1
698     self.score += 1
699     #--Displays this window for 2 seconds before displaying the next question
700     self.window2.after(2000, lambda: self.DisplayQuiz())
```

On line 700, this piece of code makes this window appear for only two seconds. This is so the program can then display the next question.

Furthermore, I have introduced a scoring system to keep track of the marks that the user has gained. For now I have kept this simple where it increments every time the user clicks on the correct question (line 698). I will go into more detail on this when I finish the quiz and begin coding the graph of the user's progress.

```
    self.score = 0 #--Sets score to 0
```

This is what the screen looks like:

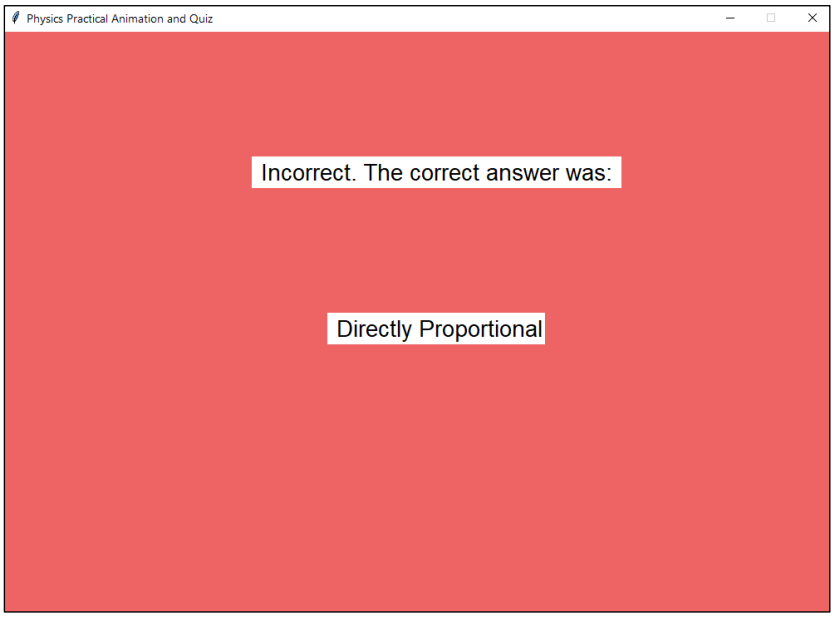


If the user gets the answer incorrect

I can follow the same principle as above when making the screen if the user gets the question incorrect.

```
752   #--If the first question is answered incorrectly
753   def IncorrectAnswer(self):
754     #--Removes all widgets in the window
755     self.QuestionNumber.grid_forget()
756     self.Question.grid_forget()
757     self.CorrectAnswer.grid_forget()
758     self.IncorrectAnswer1.grid_forget()
759     self.IncorrectAnswer2.grid_forget()
760     self.IncorrectAnswer3.grid_forget()
761     #--Paints the background red
762     self.window2.configure(bg=form.red)
763     #--Prints the correct answer to the question
764     self.incorrectAnswerText = Label(text=f' Incorrect. The correct answer was: ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
765     self.correctAnswer = Label(text=f' {self.quizArray[1][2]}', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
766     self.incorrectAnswerText.grid(row = 1, column = 2, padx = 300, pady = 150)
767     self.correctAnswer.grid(row = 2, column = 2)
768
769
770     #--Displays this window for 3 seconds before displaying the next question
771     self.window2.after(3000, lambda: self.DisplayQuiz())
```

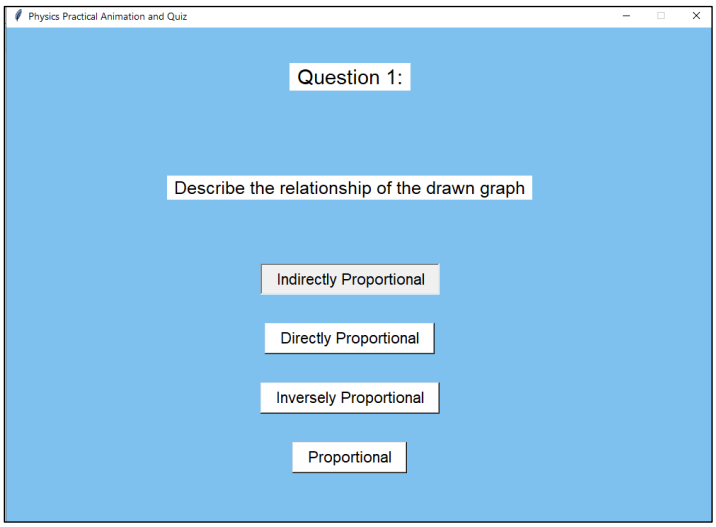
The only changes I have made is changing the background colour to red and displaying the correct answer to the question. Additionally, as a usability feature, I have allowed more time for the user to spend on this screen so they can see what the correct answer was.



Validation

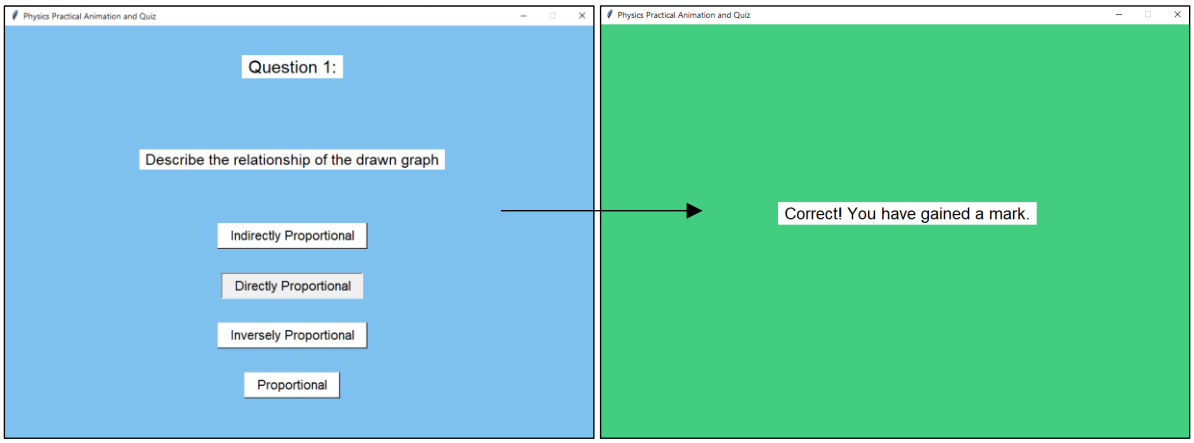
I now validate the different inputs that a user could enter.

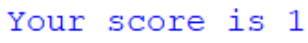
Result	Valid input – Clicking any of the multiple-choice buttons	Invalid input – Clicking anything that isn't a multiple-choice button	Boundary input – Repeatedly clicking a button or holding it down instead
Expected output?	Yes, the question screen was transitioned into the new correct/incorrect screens	Yes, the question screen remained as it was	Yes. Repeatedly clicking the button made no difference and the program received it as a valid input.



Answer buttons (Will consist of 4 multiple choice answers)	Valid input – button is pressed	If the answer is correct, the user will gain a mark and the next question should be displayed	To check if the student's answer is correct and to allow them to move onto the next question	5.3
		If the answer is incorrect, the user will get to see the correct answer. Then the next question will be displayed.	To check if the student's answer is incorrect and to allow them to move onto the next question	5.4

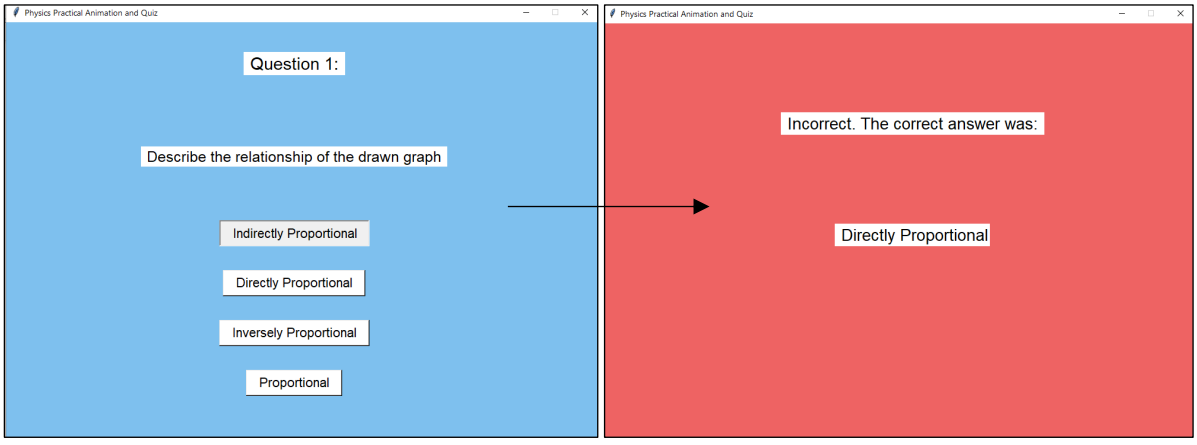
Correct answer:

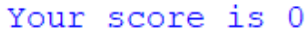




The screen was transitioned correctly and the score increased by 1.

Incorrect answer:





Tests 5.3 and 5.4 were a success.

Displaying the next question

I have created a new attribute which will allow the program to choose the next question to be displayed. This attribute is initially set to 0 but will increment each time the user gets an answer correct or incorrect

```
650         self.chooseQuestion = 0
```

When the DisplayQuiz method is run again, I have incorporated if statements which determine which question should be displayed next. When the 'self.chooseQuestion' attribute is set to 0, the first set of questions and answers will be displayed. When the attribute increases by 1, the second set of questions and answers will be displayed.

```
670     if self.chooseQuestion == 0:
671       #--First Question
672       self.QuestionNumber = Label(text=f' Question {self.quizArray[1][0]}: ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
673       self.Question = Label(text=f' {self.quizArray[1][1]} ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
674       #--Correct Answer
675       self.CorrectAnswer = Button(text=f'  {self.quizArray[1][2]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
676                                                                                                   command = self.CorrectAnswer)
677       #--Incorrect Answers
678       self.IncorrectAnswer1 = Button(text=f'  {self.quizArray[1][3]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
679                                                                                                   command = self.IncorrectAnswer)
680       self.IncorrectAnswer2 = Button(text=f'  {self.quizArray[1][4]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
681                                                                                                   command = self.IncorrectAnswer)
682       self.IncorrectAnswer3 = Button(text=f'  {self.quizArray[1][5]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
683                                                                                                   command = self.IncorrectAnswer)
684       #--Determines positioning of widgets
685       self.QuestionNumber.grid(row = 1, column = 1, columnspan = 1,padx = 400, pady = 50)
686       self.Question.grid(row = 2, column = 1, columnspan = 1,pady = 70)
687       self.CorrectAnswer.grid(row = 3, column = 1, padx = 100,pady = 20)
688       self.IncorrectAnswer1.grid(row = 4, column = 1, padx = 100,pady = 20)
689       self.IncorrectAnswer2.grid(row = 5, column = 1, pady = 20)
690       self.IncorrectAnswer3.grid(row = 6, column = 1, pady = 20)
691
692     elif self.chooseQuestion == 1:
693       #--Second Question
694       self.QuestionNumber = Label(text=f' Question {self.quizArray[2][0]}: ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
695       self.Question = Label(text=f' {self.quizArray[2][1]} ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
696       #--Correct Answer
697       self.CorrectAnswer = Button(text=f'  {self.quizArray[2][2]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
698                                                                                                   command = self.CorrectAnswer2)
699       #--Incorrect Answers
700       self.IncorrectAnswer1 = Button(text=f'  {self.quizArray[2][3]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
701                                                                                                   command = self.IncorrectAnswer)
702       self.IncorrectAnswer2 = Button(text=f'  {self.quizArray[2][4]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
703                                                                                                   command = self.IncorrectAnswer)
704       self.IncorrectAnswer3 = Button(text=f'  {self.quizArray[2][5]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
705                                                                                                   command = self.IncorrectAnswer)
706       #--Determines positioning of widgets
707       self.QuestionNumber.grid(row = 1, column = 1, columnspan = 1,padx = 400, pady = 50)
708       self.Question.grid(row = 2, column = 1, columnspan = 1,pady = 70)
709       self.CorrectAnswer.grid(row = 3, column = 1, padx = 100,pady = 20)
710       self.IncorrectAnswer1.grid(row = 4, column = 1, padx = 100,pady = 20)
711       self.IncorrectAnswer2.grid(row = 5, column = 1, pady = 20)
712       self.IncorrectAnswer3.grid(row = 6, column = 1, pady = 20)
```

When the user gets an answer correct or incorrect, the 'self.chooseQuestion' attribute increments as shown on line 734 below. The 'DisplayQuiz' method is repeated so that the next set of questions and answers can be displayed.

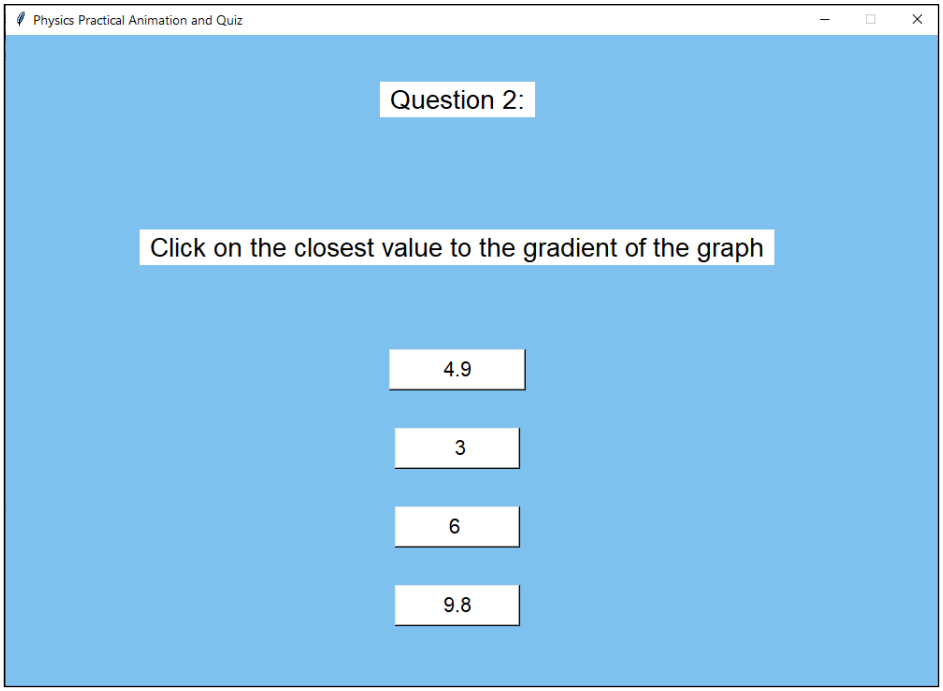
```
715   #--If the first question is answered correctly
716   def CorrectAnswer(self):
717     #--Removes all widgets in the window
718     self.QuestionNumber.grid_forget()
719     self.Question.grid_forget()
720     self.CorrectAnswer.grid_forget()
721     self.IncorrectAnswer1.grid_forget()
722     self.IncorrectAnswer2.grid_forget()
723     self.IncorrectAnswer3.grid_forget()
724     #--Paints the background green
725     self.window2.configure(bg=form.green)
726     #--Prints text
727     self.correctAnswerText = Label(text=f' Correct! You have gained a mark. ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
728     self.correctAnswerText.grid(padx = 300, pady = 300)
729
730     #--Increases the score by 1
731     self.score += 1
732
733     #--Allows the next question to be displayed
734     self.chooseQuestion += 1
735     #--Displays this window for 2 seconds before displaying the next question
736     self.window2.after(2000, lambda: self.DisplayQuiz())
737     self.window2.after(2000, lambda: self.correctAnswerText.grid_forget())
```

Line 737 removes the text that was displayed on line 727.

Before the user can transition to the next question, I will need to repaint the background:

```
662    def DisplayQuiz(self):
663      #--Removes widgets in the window
664      self.quizMaintext.grid_forget()
665      self.startText.grid_forget()
666      self.startButton.grid_forget()
667      #--Repaints the background to the original colour
668      self.window2.configure(bg="SkyBlue2")
```

This is the screen when the second question is displayed:



Checking whether the user has got the answer correct or incorrect

I can use the same method for checking the user's input as previously described. Unfortunately, I wasn't able to reuse the same 'correctAnswer' method. I needed to create a new method with the same code but a different name.

```
997   #--If the second question is answered correctly
998   def CorrectAnswer2(self):
999     #--Removes all widgets in the window
1000    self.QuestionNumber.grid_forget()
1001    self.Question.grid_forget()
1002    self.CorrectAnswer.grid_forget()
1003    self.IncorrectAnswer1.grid_forget()
1004    self.IncorrectAnswer2.grid_forget()
1005    self.IncorrectAnswer3.grid_forget()
1006    #--Paints the background green
1007    self.window2.configure(bg=form.green)
1008    #--Prints text
1009    self.correctAnswerText = Label(text=f' Correct! You have gained a mark. ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
1010    self.correctAnswerText.grid(padx = 300, pady = 300)
1011
1012    #--Increases the score by 1
1013    self.score += 1
1014
1015    #--Allows the next question to be displayed
1016    self.chooseQuestion += 1
1017    #--Displays this window for 2 seconds before displaying the next question
1018    self.window2.after(2000, lambda: self.DisplayQuiz())
1019    #--Removes this text widget so the next lot of questions can be displayed
1020    self.window2.after(2000, lambda: self.correctAnswerText.grid_forget())
```

For the IncorrectAnswer method, I have used if and elif statements to decide which question the user is on. This is so the program can display the correct answer to that question.

```
753  def IncorrectAnswer(self):
754    #--Removes all widgets in the window
755    self.QuestionNumber.grid_forget()
756    self.Question.grid_forget()
757    self.CorrectAnswer.grid_forget()
758    self.IncorrectAnswer1.grid_forget()
759    self.IncorrectAnswer2.grid_forget()
760    self.IncorrectAnswer3.grid_forget()
761    #--Paints the background red
762    self.window2.configure(bg=form.red)
763
764    #--First Question
765    if self.chooseQuestion == 0:
766      self.incorrectAnswerText = Label(text=f' Incorrect. The correct answer was: ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
767      self.correctAnswer = Label(text=f' {self.quizArray[1][2]}', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
768      self.incorrectAnswerText.grid(row = 1, column = 2, padx = 300, pady = 150)
769      self.correctAnswer.grid(row = 2, column = 2)
770
771      #--Allows the next question to be displayed
772      self.chooseQuestion+=1
773
774      #--Displays this window for 3 seconds before displaying the next question
775      self.window2.after(3000, lambda: self.DisplayQuiz())
776      #--Removes these text widgets so the next screen can be displayed
777      self.window2.after(3000, lambda: self.incorrectAnswerText.grid_forget())
778      self.window2.after(3000, lambda: self.correctAnswer.grid_forget())
779
780    #--Second Question
781    elif self.chooseQuestion == 1:
782      self.incorrectAnswerText = Label(text=f' Incorrect. The correct answer was: ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
783      self.correctAnswer = Label(text=f'   {self.quizArray[2][2]}   ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
784      self.incorrectAnswerText.grid(row = 1, column = 2, padx = 300, pady = 150)
785      self.correctAnswer.grid(row = 2, column = 2)
786
787      #--Allows the next question to be displayed
788      self.chooseQuestion+=1
789
790      #--Displays this window for 3 seconds before displaying the next question
791      self.window2.after(3000, lambda: self.DisplayQuiz())
792      #--Removes these text widgets so the next screen can be displayed
793      self.window2.after(3000, lambda: self.incorrectAnswerText.grid_forget())
794      self.window2.after(3000, lambda: self.correctAnswer.grid_forget())
```

Improving the readability of my code

When looking back at my code, I found that with 13 questions needing to be displayed, having 13 if statements that each display a different question would make the code very long and would reduce its readability. Instead of using if statements, I can use an attribute that iterates through the array to display every question.

I have set this attribute to 1 since this represents the first lot of questions and answers in the 2D array.

```
651        self.x = 1
```

I will only need these lines of code to display all the questions since I can iterate through the 2D array.

```
672    #--First Question
673    self.QuestionNumber = Label(text=f' Question {self.quizArray[self.x][0]}: ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
674    self.Question = Label(text=f' {self.quizArray[self.x][1]} ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
675    #--Correct Answer
676    self.CorrectAnswer = Button(text=f'  {self.quizArray[self.x][2]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
677                                                                                                    command = self.CorrectAnswer)
678    #--Incorrect Answers
679    self.IncorrectAnswer1 = Button(text=f'  {self.quizArray[self.x][3]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
680                                                                                                      command = self.IncorrectAnswer)
681    self.IncorrectAnswer2 = Button(text=f'  {self.quizArray[self.x][4]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
682                                                                                                      command = self.IncorrectAnswer)
683    self.IncorrectAnswer3 = Button(text=f'  {self.quizArray[self.x][5]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
684                                                                                                      command = self.IncorrectAnswer)
685    #--Determines positioning of widgets
686    self.QuestionNumber.grid(row = 1, column = 1, columnspan = 1,padx = 400, pady = 50)
687    self.Question.grid(row = 2, column = 1, columnspan = 1,pady = 70)
688    self.CorrectAnswer.grid(row = 3, column = 1, padx = 100,pady = 20)
689    self.IncorrectAnswer1.grid(row = 4, column = 1, padx = 100,pady = 20)
690    self.IncorrectAnswer2.grid(row = 5, column = 1, pady = 20)
691    self.IncorrectAnswer3.grid(row = 6, column = 1, pady = 20)
```

Since I cannot use the same 'CorrectAnswer' method for every question, I have used an if statement which allows the 'CorrectAnswer2' method to be run if the user gets the second question correct.

```
    #--Questions
    self.QuestionNumber = Label(text=f' Question {self.quizArray[self.x][0]}: ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
    self.Question = Label(text=f' {self.quizArray[self.x][1]} ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
    #--Correct answer for the first question
    if self.x == 1:
      self.CorrectAnswer = Button(text=f'  {self.quizArray[self.x][2]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
                                                                                                    command = self.CorrectAnswer)
    #--Correct answer for the second question
    if self.x == 2:
      self.CorrectAnswer = Button(text=f'  {self.quizArray[self.x][2]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
                                                                                                    command = self.CorrectAnswer2)
    #--Incorrect Answers
    self.IncorrectAnswer1 = Button(text=f'  {self.quizArray[self.x][3]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
                                                                                                    command = self.IncorrectAnswer)
    self.IncorrectAnswer2 = Button(text=f'  {self.quizArray[self.x][4]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
                                                                                                    command = self.IncorrectAnswer)
    self.IncorrectAnswer3 = Button(text=f'  {self.quizArray[self.x][5]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
                                                                                                    command = self.IncorrectAnswer)
    #--Determines positioning of widgets
    self.QuestionNumber.grid(row = 1, column = 1, columnspan = 1,padx = 400, pady = 50)
    self.Question.grid(row = 2, column = 1, columnspan = 1,pady = 70)
    self.CorrectAnswer.grid(row = 3, column = 1, padx = 100,pady = 20)
    self.IncorrectAnswer1.grid(row = 4, column = 1, padx = 100,pady = 20)
    self.IncorrectAnswer2.grid(row = 5, column = 1, pady = 20)
    self.IncorrectAnswer3.grid(row = 6, column = 1, pady = 20)
```

Since I am using this new attribute, I can change aspects of my IncorrectAnswer method. Previously, I used if statements to print the correct answer that the user should have answered. But I can remove the if statements since the self.x attribute can increment through the array instead.

```
  def IncorrectAnswer(self):
    #--Removes all widgets in the window
    self.QuestionNumber.grid_forget()
    self.Question.grid_forget()
    self.CorrectAnswer.grid_forget()
    self.IncorrectAnswer1.grid_forget()
    self.IncorrectAnswer2.grid_forget()
    self.IncorrectAnswer3.grid_forget()
    #--Paints the background red
    self.window2.configure(bg=form.quizred)

    #--Prints the correct answer to the first question
    self.incorrectAnswerText = Label(text=f' Incorrect. The correct answer was: ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
    self.correctAnswer = Label(text=f' {self.quizArray[self.x][2]}', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
    self.incorrectAnswerText.grid(row = 1, column = 2, padx = 300, pady = 150)
    self.correctAnswer.grid(row = 2, column = 2)
    #--Allows the next question to be displayed
    self.x +=1

    #--Displays this window for 3 seconds before moving onto the next question
    self.window2.after(3000, lambda: self.DisplayQuiz())
    #--Removes these text widgets so the next screen can be displayed
    self.window2.after(3000, lambda: self.incorrectAnswerText.grid_forget())
    self.window2.after(3000, lambda: self.correctAnswer.grid_forget())
```

Whenever the CorrectAnswer or IncorrectAnswer method is run, this attribute increments by 1 so the next question in the 2D array can be displayed.

```
        #--Allows the next question to be displayed
        self.x +=1
```

Randomising coordinates

Before I can progress and repeat this process for all 13 questions, I will need a way to randomise the order in which the possible answers are displayed. This is an important usability feature as it prevents the user from guessing the answers, especially when the quiz is repeated.

```
import random
```

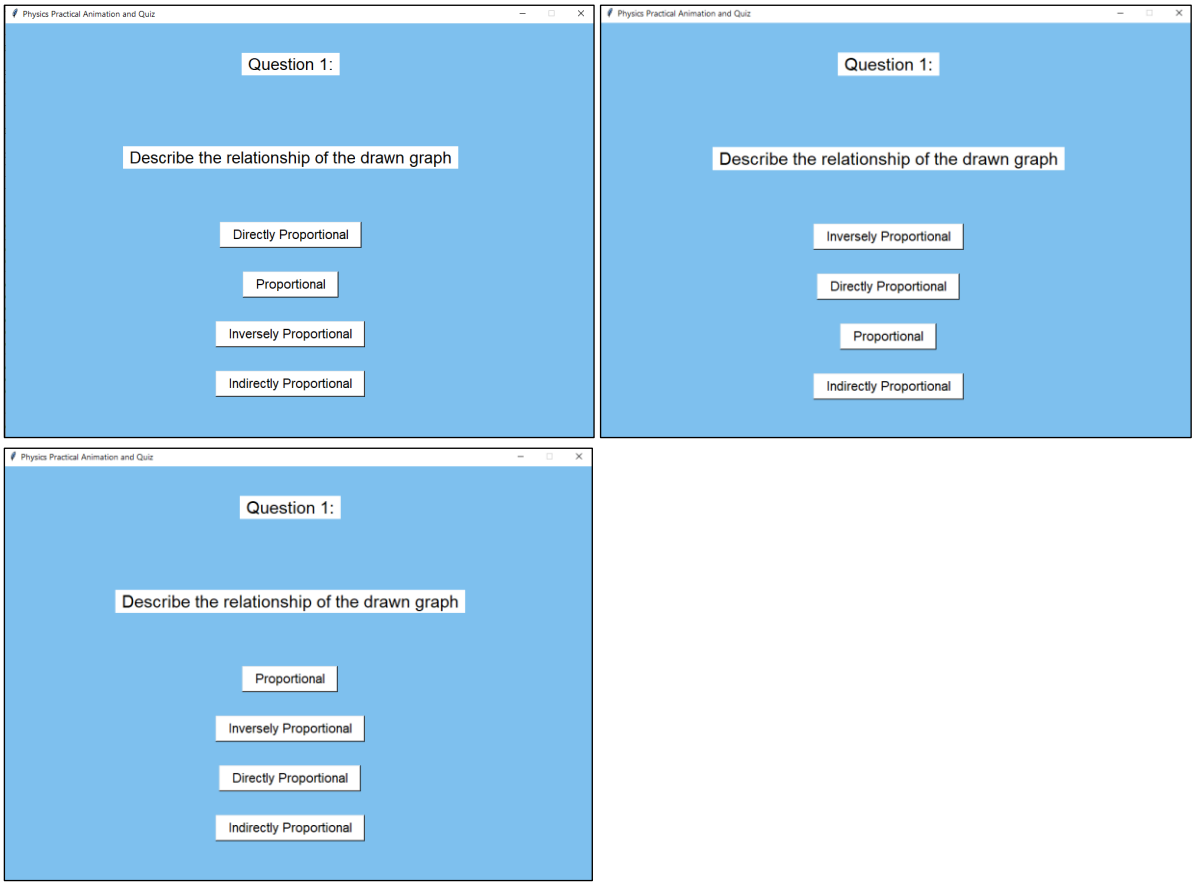
```
    #--Randomises the order of the displayed answers
    rowchoice = [3,4,5,6]
    row1 = random.choice(rowchoice)
    #--Removing items from the list so the same coordinates cannot be used twice
    rowchoice.remove(row1)
    row2 = random.choice(rowchoice)
    rowchoice.remove(row2)
    row3 = random.choice(rowchoice)
    rowchoice.remove(row3)
    row4 = random.choice(rowchoice)
    rowchoice.remove(row4)
```

I have created a list with all the possible line numbers that can be used, shown on line 692. On line 693, I have chosen a number at random from this list. So this number cannot be used twice, I have removed it from the list. The process of choosing and removing is repeated until line 701.

The variables which store the row numbers are used to display the answers in a random order. When the next question needs to be displayed, since the method will be repeated, so will this process of choosing from a list. Therefore, the answers will always be arranged in a random order.

```
706     self.CorrectAnswer.grid(row = row1, column = 1, padx = 100,pady = 20)
707     self.IncorrectAnswer1.grid(row = row2, column = 1, padx = 100,pady = 20)
708     self.IncorrectAnswer2.grid(row = row3, column = 1, pady = 20)
709     self.IncorrectAnswer3.grid(row = row4, column = 1, pady = 20)
```

Here are some examples of the resulting screens:



Displaying all 13 questions

For the rest of the questions that need to be displayed, I can repeat what I have done for the first two. I have included more if statements so that the next questions in the array can be displayed.

```
673  def DisplayQuiz(self):
674    #--Removes widgets in the window
675    self.quizMaintext.grid_forget()
676    self.startText.grid_forget()
677    self.startButton.grid_forget()
678    #--Repaints the background to the original colour
679    self.window2.configure(bg="SkyBlue2")
680
681    #--Questions
682    self.QuestionNumber = Label(text=f' Question {self.quizArray[self.x][0]}: ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
683    self.Question = Label(text=f' {self.quizArray[self.x][1]} ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLargeQuiz))
684
685    #--Correct answer for the first question
686    if self.x == 1:
687      self.CorrectAnswer = Button(text=f'  {self.quizArray[self.x][2]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
688                                                                                                    command = self.CorrectAnswer)
689    #--Correct answer for the second question
690    if self.x == 2:
691      self.CorrectAnswer = Button(text=f'  {self.quizArray[self.x][2]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
692                                                                                                    command = self.CorrectAnswer2)
693    #--Correct answer for the third question
694    if self.x == 3:
695      self.CorrectAnswer = Button(text=f'  {self.quizArray[self.x][2]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
696                                                                                                    command = self.CorrectAnswer3)
697    #--Correct answer for the fourth question
698    if self.x == 4:
699      self.CorrectAnswer = Button(text=f'  {self.quizArray[self.x][2]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
700                                                                                                    command = self.CorrectAnswer4)
701    #--Correct answer for the fifth question
702    if self.x == 5:
703      self.CorrectAnswer = Button(text=f'  {self.quizArray[self.x][2]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
704                                                                                                    command = self.CorrectAnswer5)
705    #--Correct answer for the sixth question
706    if self.x == 6:
707      self.CorrectAnswer = Button(text=f'  {self.quizArray[self.x][2]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
708                                                                                                    command = self.CorrectAnswer6)
709    #--Correct answer for the seventh question
710    if self.x == 7:
711      self.CorrectAnswer = Button(text=f'  {self.quizArray[self.x][2]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
712                                                                                                    command = self.CorrectAnswer7)
713    #--Correct answer for the eighth question
714    if self.x == 8:
715      self.CorrectAnswer = Button(text=f'  {self.quizArray[self.x][2]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
716                                                                                                    command = self.CorrectAnswer8)
717    #--Correct answer for the ninth question
718    if self.x == 9:
719      self.CorrectAnswer = Button(text=f'  {self.quizArray[self.x][2]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
720                                                                                                    command = self.CorrectAnswer9)
```

```
721    #--Correct answer for the tenth question
722    if self.x == 10:
723      self.CorrectAnswer = Button(text=f'  {self.quizArray[self.x][2]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
724                                                                                                    command = self.CorrectAnswer10)
725    #--Correct answer for the eleventh question
726    if self.x == 11:
727      self.CorrectAnswer = Button(text=f'  {self.quizArray[self.x][2]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
728                                                                                                    command = self.CorrectAnswer11)
729    #--Correct answer for the twelfth question
730    if self.x == 12:
731      self.CorrectAnswer = Button(text=f'  {self.quizArray[self.x][2]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
732                                                                                                    command = self.CorrectAnswer12)
733
734    #--Correct answer for the thirteenth question
735    if self.x == 13:
736      self.CorrectAnswer = Button(text=f'  {self.quizArray[self.x][2]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
737                                                                                                    command = self.CorrectAnswer13)
738    #--Incorrect Answers
739    self.IncorrectAnswer1 = Button(text=f'  {self.quizArray[self.x][3]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
740                                                                                                    command = self.IncorrectAnswer)
741    self.IncorrectAnswer2 = Button(text=f'  {self.quizArray[self.x][4]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
742                                                                                                    command = self.IncorrectAnswer)
743    self.IncorrectAnswer3 = Button(text=f'  {self.quizArray[self.x][5]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
744                                                                                                    command = self.IncorrectAnswer)
745    #--Randomises the order of the displayed answers
746    rowchoice = [3,4,5,6]
747    row1 = random.choice(rowchoice)
748    #--Removing items from the list so the same coordinates cannot be used twice
749    rowchoice.remove(row1)
750    row2 = random.choice(rowchoice)
751    rowchoice.remove(row2)
752    row3 = random.choice(rowchoice)
753    rowchoice.remove(row3)
754    row4 = random.choice(rowchoice)
755    rowchoice.remove(row4)
756
757    #--Determines positioning of widgets
758    self.QuestionNumber.grid(row = 1, column = 1, columnspan = 1,padx = 400, pady = 50)
759    self.Question.grid(row = 2, column = 1, columnspan = 1,pady = 70)
760    self.CorrectAnswer.grid(row = row1, column = 1, padx = 100,pady = 20)
761    self.IncorrectAnswer1.grid(row = row2, column = 1, padx = 100,pady = 20)
762    self.IncorrectAnswer2.grid(row = row3, column = 1, pady = 20)
763    self.IncorrectAnswer3.grid(row = row4, column = 1, pady = 20)
```

If the user answers the question correctly

For each question that will be displayed, there is a new CorrectAnswer method. There will be 13 of these methods as there are 13 questions. While these methods have different names, the code is identical. I wasn't able to reuse the same method for every question.

```
1043  #--If the first question is answered correctly
1044  def CorrectAnswer(self):
1045    #--Removes all widgets in the window
1046    self.QuestionNumber.grid_forget()
1047    self.Question.grid_forget()
1048    self.CorrectAnswer.grid_forget()
1049    self.IncorrectAnswer1.grid_forget()
1050    self.IncorrectAnswer2.grid_forget()
1051    self.IncorrectAnswer3.grid_forget()
1052    #--Paints the background green
1053    self.window2.configure(bg=form.green)
1054    #--Prints text
1055    self.correctAnswerText = Label(text=f' Correct! You have gained a mark. ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
1056    self.correctAnswerText.grid(padx = 300, pady = 300)
1057
1058    #--Increases the score by 1
1059    self.score += 1
1060    #--Allows the next question to be displayed
1061    self.x +=12
1062    #--Displays this window for 2 seconds before displaying the next question
1063    self.window2.after(2000, lambda: self.DisplayQuiz())
1064    #--Removes this text widget so the next lot of questions and answers can be displayed
1065    self.window2.after(2000, lambda: self.correctAnswerText.grid_forget())
```

If the user answers the question incorrectly

As the user progresses through the questions, the self.x attribute will increase. This means that the correct answer that the user should have pressed will always be relevant to the current question that the user is on.

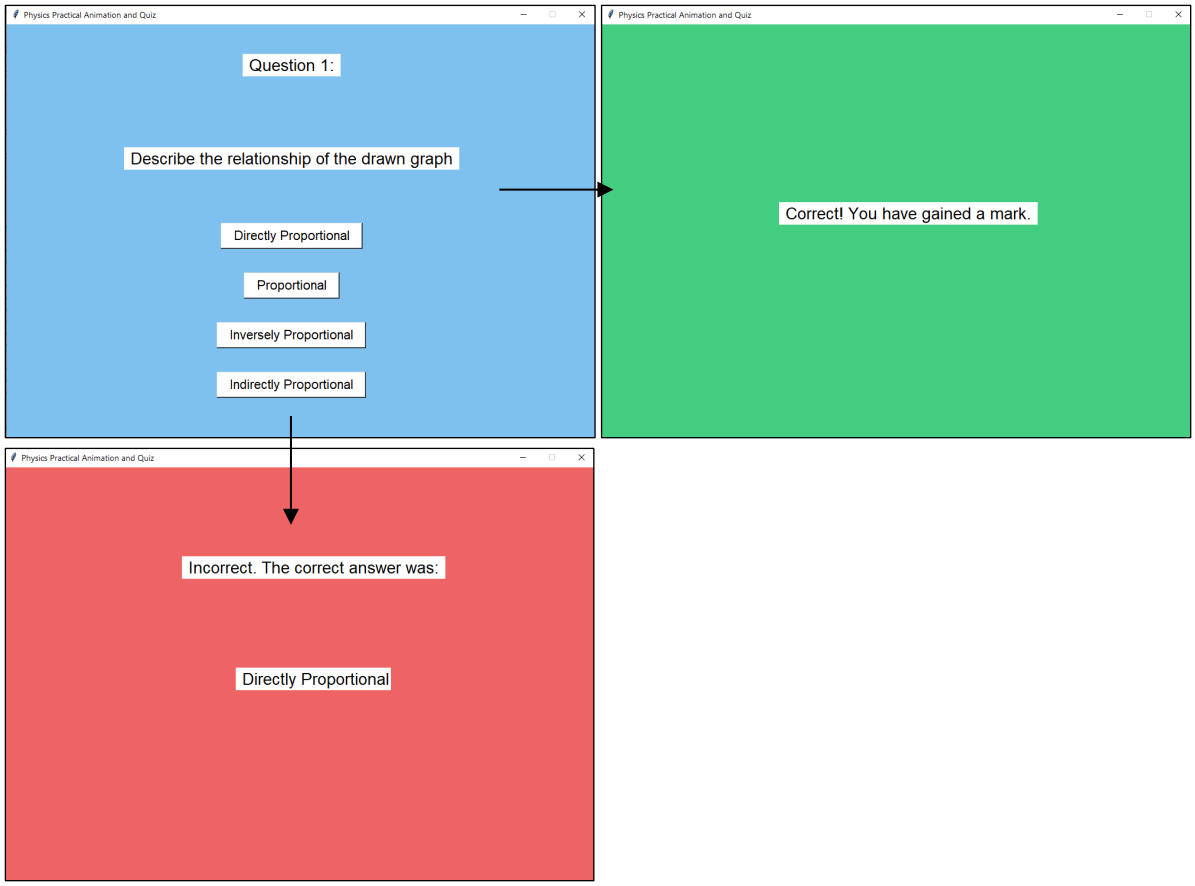
```
1356  def IncorrectAnswer(self):
1357    #--Removes all widgets in the window
1358    self.QuestionNumber.grid_forget()
1359    self.Question.grid_forget()
1360    self.CorrectAnswer.grid_forget()
1361    self.IncorrectAnswer1.grid_forget()
1362    self.IncorrectAnswer2.grid_forget()
1363    self.IncorrectAnswer3.grid_forget()
1364    #--Paints the background red
1365    self.window2.configure(bg=form.quizred)
1366
1367    #--Prints the correct answer to the first question
1368    self.incorrectAnswerText = Label(text=f' Incorrect. The correct answer was: ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
1369    self.correctAnswer = Label(text=f' {self.quizArray[self.x][2]}', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
1370    self.incorrectAnswerText.grid(row = 1, column = 2, padx = 300, pady = 150)
1371    self.correctAnswer.grid(row = 2, column = 2)
1372    #--Allows the next question to be displayed
1373    self.x +=1
1374
1375    #--Displays this window for 3 seconds before moving onto the next question
1376    self.window2.after(3000, lambda: self.DisplayQuiz())
1377    #--Removes these text widgets so the next screen can be displayed
1378    self.window2.after(3000, lambda: self.incorrectAnswerText.grid_forget())
1379    self.window2.after(3000, lambda: self.correctAnswer.grid_forget())
```

Development Testing **Test 5.2, 5.3, Test 5.4**

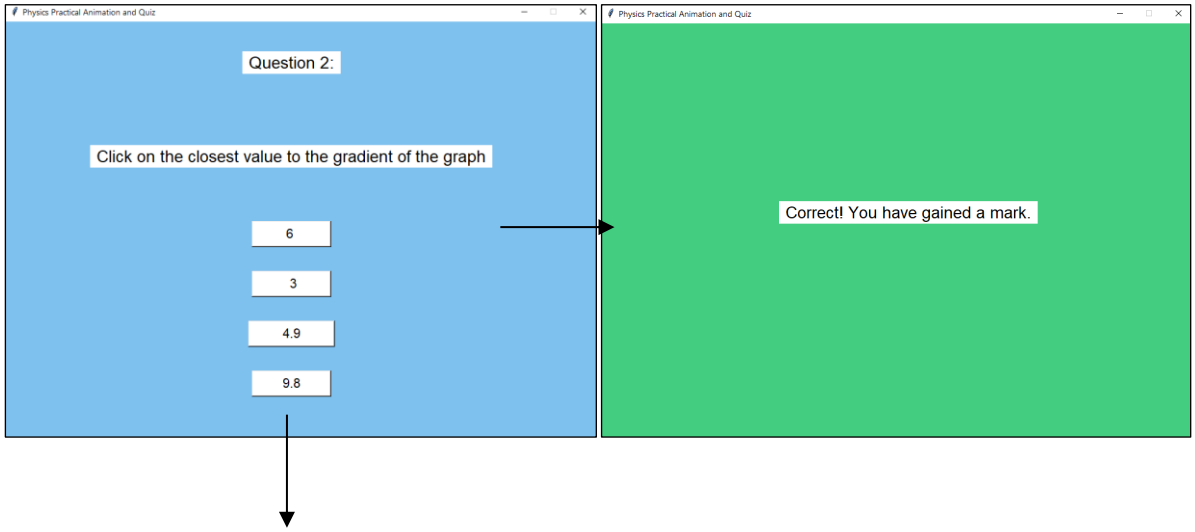
Displaying questions from CSV file	Valid input – button to start the quiz is pressed	Displays the questions	So the user can attempt the quiz	5.2
Answer buttons (Will consist of 4 multiple choice answers)	Valid input – button is pressed	If the answer is correct, the user will gain a mark and the next question should be displayed	To check if the student's answer is correct and to allow them to move onto the next question	5.3
		If the answer is incorrect, the user will get to see the correct answer. Then the next question will be displayed.	To check if the student's answer is incorrect and to allow them to move onto the next question	5.4

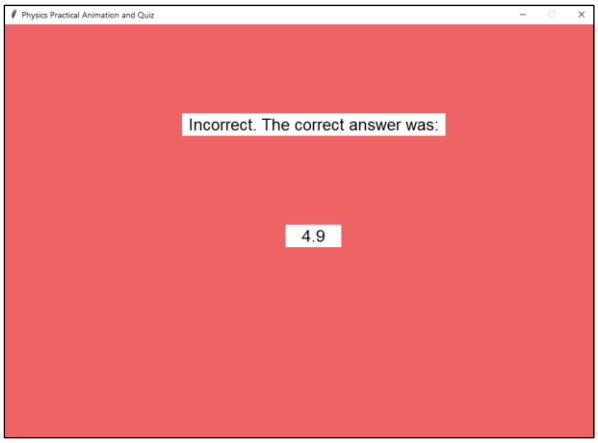
I will now test that the user can do the quiz for all 13 questions

Question 1:

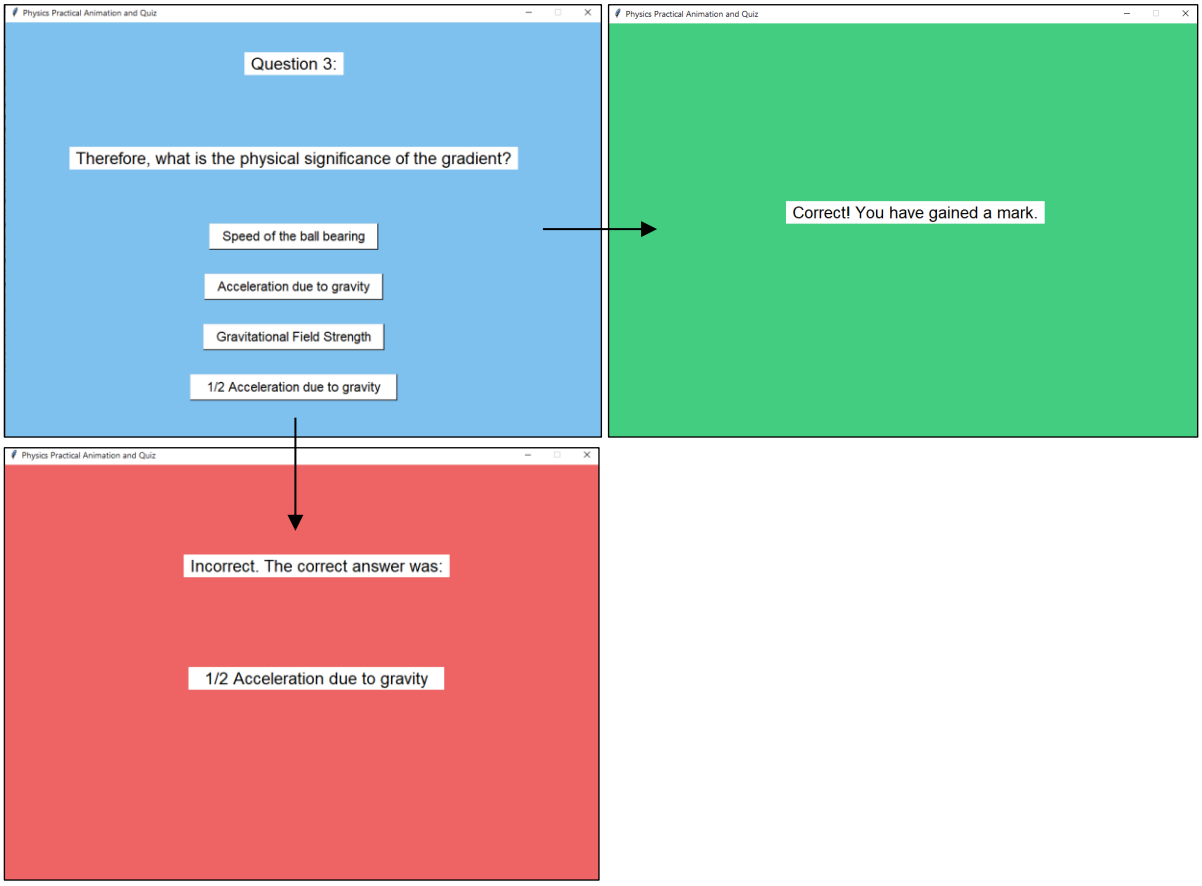


Question 2:

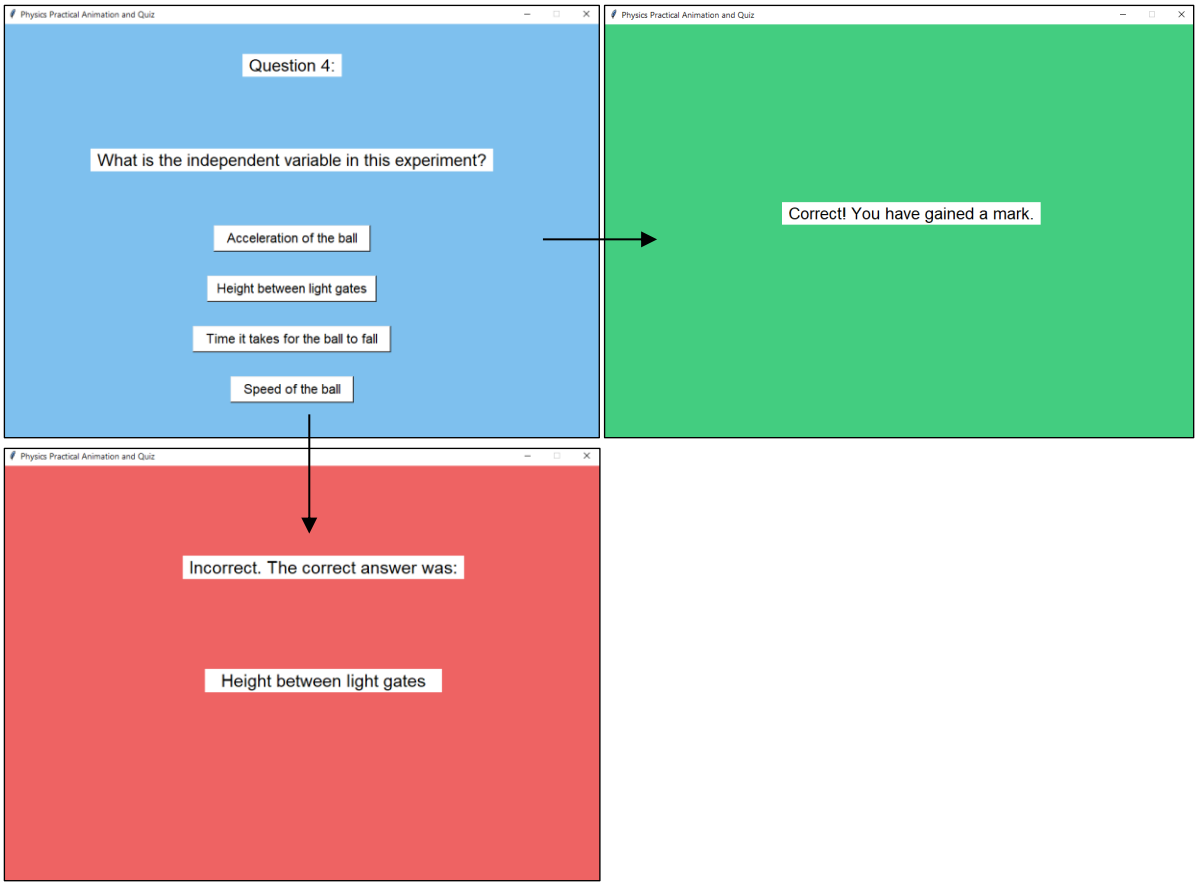




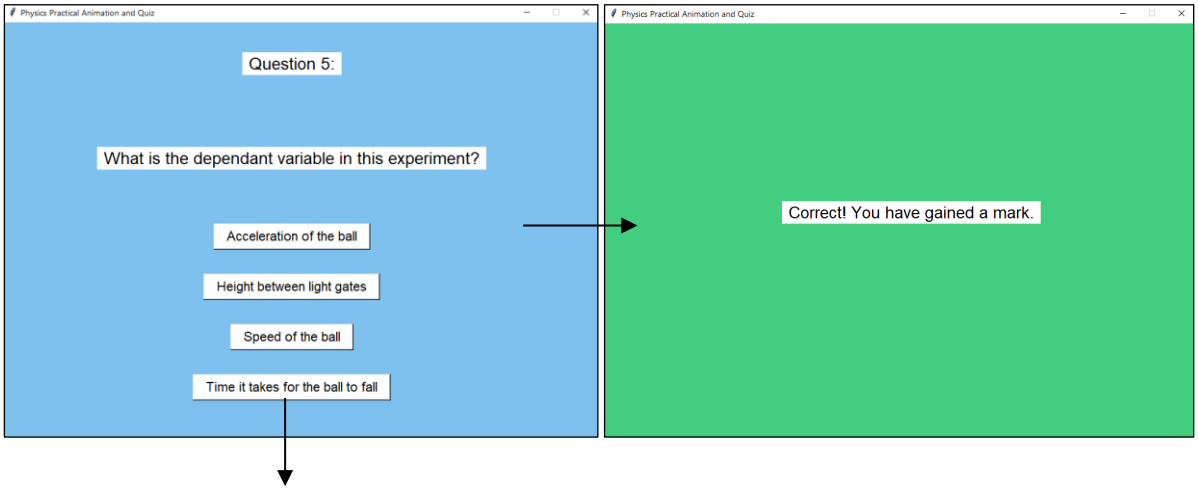
Question 3:

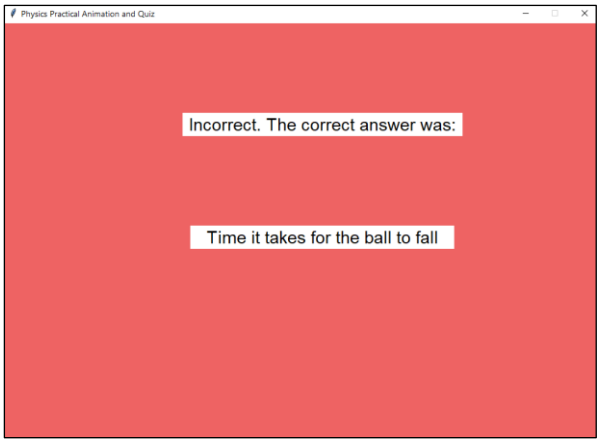


Question 4:

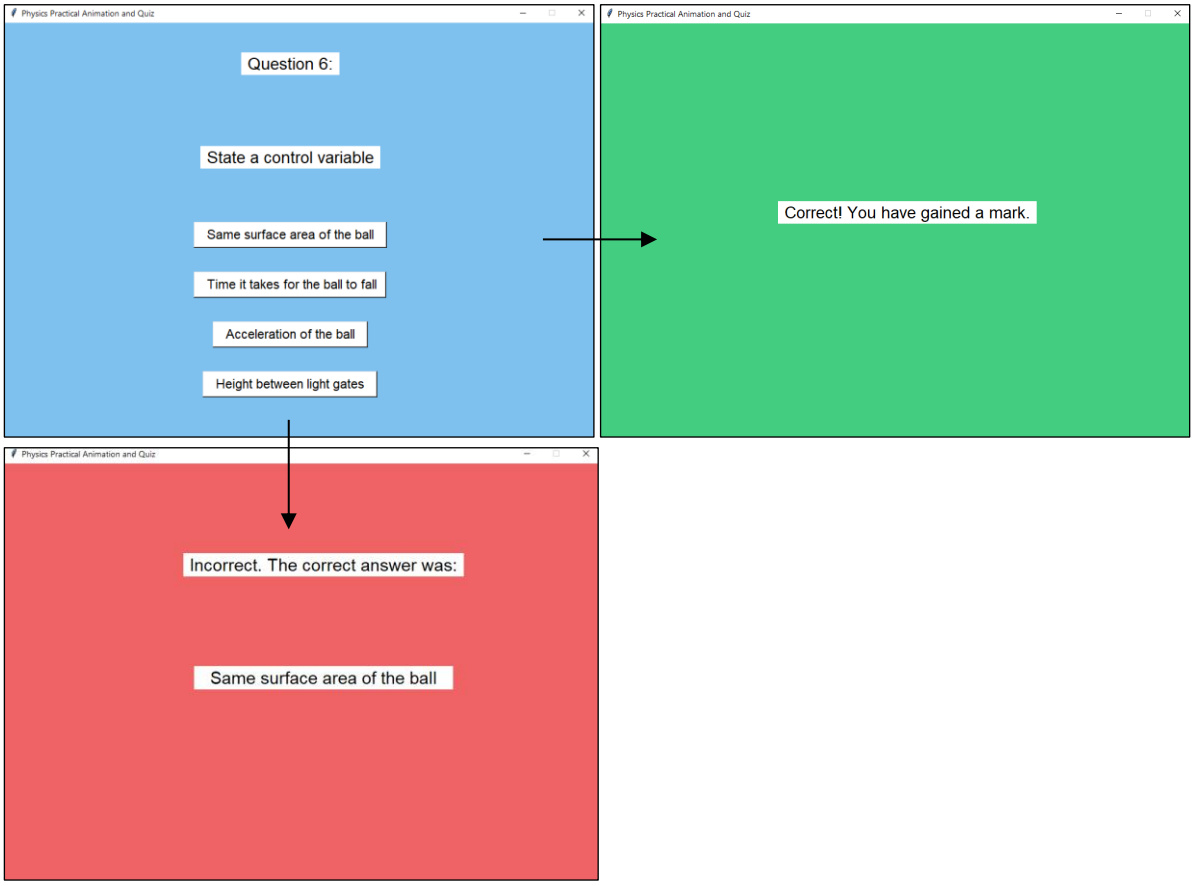

Physics Practical Animation and Quiz
Question 4:
What is the independent variable in this experiment?
Acceleration of the ball
Height between light gates
Time it takes for the ball to fall
Speed of the ball
Physics Practical Animation and Quiz
Correct! You have gained a mark.
Physics Practical Animation and Quiz
Incorrect. The correct answer was:
Height between light gates

Question 5:

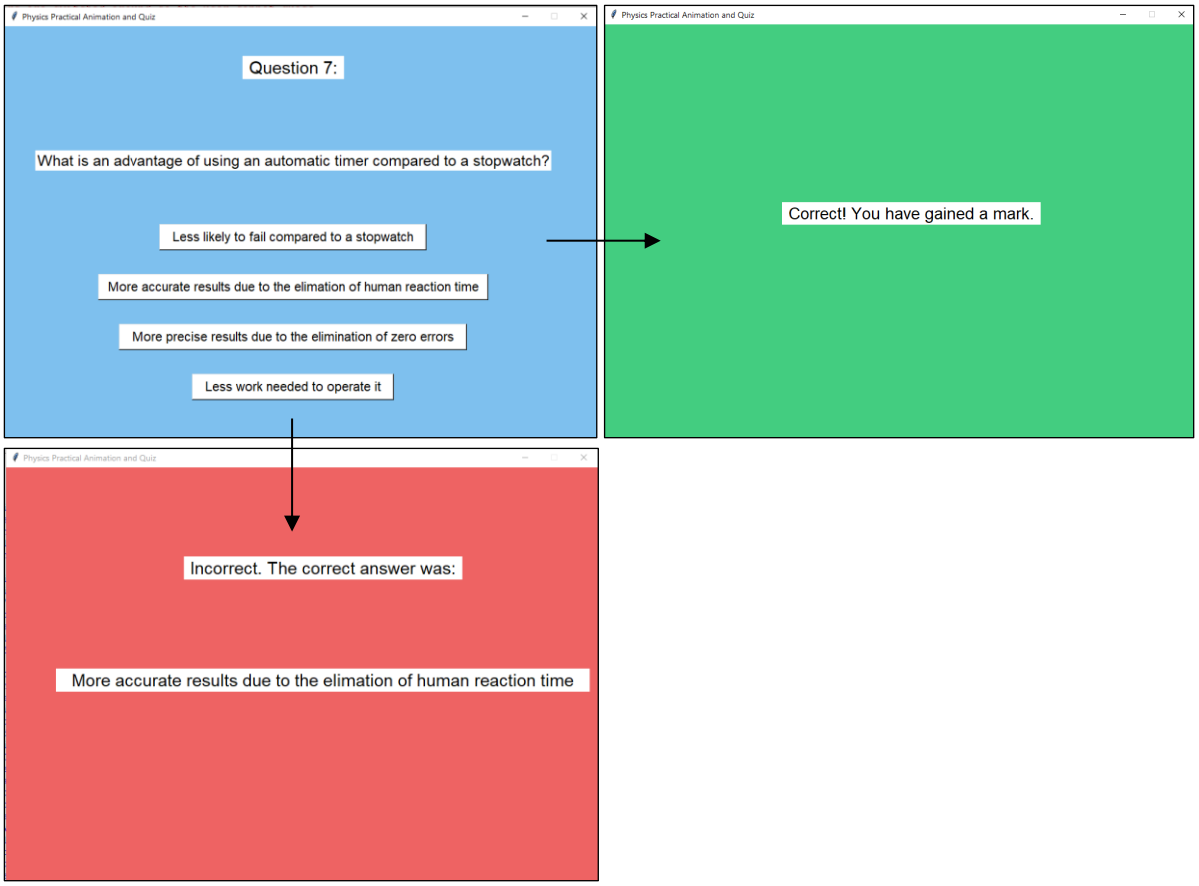

Physics Practical Animation and Quiz
Question 5:
What is the dependant variable in this experiment?
Acceleration of the ball
Height between light gates
Speed of the ball
Time it takes for the ball to fall
Physics Practical Animation and Quiz
Correct! You have gained a mark.



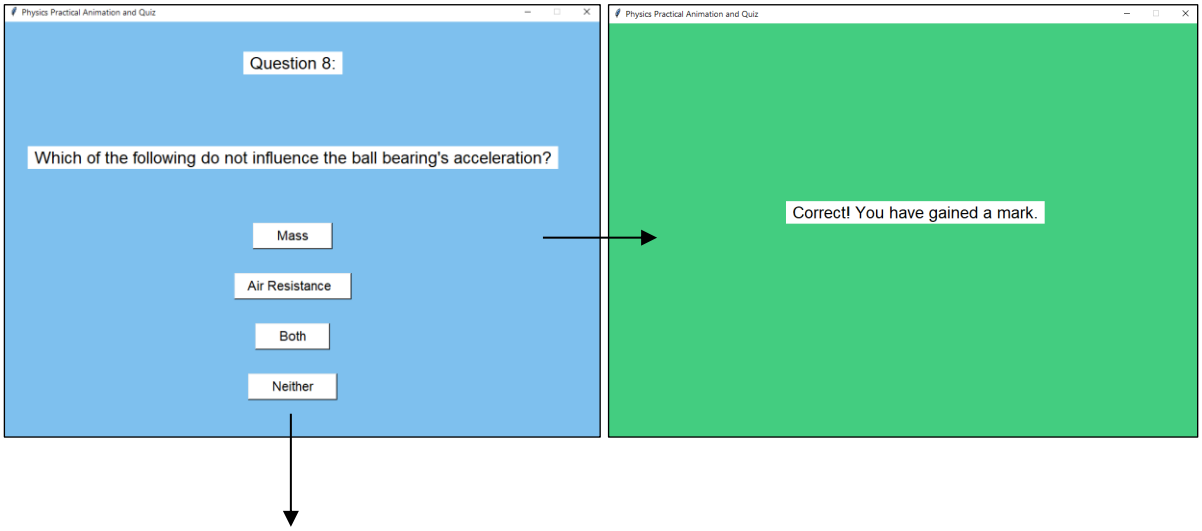
Question 6:

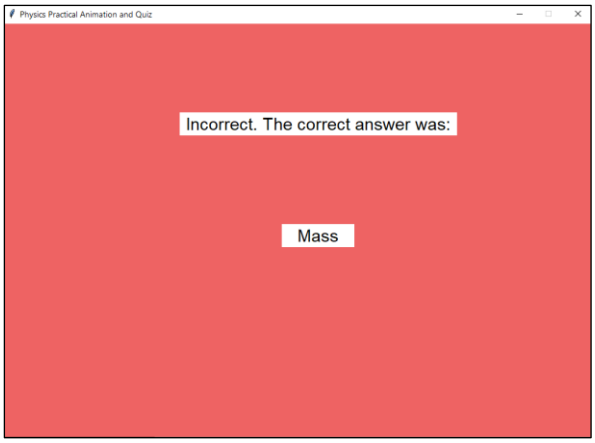


Question 7:

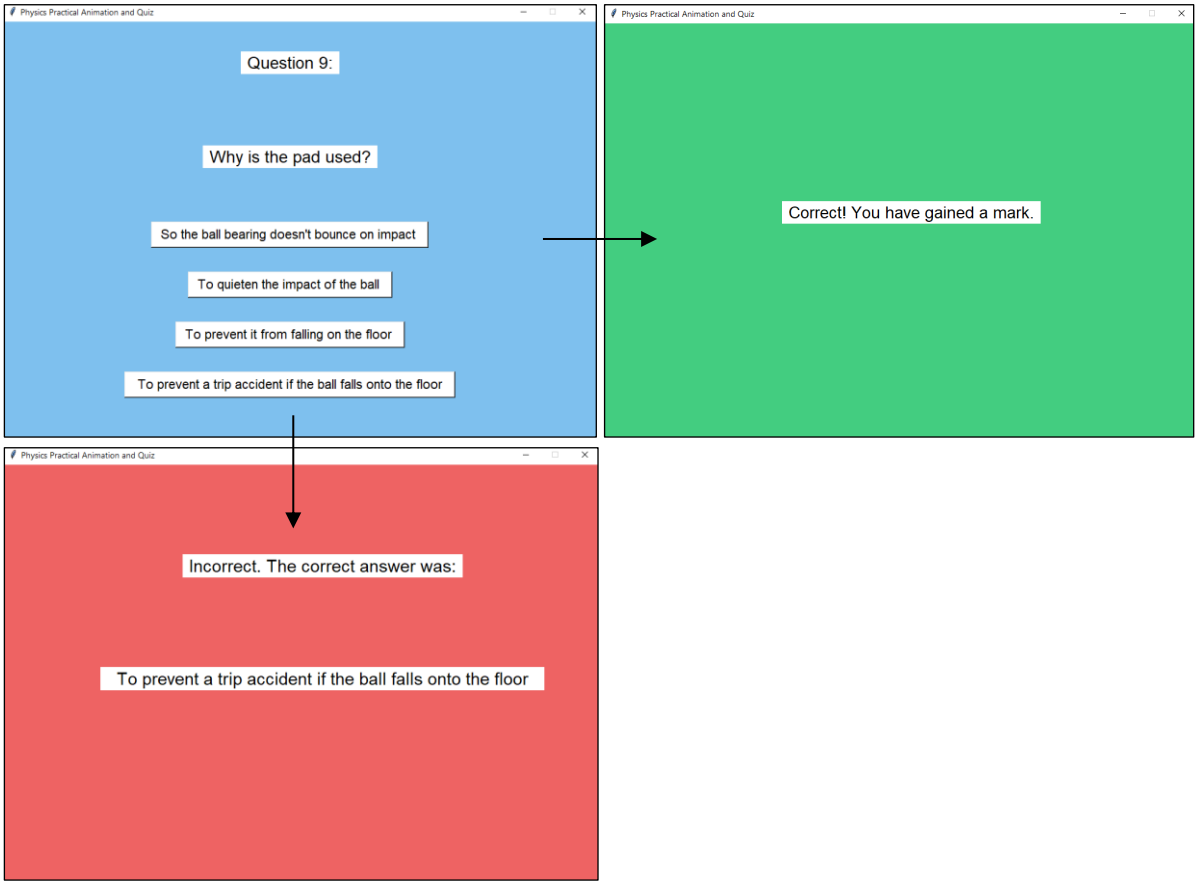

Physics Practical Animation and Quiz
Question 7:
What is an advantage of using an automatic timer compared to a stopwatch?
Less likely to fail compared to a stopwatch
More accurate results due to the elimation of human reaction time
More precise results due to the elimination of zero errors
Less work needed to operate it
Physics Practical Animation and Quiz
Correct! You have gained a mark.
Physics Practical Animation and Quiz
Incorrect. The correct answer was:
More accurate results due to the elimation of human reaction time

Question 8:

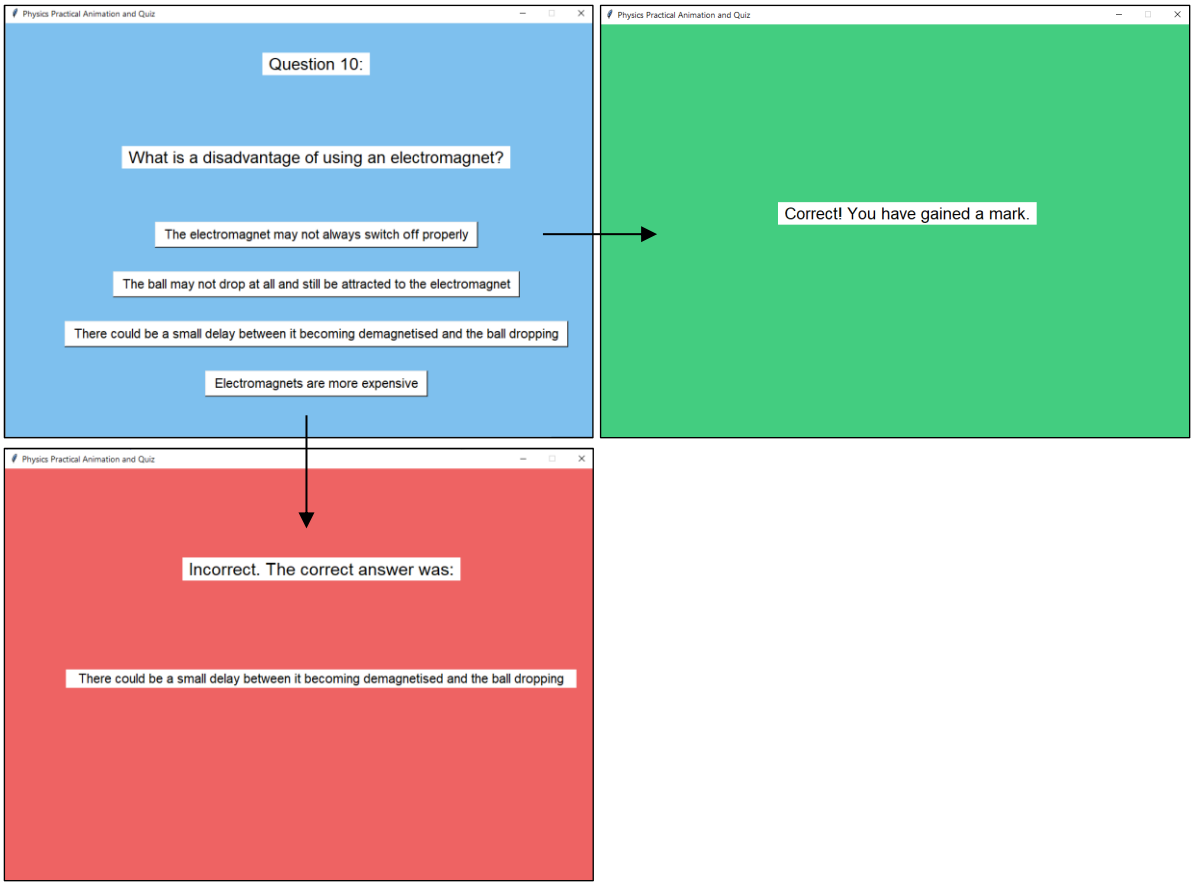

Physics Practical Animation and Quiz
Question 8:
Which of the following do not influence the ball bearing's acceleration?
Mass
Air Resistance
Both
Neither
Physics Practical Animation and Quiz
Correct! You have gained a mark.



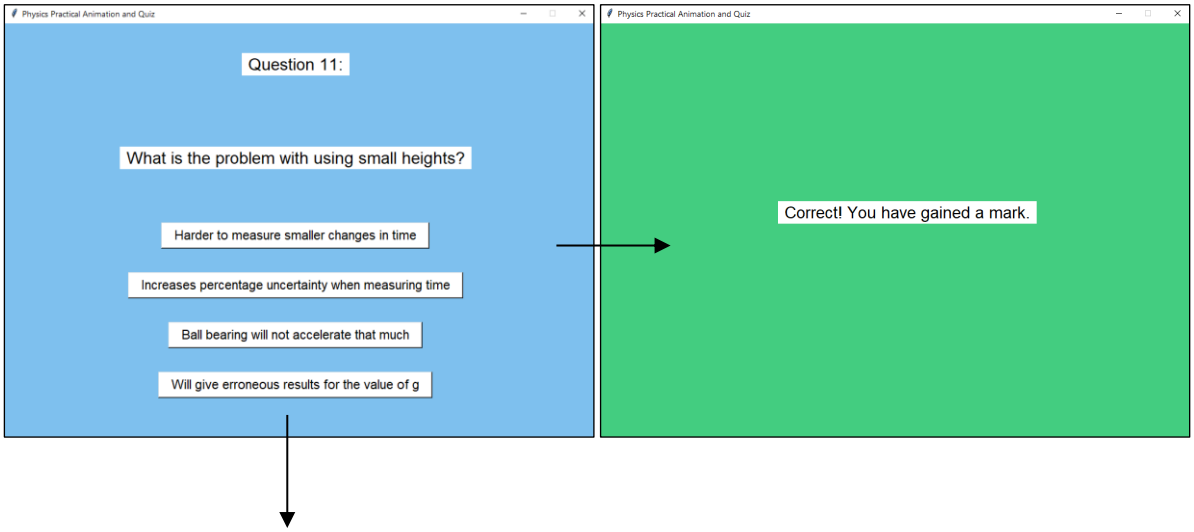
Question 9:

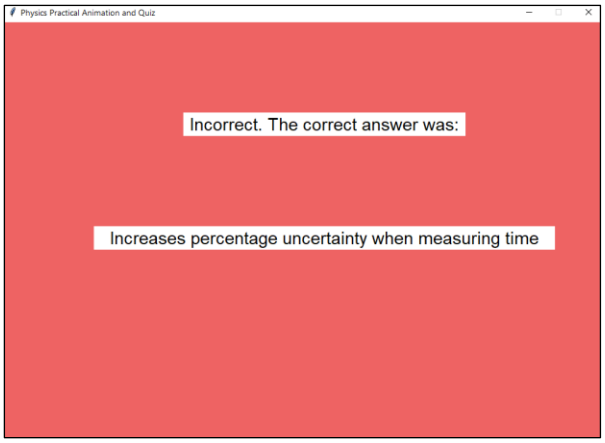


Question 10:

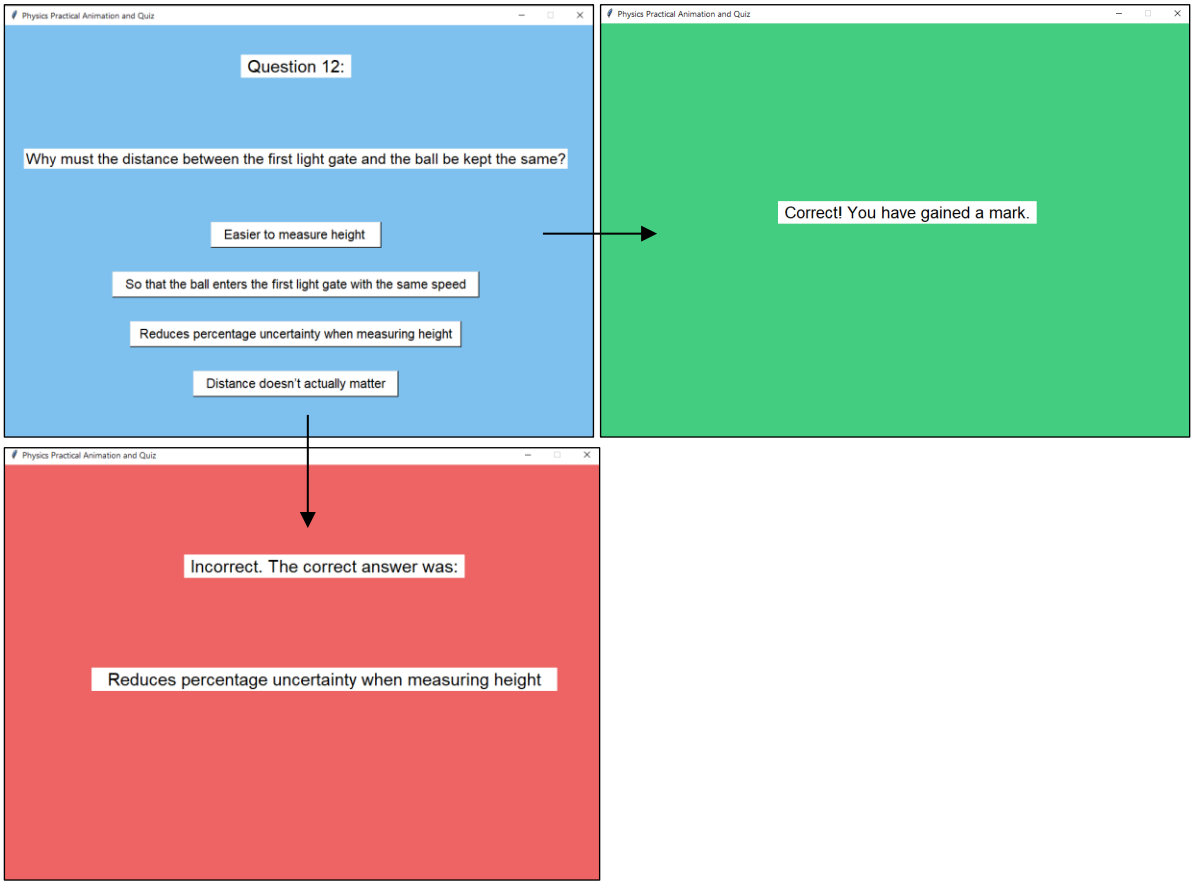

Physics Practical Animation and Quiz
Question 10:
What is a disadvantage of using an electromagnet?
The electromagnet may not always switch off properly
The ball may not drop at all and still be attracted to the electromagnet
There could be a small delay between it becoming demagnetised and the ball dropping
Electromagnets are more expensive
Physics Practical Animation and Quiz
Correct! You have gained a mark.
Physics Practical Animation and Quiz
Incorrect. The correct answer was:
There could be a small delay between it becoming demagnetised and the ball dropping

Question 11:

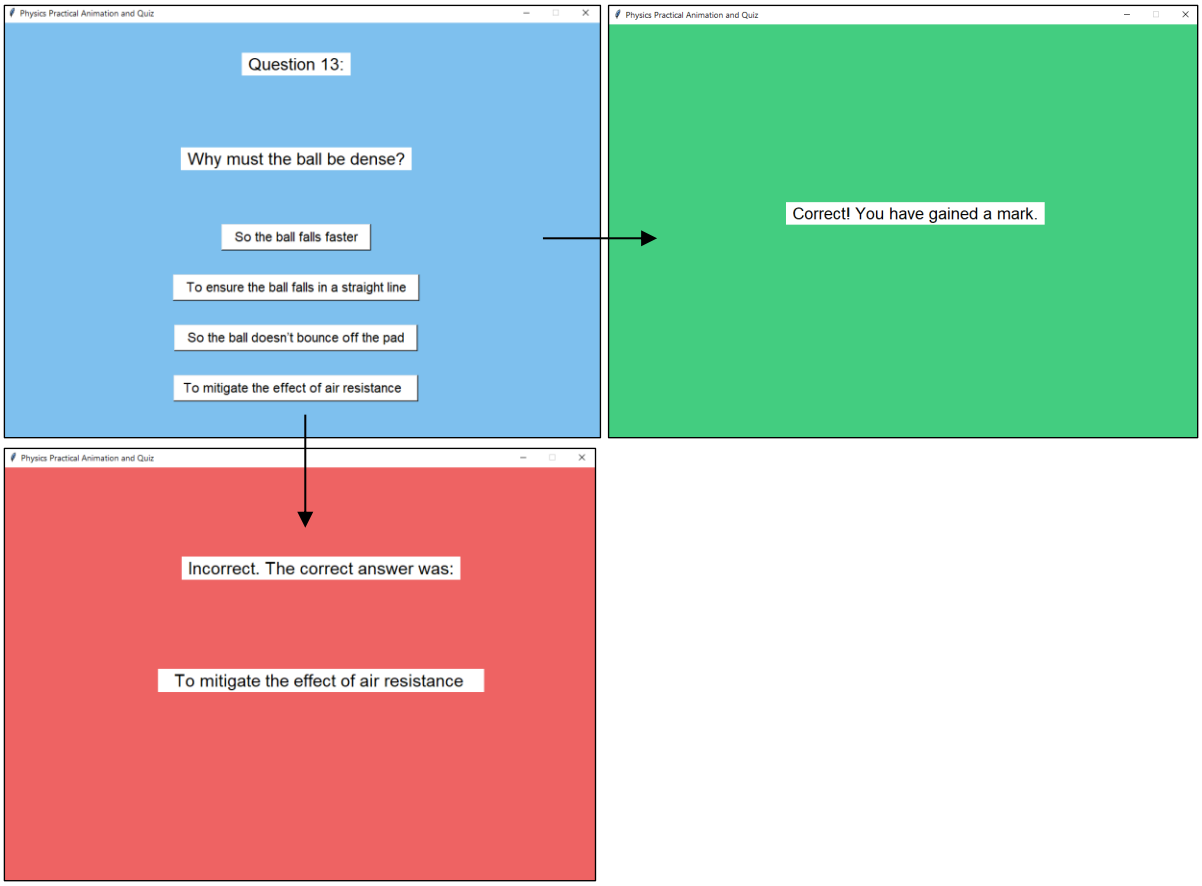

Physics Practical Animation and Quiz
Question 11:
What is the problem with using small heights?
Harder to measure smaller changes in time
Increases percentage uncertainty when measuring time
Ball bearing will not accelerate that much
Will give erroneous results for the value of g
Physics Practical Animation and Quiz
Correct! You have gained a mark.



Question 12:



Question 13



Tests 5.2, 5.3 and 5.4 are a success.

Last question in the quiz

The last question in the quiz will be slightly different from the rest. When this question is answered, the quiz will have to end, transitioning the user to a new screen.

If the user gets the last question correct or incorrect, the EndQuiz method will be called.

```
1333    #--If the thirteenth question is answered correctly
1334    def CorrectAnswer13(self):
1335      #--Removes all widgets in the window
1336      self.QuestionNumber.grid_forget()
1337      self.Question.grid_forget()
1338      self.CorrectAnswer.grid_forget()
1339      self.IncorrectAnswer1.grid_forget()
1340      self.IncorrectAnswer2.grid_forget()
1341      self.IncorrectAnswer3.grid_forget()
1342      #--Paints the background green
1343      self.window2.configure(bg=form.green)
1344      #--Prints text
1345      self.correctAnswerText = Label(text=f' Correct! You have gained a mark. ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
1346      self.correctAnswerText.grid(padx = 300, pady = 300)
1347
1348      #--Increases the score by 1
1349      self.score += 1
1350      #--Displays this window for 2 seconds before finishing the quiz
1351      self.window2.after(2000, lambda: self.EndQuiz())
1352      #--Removes this text widget so the next screen can be displayed
1353      self.window2.after(2000, lambda: self.correctAnswerText.grid_forget())
```

```
1666  #--If the user answers the question incorrectly
1667  def IncorrectAnswer(self):
1668    #--Removes all widgets in the window
1669    self.QuestionNumber.grid_forget()
1670    self.Question.grid_forget()
1671    self.CorrectAnswer.grid_forget()
1672    self.IncorrectAnswer1.grid_forget()
1673    self.IncorrectAnswer2.grid_forget()
1674    self.IncorrectAnswer3.grid_forget()
1675    #--Paints the background red
1676    self.window2.configure(bg=form.quizred)
1677
1678    #--Prints the correct answer to the first question
1679    print(f'Your score is {self.score}')
1680    self.incorrectAnswerText = Label(text=f' Incorrect. The correct answer was: ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
1681    self.correctAnswer = Label(text=f' {self.quizArray[self.x][2]}', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
1682    self.incorrectAnswerText.grid(row = 1, column = 2, padx = 300, pady = 150)
1683    self.correctAnswer.grid(row = 2, column = 2)
1684
1685    #If the user is on the final question
1686    if self.x == 13:
1687      #--Displays this window for 3 seconds before finishing the quiz
1688      self.window2.after(3000, lambda: self.EndQuiz())
1689
1690    #--Allows the next question to be displayed
1691    self.x +=1
1692    #--Displays this window for 3 seconds before moving onto the next question
1693    self.window2.after(3000, lambda: self.DisplayQuiz())
1694    #--Removes these text widgets so the next screen can be displayed
1695    self.window2.after(3000, lambda: self.incorrectAnswerText.grid_forget())
1696    self.window2.after(3000, lambda: self.correctAnswer.grid_forget())
```

FINISHING THE QUIZ

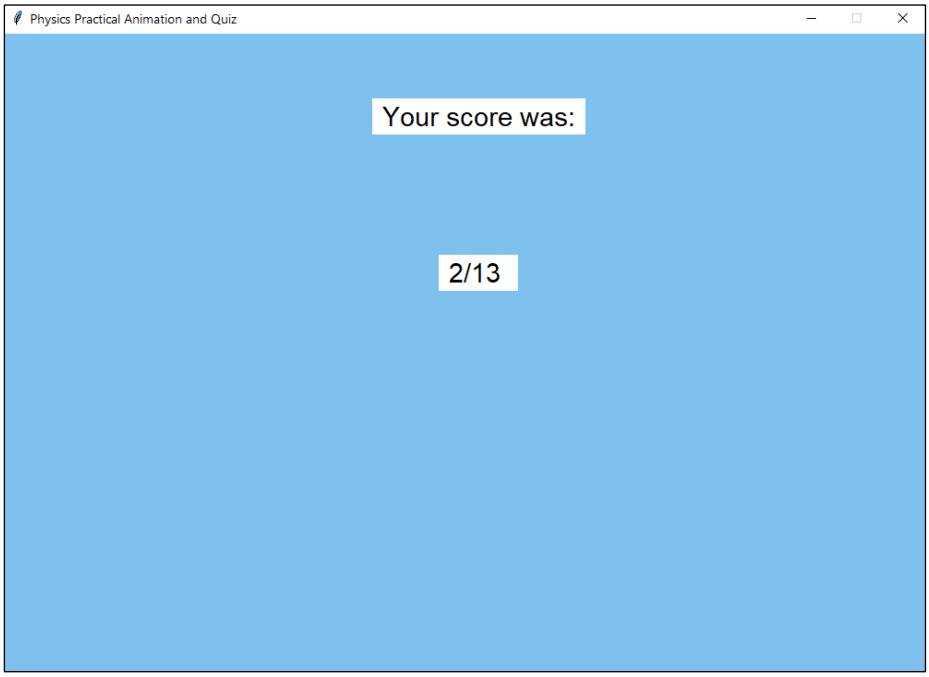
When the user answers the final question, I will transition them to a new screen where they can view the score that they got. Moreover, they can view the graph of their progress over time and repeat the quiz if they want to.

Since I have removed the widgets previously, all I have to do is to repaint the background.

```
1279    def EndQuiz(self):
1280      #--Repaints the window
1281      self.window2.configure(bg="SkyBlue2")
```

First, I have displayed the user's score to the screen

```
1285      #--Displays the user's score
1286      self.scoreText = Label(text=f' Your score was: ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
1287      self.scoreInteger = Label(text=f' {self.score}/13  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
1288      #--Positions widgets
1289      self.scoreText.grid(row = 1, column = 2, padx = 400, pady = 50)
1290      self.scoreInteger.grid(row = 2, column = 2, pady = 150)
```



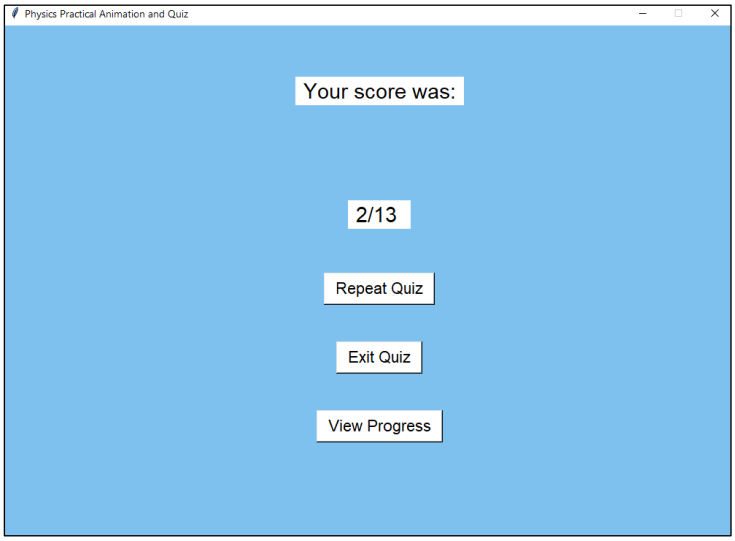
Next, I have laid out three buttons. Once the quiz has finished, the user has three options: repeating the quiz, exiting the quiz or viewing their progress on a graph. When either button is pressed, a new method will be called.

```
1300    #--Buttons that the user can press
1301    self.repeatQuiz = Button(text=f' Repeat Quiz ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium), command = self.RepeatQuiz)
1302    self.exitQuiz = Button(text=f' Exit Quiz ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium), command = self.ExitQuiz)
1303    self.graphProgress = Button(text=f' View Progress ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium), command = self.GraphProgress)
1304    #--Positions widgets
1305    self.repeatQuiz.grid(row = 4, column = 2, pady = 0)
1306    self.exitQuiz.grid(row = 5, column = 2, pady = 50)
1307    self.graphProgress.grid(row = 6, column = 2)
```

Development Testing

Answer buttons (Will consist of 4 multiple choice answers)	Valid input – button is pressed and the student is on the last question	Will give a score and allow the student to repeat the quiz or see their progress on a graph	To allow the student to view their score and progress	5.5

When the student finishes the quiz, they should be able to view their score and have the option to view their progress or repeat the practical.




```
Quiz finished
```

```
1722    #--Will run when the user answers the last question
1723    def EndQuiz(self):
1724      print("Quiz finished")
```

This test was a success.

Repeating the quiz

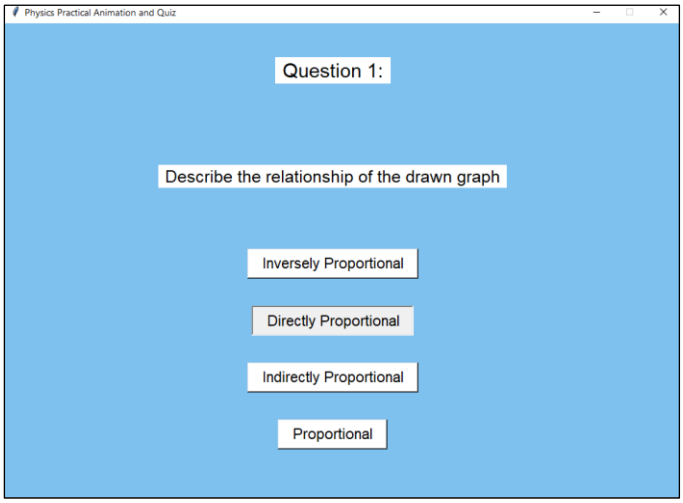
I will clear the contents of the quizArray so that the questions can be appended to it again. I have also reset the score to 0. On line 1323, the LoadQuestions method is repeated so that the questions can be appended from the Excel file to the array and the quiz can start again.

```
1722    #--Allows the user to repeat the quiz
1723    def RepeatQuiz(self):
1724        #--Removes widgets from the display
1725        self.repeatQuiz.grid_forget()
1726        self.exitQuiz.grid_forget()
1727        self.graphProgress.grid_forget()
1728        self.scoreText.grid_forget()
1729        self.scoreInteger.grid_forget()
1730
1731        #--Clears the contents of the array so the quiz can be repeated
1732        self.quizArray.clear()
1733
1734        #--Resets the score
1735        self.score = 0
1736
1737        print("Quiz repeated")
1738        #--Repeats the quiz
1739        command = self.LoadQuestions()
```

Development Testing **Test 5.8**

Repeat Button	Valid input – button is pressed	Will repeat the quiz	If the student wants another attempt at the quiz	5.8

When clicking the repeat button, the screen for the first question was displayed properly. However, when choosing the correct answer, the program did not register anything so the user could not progress. When clicking the incorrect answer, however, the screen changed successfully.




```
Quiz repeated
First question displayed
```

To fix this, I have made a new attribute which determines whether the quiz has been repeated or not.

```
1241        self.repeatAnswer = 0
```

If the quiz has been repeated, this attribute will increment by 1

```
1251        self.repeatAnswer +=1
```

The program will recognise whether the quiz has been repeated and the CorrectAnswer2 method can be run instead.

```
1271    #--This solves my problem where the I couldn't click the correct answers when repeating the quiz
1272    if self.repeatAnswer > 1 and self.x == 1:
1273      self.CorrectAnswer = Button(text=f'  {self.quizArray[self.x][2]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
1274                                                                                                    command = self.CorrectAnswer2)
```

Testing for robustness

I will now test how robust my program is by repeating the quiz multiple times.

```
Quiz finished
Quiz repeated
Quiz finished
Quiz repeated
Quiz finished
Quiz repeated
Quiz finished
Quiz repeated
Quiz finished
Quiz repeated
Quiz finished
Quiz repeated
Quiz finished
Quiz repeated
Quiz finished
Quiz repeated
Quiz finished
Quiz repeated
Quiz finished
Quiz repeated
Quiz finished
Quiz repeated
Quiz finished
Quiz repeated
Quiz finished
Quiz repeated
```

Every time the quiz was repeated, I ran into no errors and the questions and answers were displayed as shown in my screenshots. Therefore, test 5.8 was a success.

Validation

I have validated the user's input. Shown below, the program coped with all types of inputs.

Result	Valid input – Clicking the repeat button	Invalid input – Clicking anything that isn't the repeat button	Boundary input – Repeatedly clicking or holding the repeat button
Expected output?	Yes, the quiz was repeated	Yes, the quiz didn't repeat	Yes, the quiz was repeated.

Exiting the quiz

After the quiz has finished, I will give the user the option to exit the quiz.

```
1740    #--Will exit the program
1741    def ExitQuiz(self):
1742      self.window2.destroy()
1743      print("Program exited")
```

Development Testing

Final Exit button	Valid input – button is pressed	Returns the user to the login screen	So the user can properly exit the program	8.0

When pressing the exit button, the program window closed correctly.

```
Quiz finished
Program exited
```

However, in my success criteria, I wanted the user to be able to return to the login screen. I will not have time to implement this. Additionally, it is made harder by the fact that the window for the login screen has been destroyed.

Therefore, test 8.0 was a partial success since the user could quit the program but it does not fulfil the requirements of my success criteria.

Validation

I have validated the user's input. Shown below, the program coped with all types of inputs.

Result	Valid input – Clicking the exit button	Invalid input – Clicking anything that isn't the exit button	Boundary input – Repeatedly clicking or holding the exit button
Expected output?	Yes, the program was exited	Yes, the program wasn't exited	Yes, the quiz was exited

GRAPHING THE USER'S PROGRESS

When the user repeats the quiz more than once, I will allow them to see their progress over these attempts by plotting their score against the time when they completed the quiz. If I had more time, I would implement a system where the user could view their progress over a longer period, such as over a week or a month.

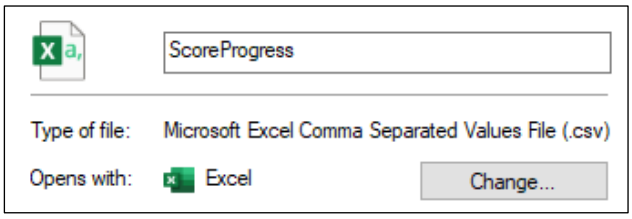
To start with, I have created a new method called GraphProgress. I have imported the current time from Pygame. This will form the basis for the x-axis of the graph.

```
1913    #--Allows the user to view their progress over time
1914    def GraphProgress(self):
1915      #--Allows program to record the current time
1916      import datetime
1917      from datetime import date
1918
1919      #--Records the current time
1920      t = time.localtime()
1921      current_time = time.strftime("%H:%M:%S", t)
1922
1923      #--List of times and scores
1924      self.progressList = []
```

The below piece of code appends the time and student's score to a CSV file so that it is always saved. When the graph needs to be displayed, on line 1933, I have opened the file to read its contents to a 2D array.

```
1926      #--Appends the current time and user's score to a csv file
1927      rows = [[current_time, self.score]]
1928      with open('ScoreProgress.csv','a', newline='') as ScoreProgress:
1929              writer = csv.writer(ScoreProgress)
1930              writer.writerows(rows) #--Rows written
1931
1932      #--Writes contents of this file to a list
1933      with open('ScoreProgress.csv','r', newline='') as ScoreProgress:
1934              reader = csv.reader(ScoreProgress)
1935              for row in reader: #--File contents are appended as a 2d array/list
1936                self.progressList.append(row)
```

Below, is the file and its contents:



	A	B	C
1	19:00:27	2	
2	19:22:29	3	
3	19:45:41	6	
4			

I have deleted the first item in the 2D array since it will be blank. For some reason, Excel skips the first row when appending items to it.

```
1775      del self.progressList[0] #--Deletes first item in list since this will be blank
```

I have created a new attribute which is equal to the length of the 2D array. I will use this attribute when iterating through loops that I will explain about later. On line 1940, I have called a new method that deals with displaying the actual graph window.

```
1939      self.length = len(self.progressList)
1940      command = self.DisplayGraph()
```

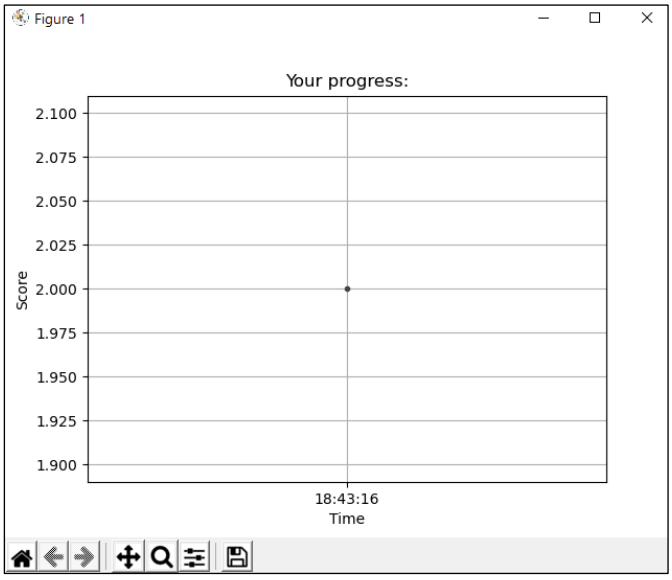
On line 1945 and 1946, I have created two new lists which will be used for each axis of the graph.

```
1943    def DisplayGraph(self):
1944      #--Additional lists that data will be appended to
1945      Time_Scores = []
1946      Scores = []
1947
1948      #--These variables will allow me to iterate through lists
1949      x = 0
1950      u = 0
1951      z = 0
1952
1953      for z in range(self.length+1):
1954        #--Appends all the times to a list
1955        Time_Scores.append(self.progressList[u][0])
1956        #--Appends all the scores to a list
1957        Scores.append(self.progressList[u][1])
1958        #--Iterates through list
1959        u+=1
```

I have used a for loop on line 1953 to iterate through the 'self.progressList' array. This is so I can append the scores and times to two lists which will be used in the Matplotlib graph.

Once this process has completed, I have converted the scores to integers and reordered them on line 1965. This is because Matoplotlib naturally plots data so that it follows a straight line. This is not realistic as a student's score is likely to fluctuate.

```
1961        if len(Time_Scores) == len(self.progressList):
1962          #--Converts the list of scores from strings to integers
1963          Scores = list(map(int, Scores))
1964          #--Orders the two lists so they can be plotted
1965          Time_Scores, Scores = zip(*sorted(zip(Time_Scores, Scores)))
1966
1967          #--Graph is plotted
1968          plt.plot(Time_Scores, Scores, color='#444444', marker = '.', linestyle= '-')
1969          plt.xlabel(f"Time")
1970          plt.ylabel(f"Score")
1971          plt.title(f"Your progress:")
1972
1973          plt.grid(True)
1974          plt.show()
1975
1976          print("Graph drawn")
1977          #--For loop is broken
1978          break
```

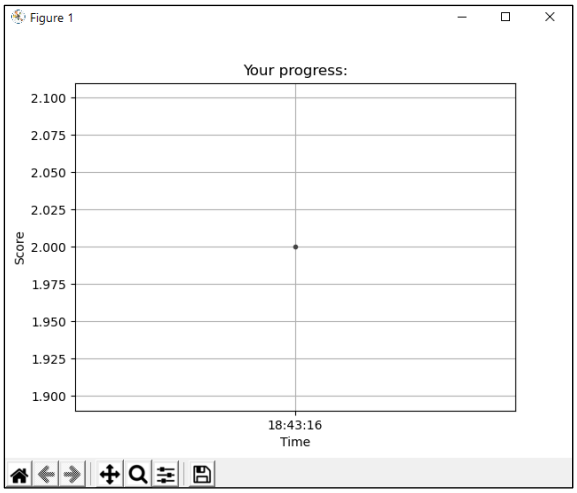


As a usability feature which relates to my screen designs, I have used gridlines to make it easier for the user to assess their progress.

Development Testing

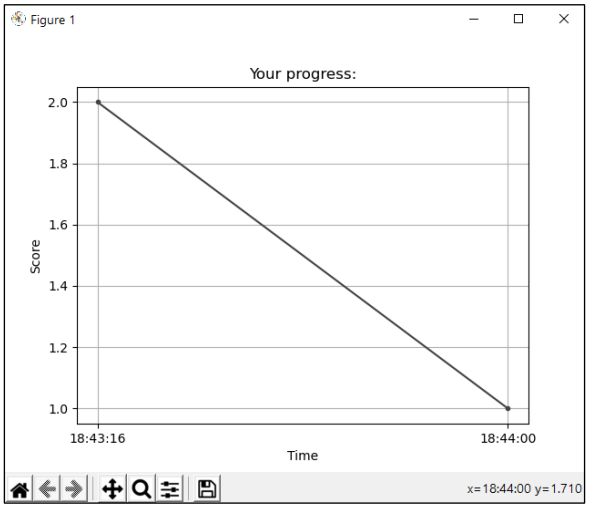
Draw Graph Button	Valid input – button is pressed	Opens a new Matplotlib window with a graph drawn	So the student can view their progress	6.0

When the user completes the quiz for the first time:



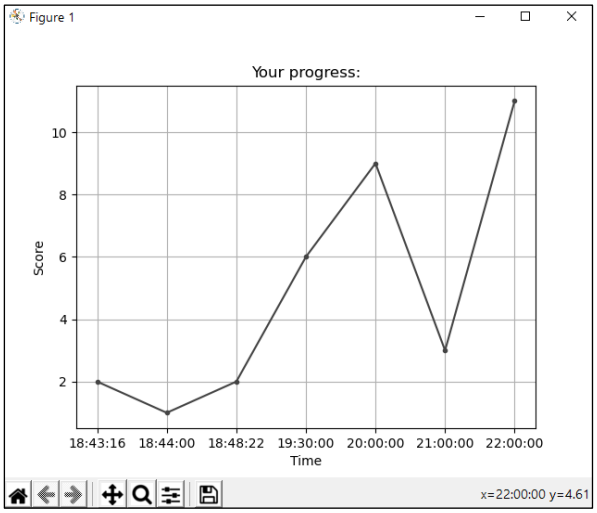

```
Your score is 2
Quiz finished
Graph Drawn
```

When the user completes the quiz for the second time:




```
Your score is 2
Quiz finished
Graph Drawn
Quiz repeated
Your score is 1
Quiz finished
Graph Drawn
```

To test robustness, I have completed the quiz multiple times:




```
Your score is 2
Quiz finished
Graph Drawn
Quiz repeated
Your score is 1
Quiz finished
Graph Drawn
Quiz repeated
Your score is 2
Quiz finished
Graph Drawn
Quiz repeated
Your score is 6
Quiz finished
Graph Drawn
Quiz repeated
Your score is 9
Quiz finished
Graph Drawn
Quiz repeated
Your score is 3
Quiz finished
Graph Drawn
Quiz repeated
Your score is 11
Quiz finished
Graph Drawn
```

Amendments:

When the user exits the program, I have deleted the file which contains the scores and times belonging to the user. I have done this because if someone else wants to use the program, their

scores and times will get saved to the same file. This means on the graph, they will see the scores of other people which isn't very helpful.

Below, I have imported the module 'os' so I can delete the file in the user's directory.

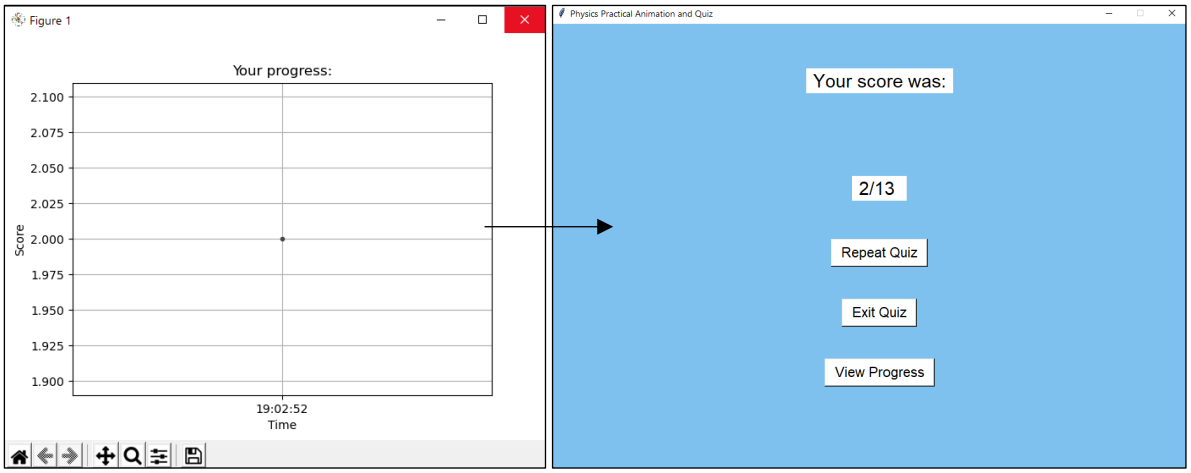
```
7 import os
```

```
1768     #--Deletes this file so each student has their own progress charts
1769     os.remove('ScoreProgress.csv')
```

When the user starts the program, this file will be created again.

Development Testing:

Exit button	Valid input – button is pressed	Exits window and return to the main menu	When the student has finished viewing the graph	6.1



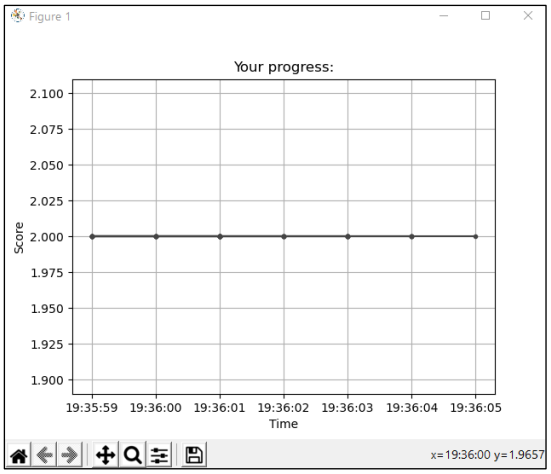
The user can close the Matplotlib window and return to the main screen. They can choose to exit or repeat the practical if they want to.

Shown by the above evidence, both my tests 6.0 and 6.1 were a success.

Validation

I have validated the user's input. Shown below, the program coped with all types of inputs.

Result	Valid input – Clicking the graph button	Invalid input – Clicking anything that isn't the exit button	Boundary input – Repeatedly clicking or holding the exit button
Expected output?	Yes, the graph window was displayed	Yes, the graph window was not displayed	Partially, the graph window was displayed but the plotted points were unexpected



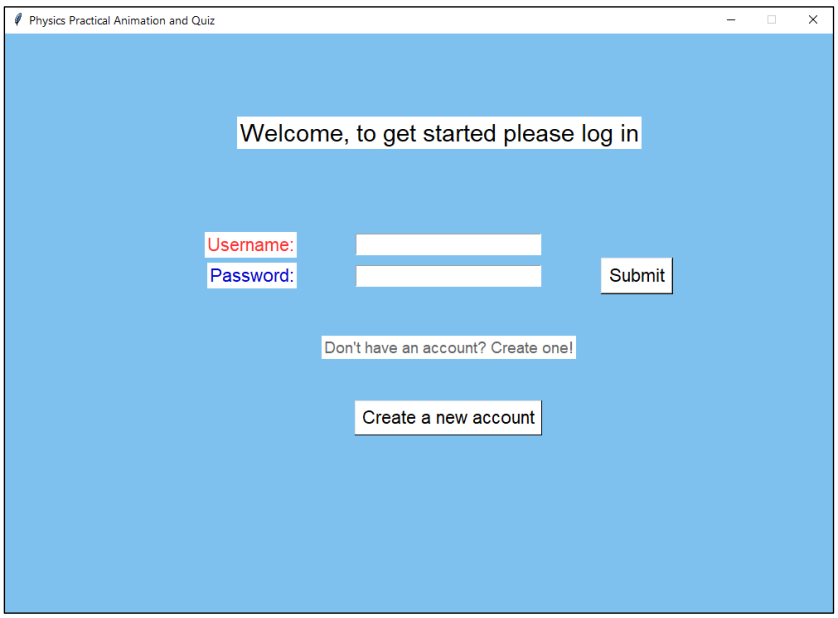
This is what the graph window looked like when the button was pressed multiple times. The same score is plotted multiple times since when the graph button is pressed, it goes through the whole process of the appending and reading from a file again. I attempted to solve this by introducing a Boolean value which prevents this from happening. However, due to time constraints, I didn't get to finish this.

Final touches

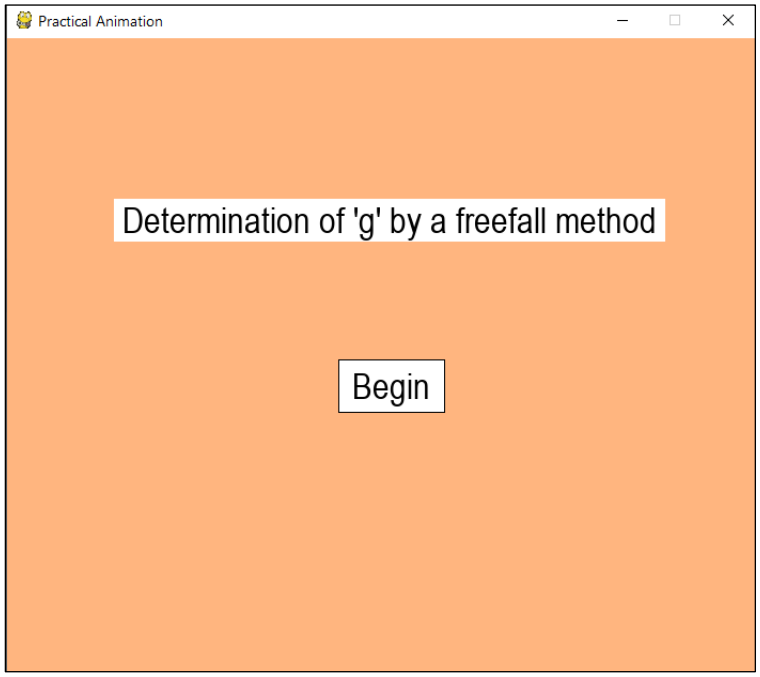
I have now completed the development of my program. Before I move onto evaluation, I want to make some final touches to my program.

Since I used a new background colour for the quiz screen, I will also use this for the login screen to provide uniformity.

```
19 window.configure(bg="SkyBlue2")
```

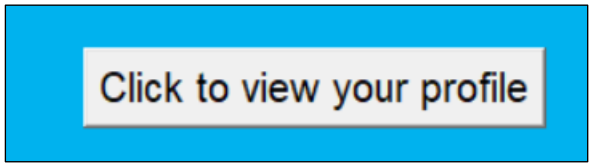


I also changed the background colour for the animation screen so that it was more aesthetically pleasing.

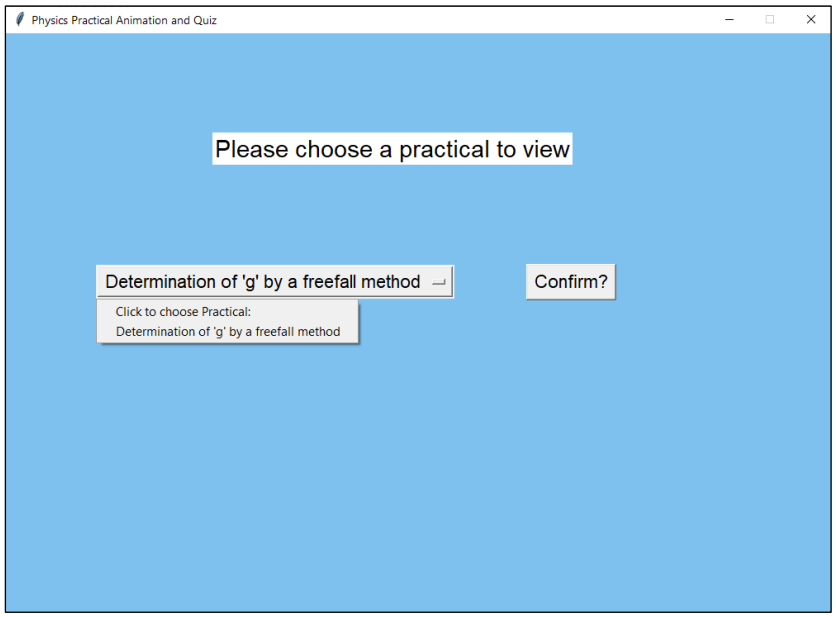


Profile button:

When the user chooses a practical, there is an additional profile button allowing them to view their past scores.



Since I didn't get enough time to implement this, I will remove this button from the screen. The screen for choosing the practical looks like this:



I will now move onto the evaluation where I will assess whether my program meets its success criteria.

EVALUATION

Post-Development Testing

Below is the testing that will be done by the user.

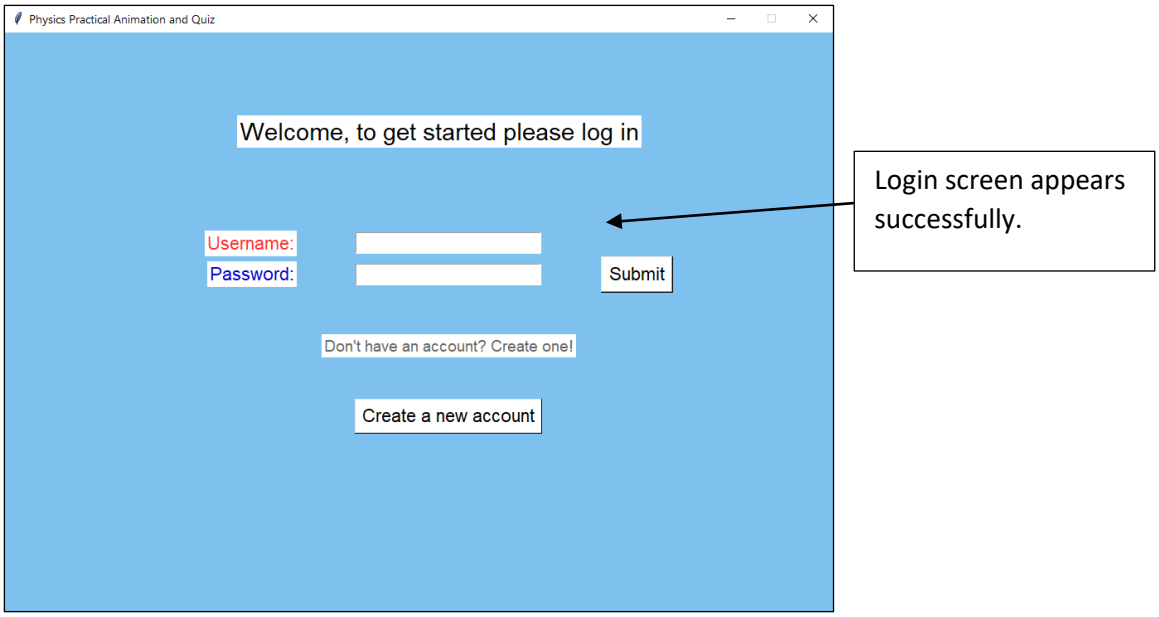
Test Number	What is being tested?	Input data	Data input type	Expected outcome	Actual outcome	Test Evidence
1	Does a login screen appear when the program is run?	Running the program	Valid	Login screen displayed	As expected	Screenshot 1
2	Is the user able to login successfully using their username and password?	Username and Password	Valid and boundary	User is transitioned to the next screen	As expected	Screenshot 2 Screenshot 3
3	Does the program reject any invalid username and password attempts?	Invalid username and password	Invalid	User must re-enter their username and password	As expected	Screenshot 4
4	Does the program limit the user to five login attempts?	More than five login attempts	Invalid	User has to wait 30 seconds before they can try again	As expected	Screenshot 5
5	Is the user able to create a new account?	Clicking the create account button	Valid	User is transitioned to a new screen where they can enter their new details	As expected	Screenshot 6
6	Is the user able to save a valid username and password and then login with it?	A new username and a new password	Valid	User can log in with their new details	As expected	Screenshot 7
7	Does the program reject invalid usernames	Invalid username and password	Invalid	User can have another go at entering their username and password	As expected	Screenshot 8

	and passwords?					
8	Can the user choose a Physics practical animation to watch?	Clicking drop-down menu to choose practical	Valid	User can choose an animation to watch	As expected	Screenshot 9
9	Can the user start the animation?	Clicking button to play the animation	Valid	Animation can be played	As expected	Screenshot 10
10	Can the user drop the ball using the spacebar?	Clicking spacebar	Valid	The ball accelerates due to gravity and the time taken is displayed	As expected	Screenshot 11
11	Can the user change the height between light gates?	Clicking the up and down arrow buttons	Valid and boundary	Light gates can be moved for various heights. The ball should be able to be dropped for these heights.	As expected	Screenshot 12 Screenshot 13
12	Can the user repeat the animation?	Clicking spacebar	Valid	Animation is repeated	As expected	Screenshot 14
13	Can the user open a graph of relevant plotted points?	Clicking the graph button	Valid and boundary	Opens a new window containing a graph	As expected	Screenshot 15
14	Is the user able to exit the animation?	Clicking the exit button in the window	Valid	Animation window should be closed	As expected	Screenshot 16
15	Is the user able to start the quiz?	Clicking button to start the quiz	Valid	Quiz should start	As expected	Screenshot 17
16	Are the correct questions and answers displayed?	No input	Valid	Correct questions and answers should be displayed	As expected	Screenshot 18
17	Can the user submit their answers?	Clicking multiple-choice buttons	Valid	The program should receive user inputs	As expected	Screenshot 19
18	Does the program detect correct answers?	Clicking correct answer	Valid	The user's score increases by 1	As expected	Screenshot 20
19	Does the program	Clicking the wrong answer	Valid	The user's score remains the same	As expected	Screenshot 21

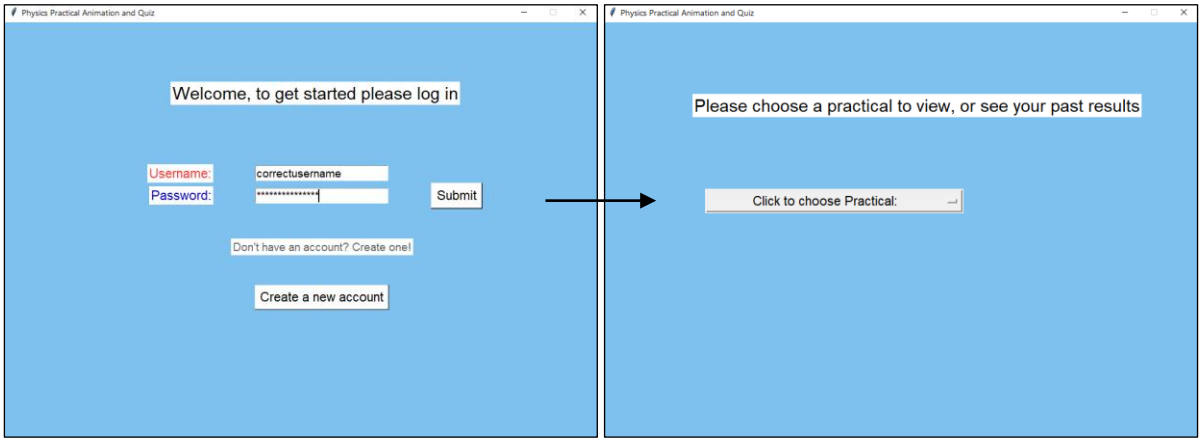
	detect wrong answers?			The correct answer the user should have pressed should be displayed		
20	Does the program display the next question?	Clicking either a correct or incorrect answer	Valid	Next question should be displayed	As expected	Screenshot 22
21	Does the program keep and display a score?	No input	Valid	The score is updated every time the user answers a question and is displayed at the end	As expected	Screenshot 23
20	Can the quiz be repeated?	Clicking repeat button	Valid	The user can try the quiz again	As expected	Screenshot 24
21	Can the user view a graph of their progress?	Clicking graph button	Valid	A graph of the user's progress is displayed	As expected	Screenshot 25
22	Can the user view their past scores in a file	Clicking past scores button	Valid	The user's past scores are displayed	Not expected	Screenshot 26
23	Can the user log out and return to the login screen?	Clicking log out button	Valid	User is returned to the login screen	Not expected	Screenshot 27
24	Can the user exit the program entirely?	Clicking the exit button	Valid	Program is terminated	As expected	Screenshot 28

Evidence:

Screenshot 1

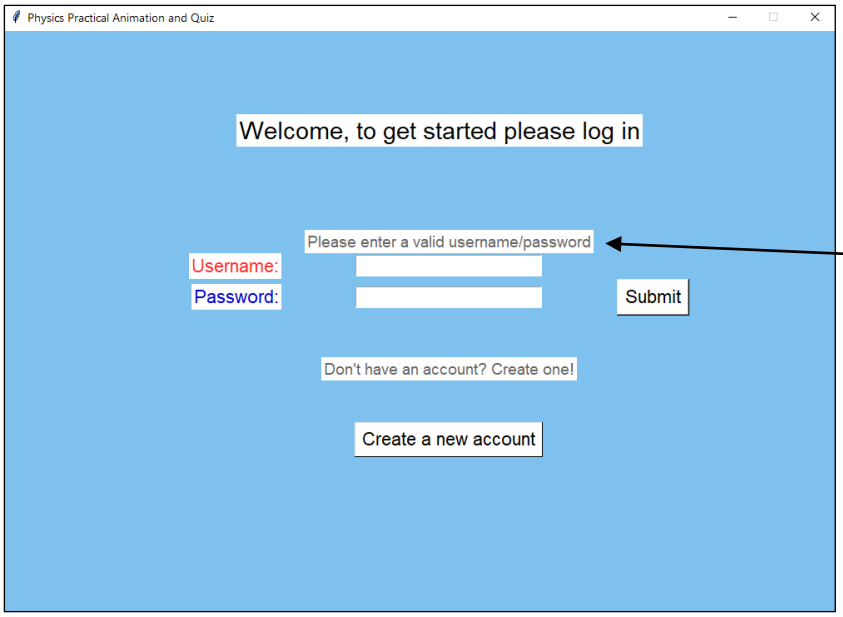


Screenshot 2:



When entering a valid username and password, the user is transitioned to a new screen. When entering a password, the program displays it as asterisks which is an important security feature.

Screenshot 3

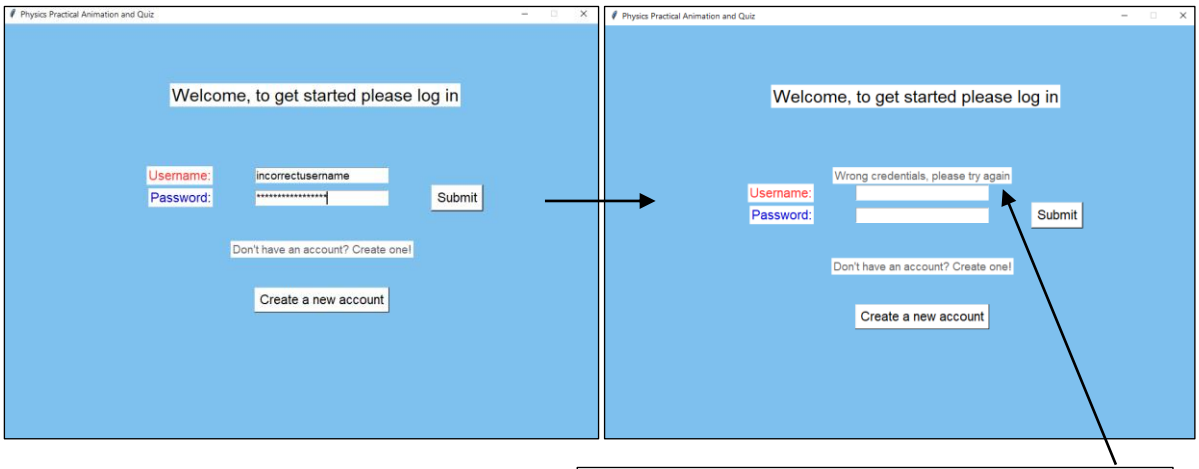


Program rejects boundary inputs – an example being entering no username or password

My program is robust as it prevents boundary inputs. In this case, the user be will be allowed another go at entering their details.

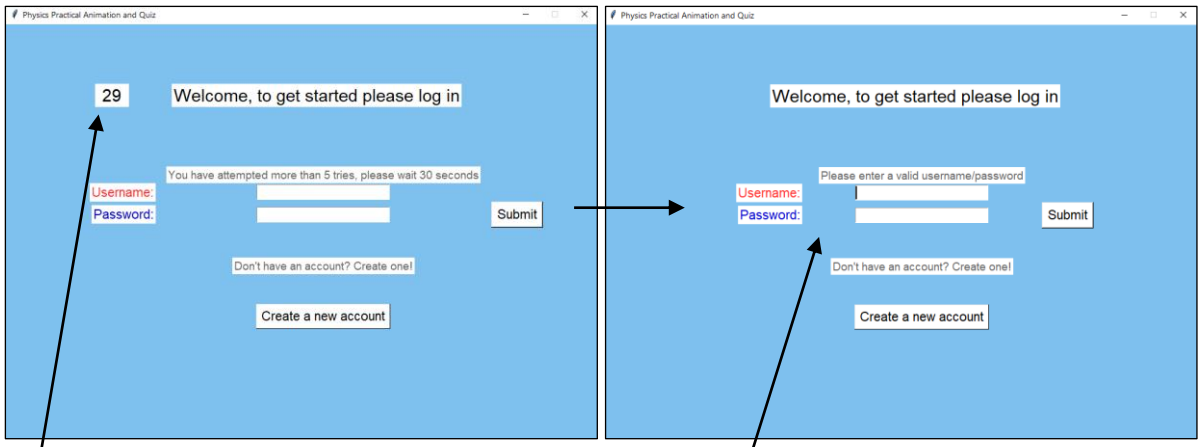
Screenshot 4

Entering an invalid username or password



The program prompts the user to try again if their entered details are incorrect. This is part of usability as it allows the user multiple attempts at entering their details.

Screenshot 5

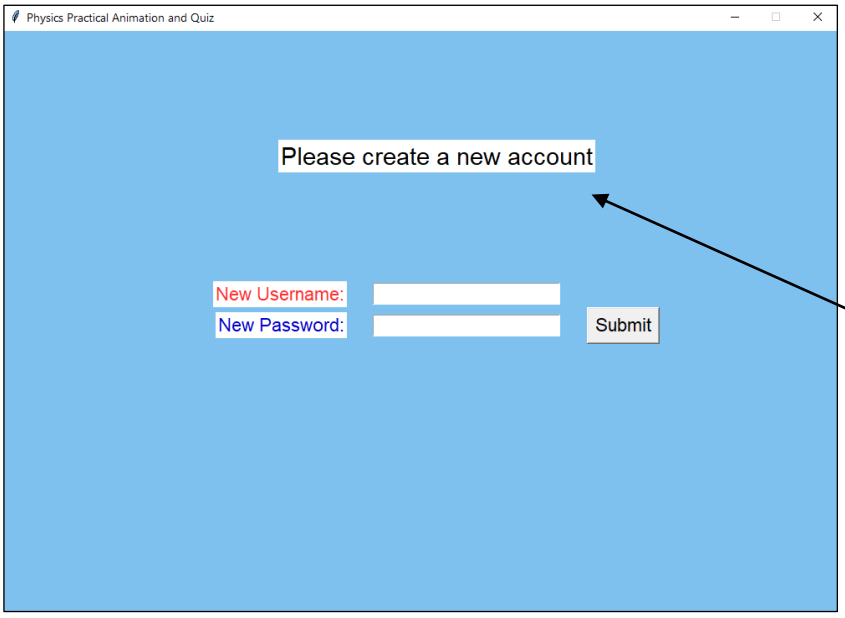


If user enters their details incorrectly more than 5 times, they will be locked out for 30 seconds.

After 30 seconds, they can enter their details again.

Additionally, if the user attempts to enter their details during the 30 seconds, they will be locked out for a further 30 seconds. This improves the robustness of my program as it prevents brute-force attacks.

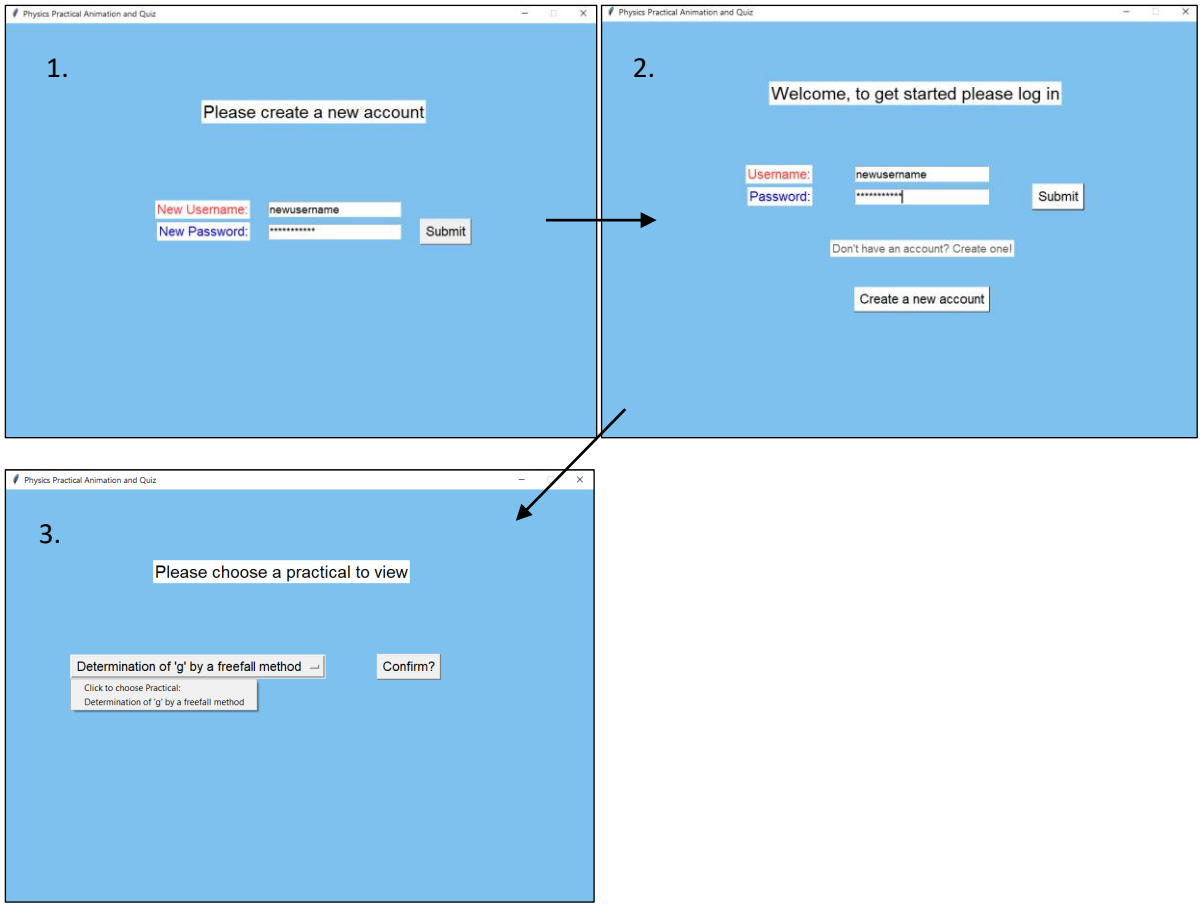
Screenshot 6



The user can create an new account

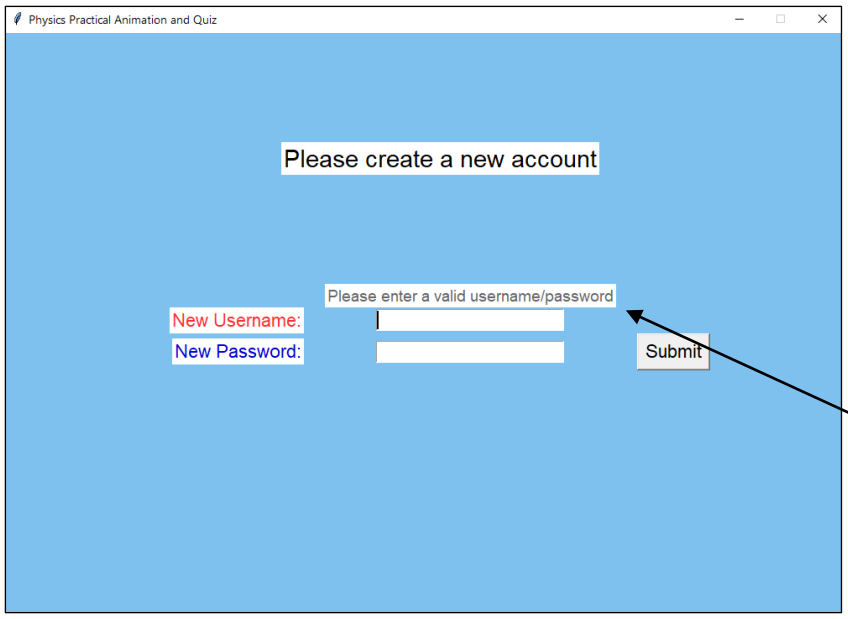
Screenshot 7

Entering a valid username and password



1. The user enters their new credentials which will be saved
2. When they press the submit button, they are transitioned to the login screen.
3. If they enter their credentials correctly, they are transitioned to the next screen.

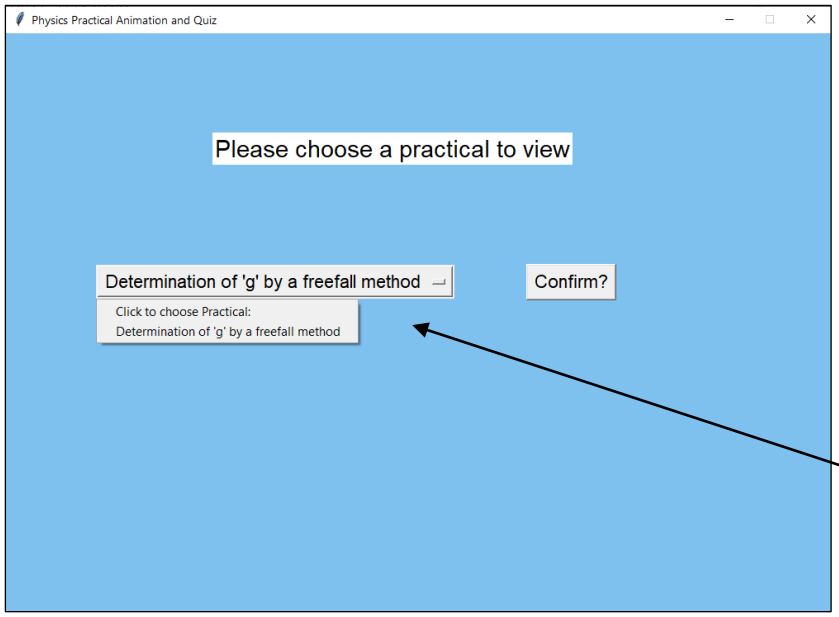
Screenshot 8



If the user doesn't enter anything, the program rejects this and prompts the user to try again.

By rejecting this input, my program is robust as it prevents blank entries from being added to the CSV file.

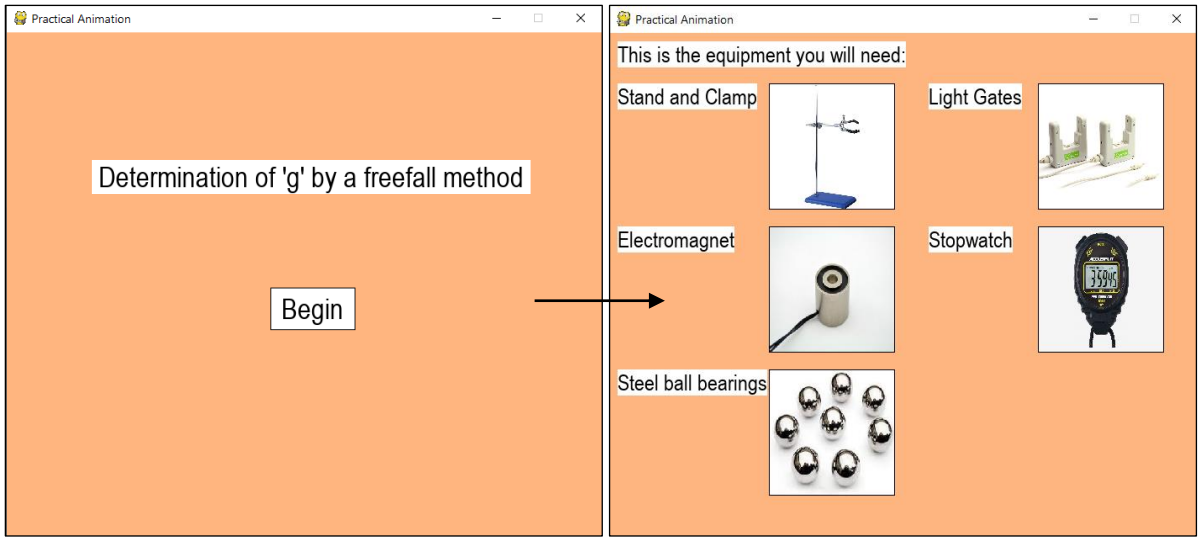
Screenshot 9



The user is able to choose a practical from a dropdown menu

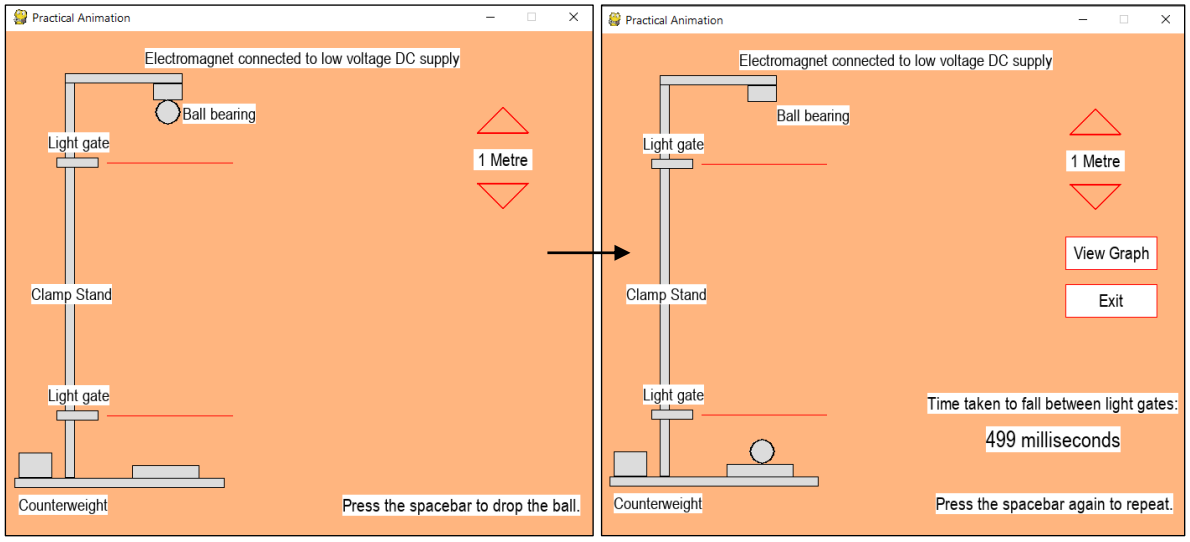
In case the user clicks on an option but doesn't want to continue with it, I have included a confirm button. This improves the usability of my program as it allows the user to validate whether they do want to view the chosen animation.

Screenshot 10



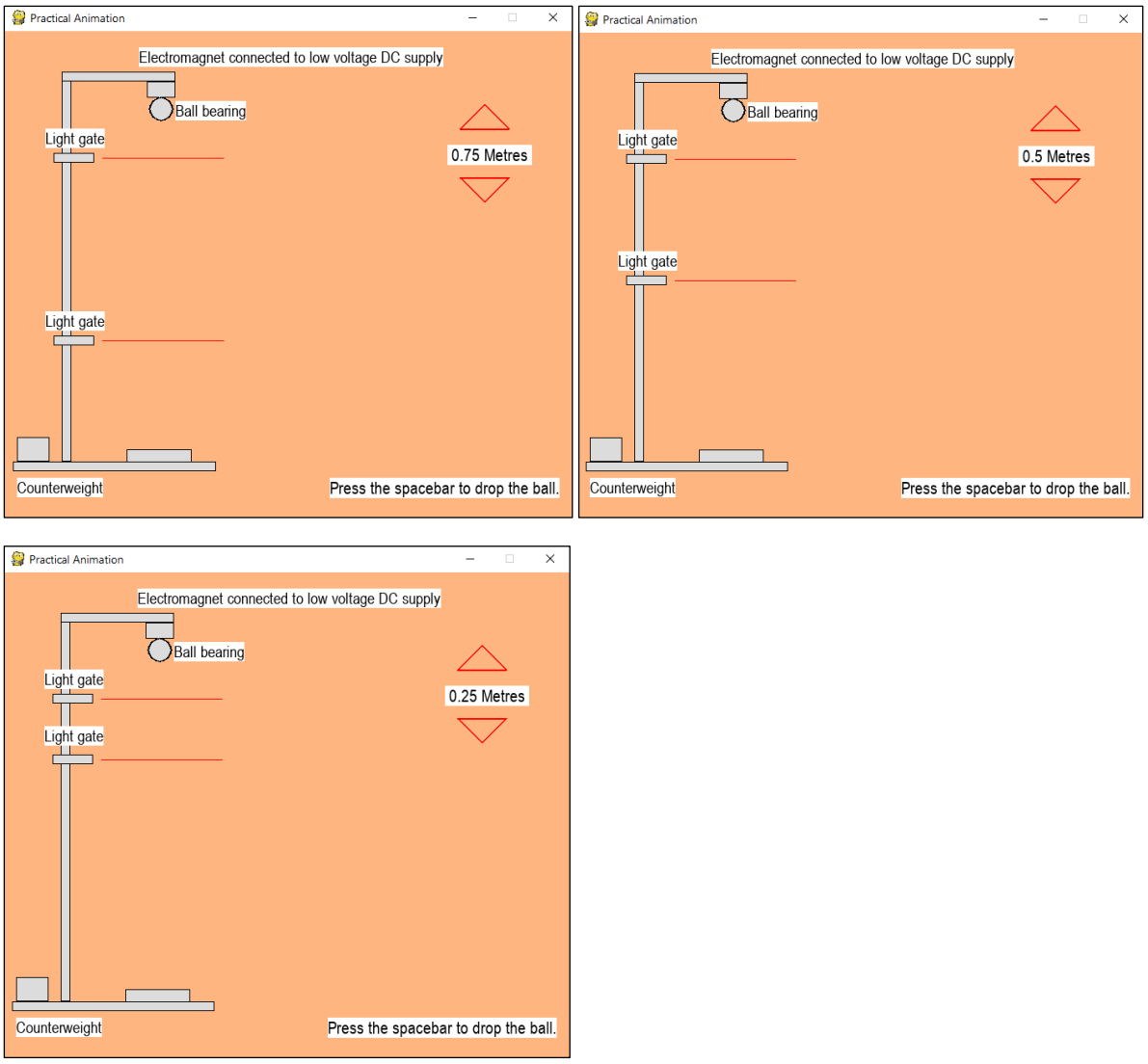
When the begin button is pressed, the screen is transitioned. The pictures are displayed one at a time so that the user has enough time to read what is onscreen. This improves the usability of my program.

Screenshot 11

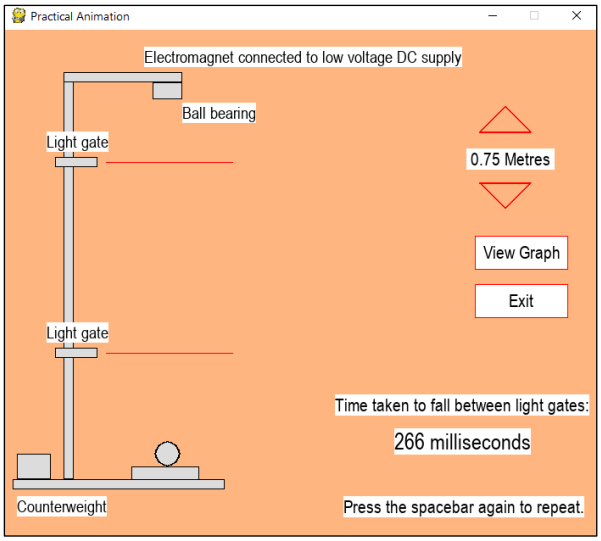


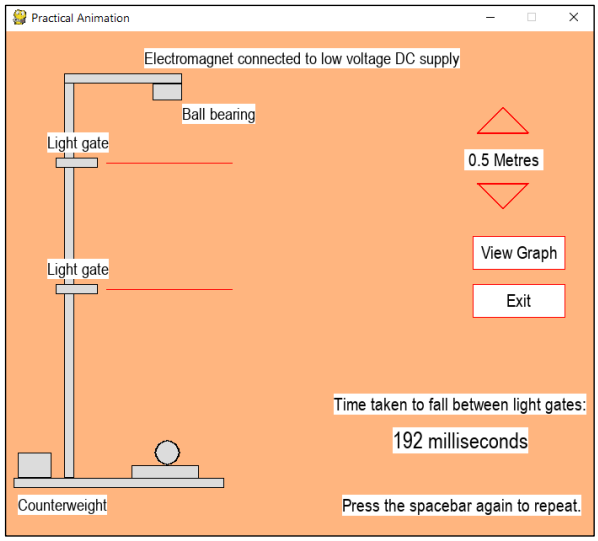
When the spacebar is pressed, the ball accelerates due to gravity. The time taken for the ball to fall is displayed.

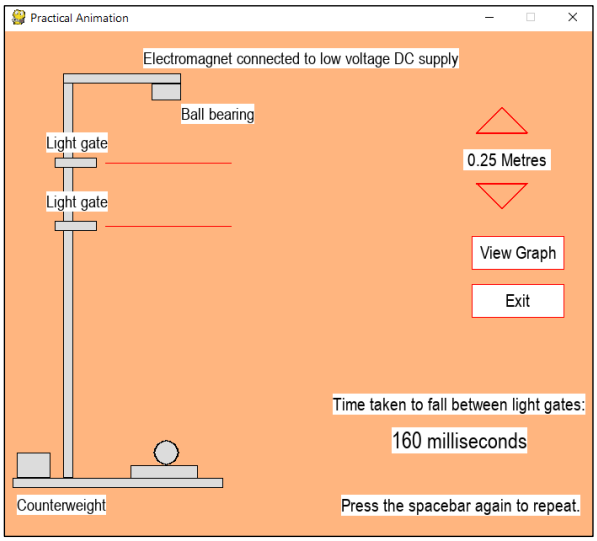
Screenshot 12



When the user pressed either the up or down arrow button, the light gates changed positions to reflect this. The ball could then be dropped for these heights. Even though the buttons are triangular, the area needed to press them is rectangular. This is to increase the area that will be registered as a click which improves usability.

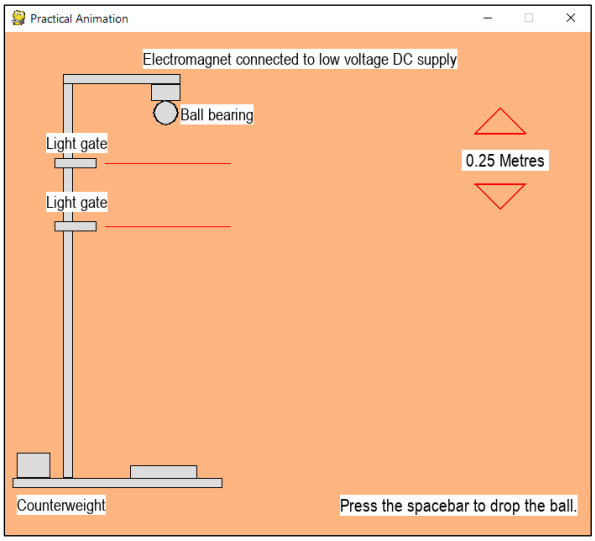


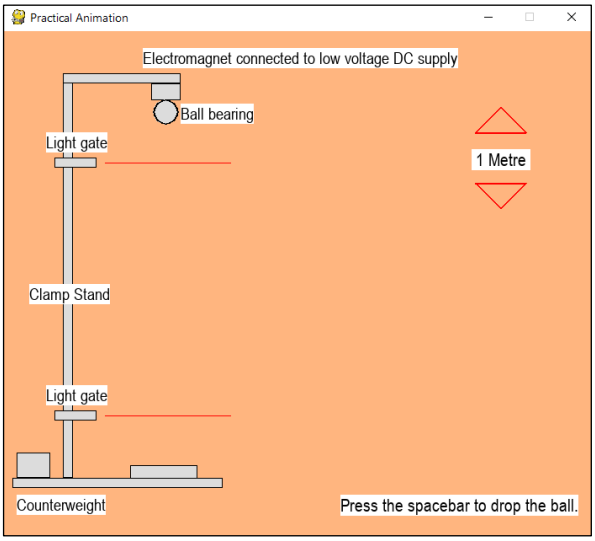




Screenshot 13

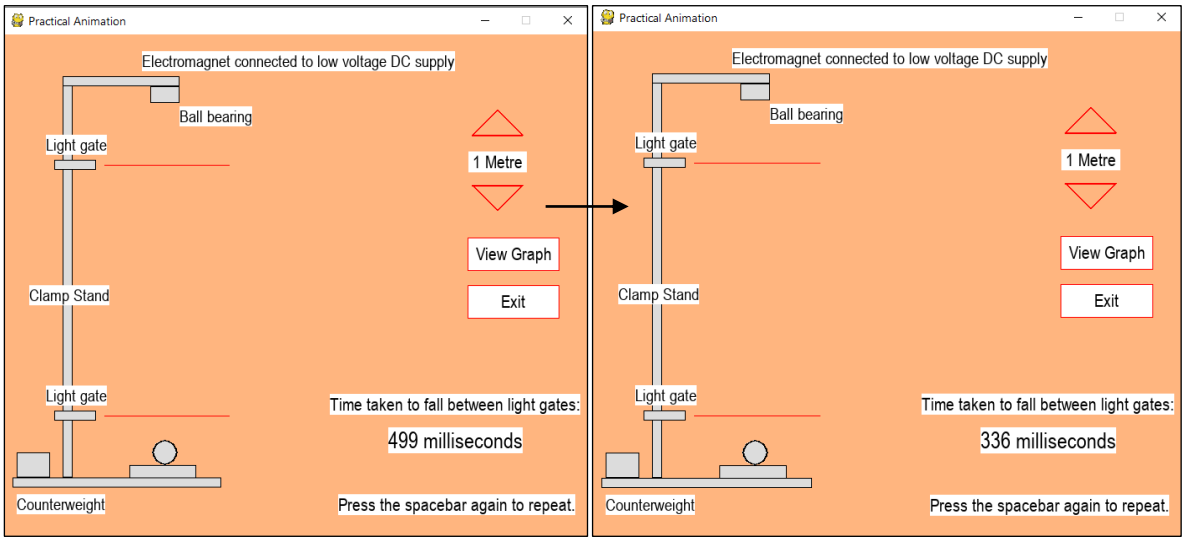
Boundary input





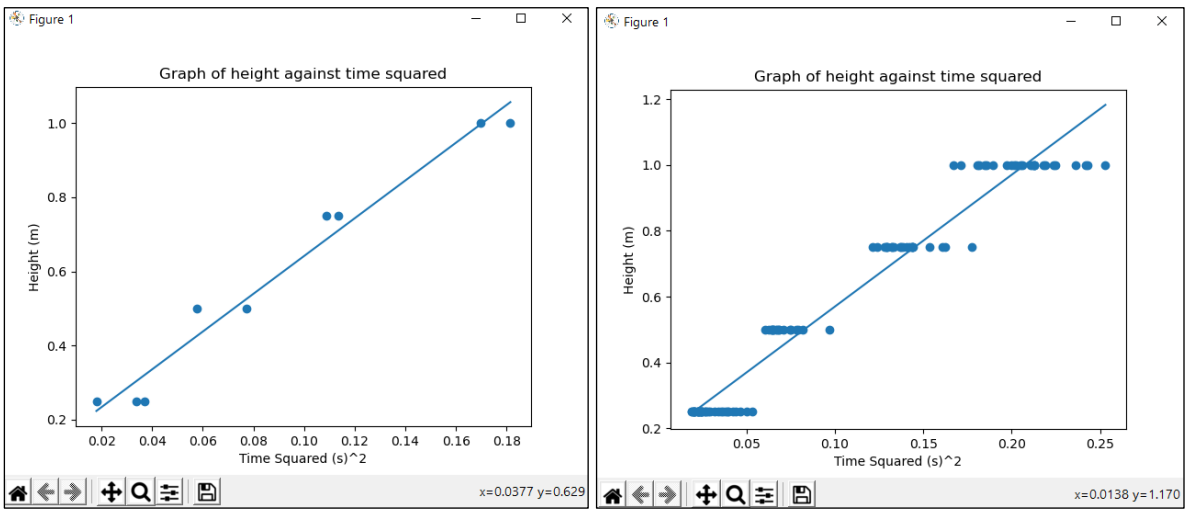
When the buttons were pressed at the maximum and minimum height of 1 metre and 0.25 metres, the light gates did not move. This shows that my program is robust as only valid button presses were accepted.

Screenshot 14



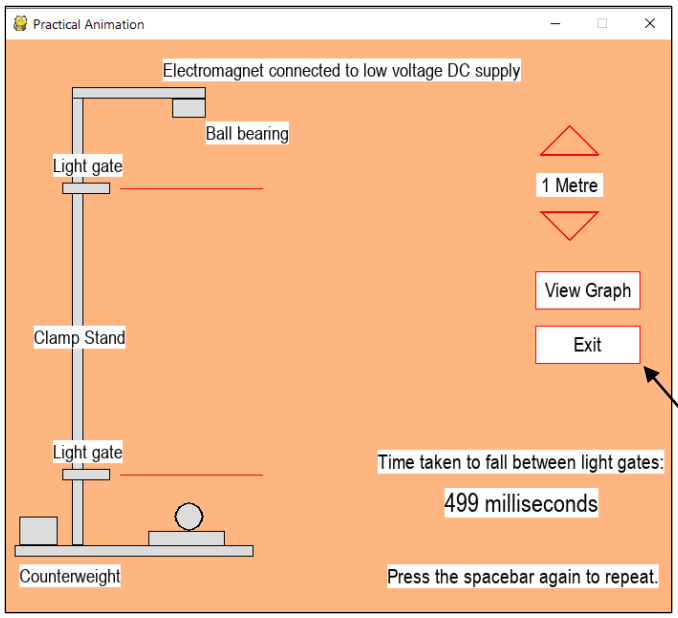
The program can be repeated multiple times, shown by how a different time has been recorded and displayed.

Screenshot 15



When the graph button is pressed, relevant points are plotted and a line of best fit is drawn. I have tested this with a variety of inputs. Shown on the right is when the animation has been repeated many times. This demonstrates the robustness of my program.

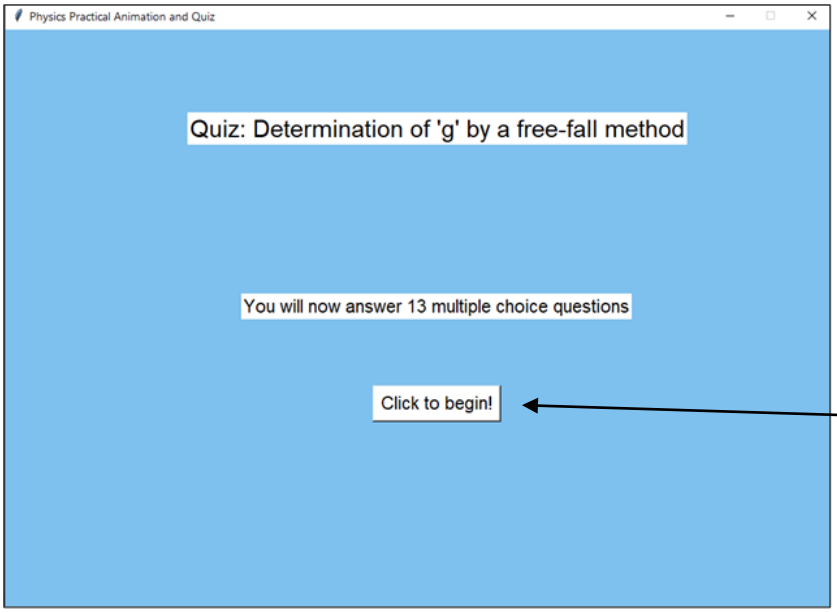
Screenshot 16



When the exit button is pressed, the animation window is closed

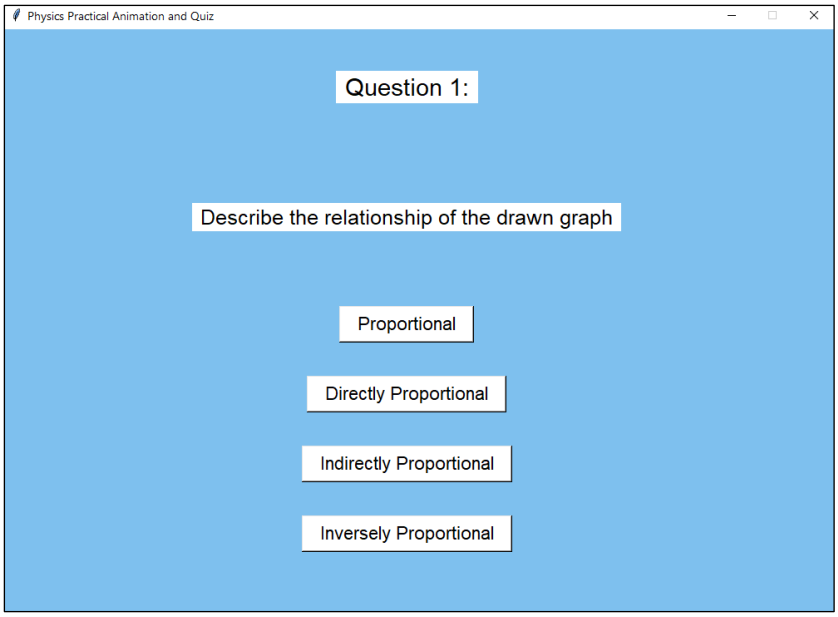
The user always has this option available to them giving them full control over the animation, improving usability.

Screenshot 17



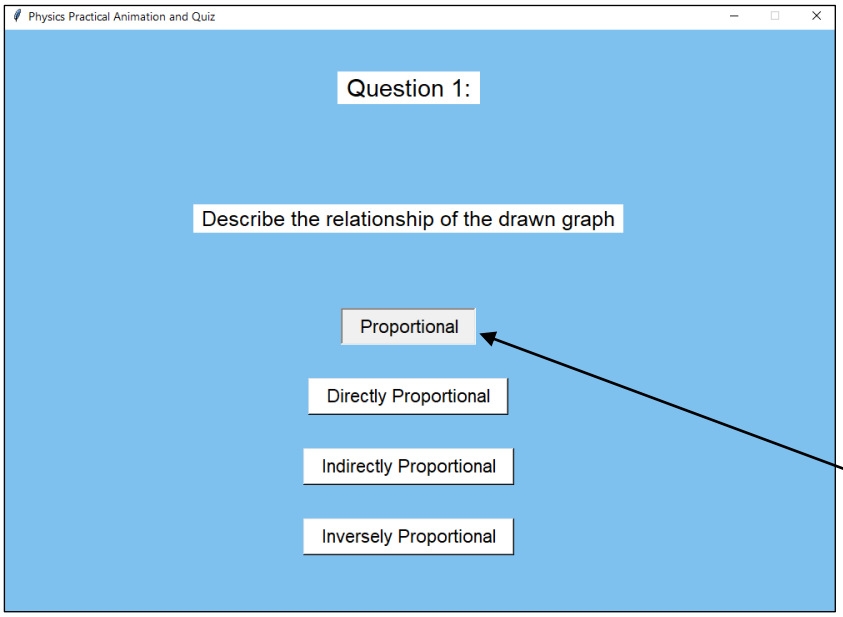
When the animation window is closed, the user will be transitioned to a new window containing the quiz. Here, they are able to start the quiz.

Screenshot 18



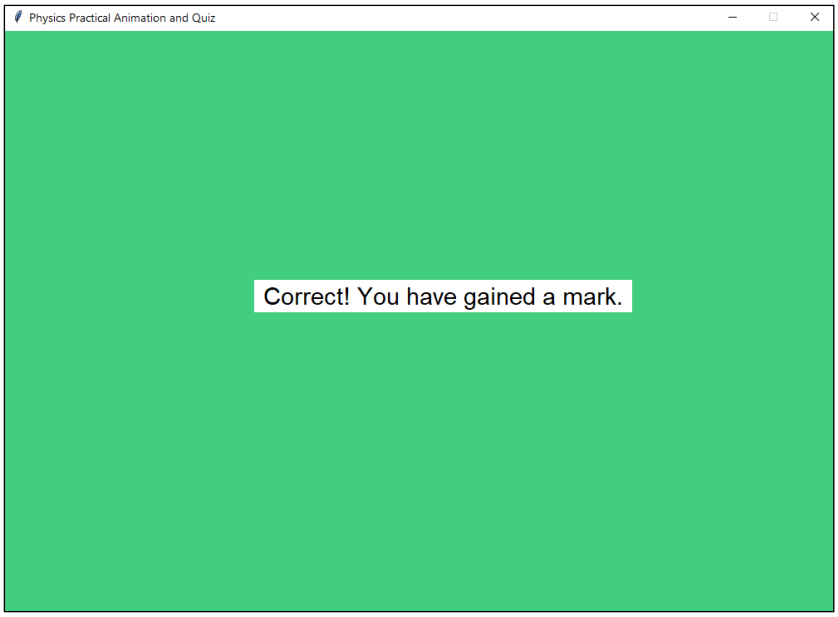
When the quiz is started, the correct questions and answers are displayed. The rest of the questions and answers are the same as the screenshots for development tests 5.3 and 5.4.

Screenshot 19



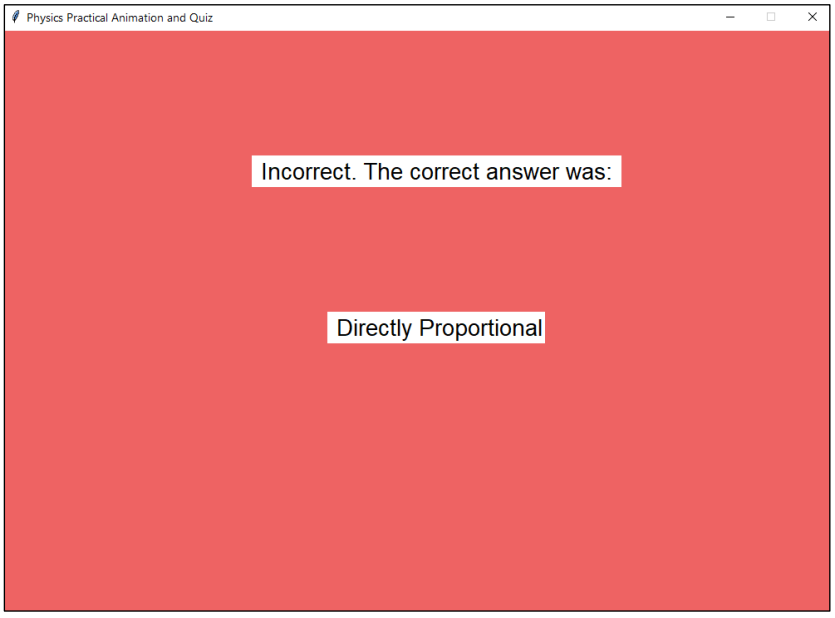
The user is able to click any of the answers which the program will detect

Screenshot 20



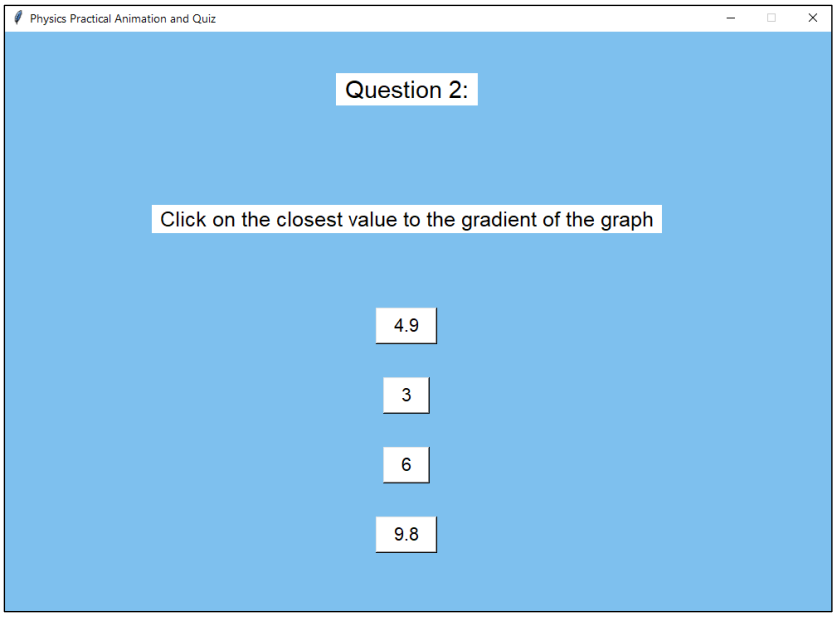
If the user gets the question correct, this screen will be displayed and their score will increase by 1.

Screenshot 21



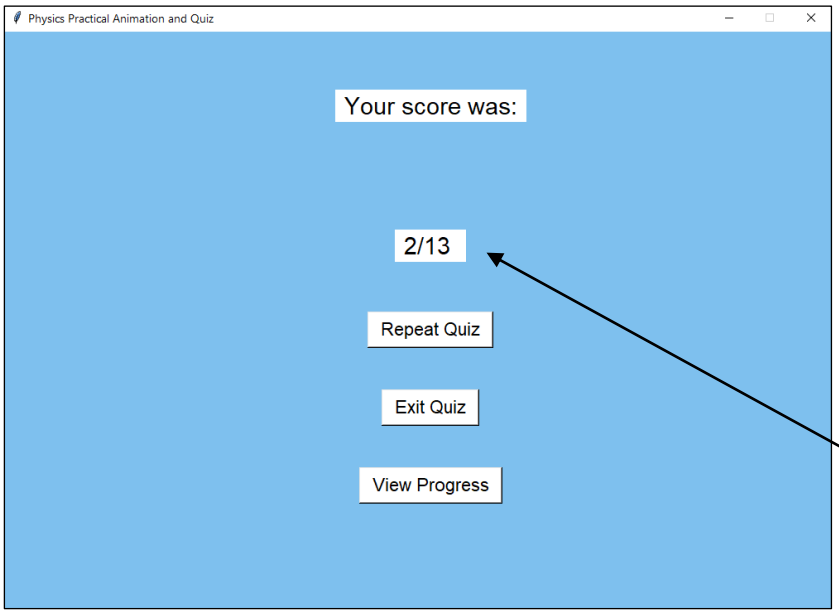
If the user gets the question incorrect, the correct answer will be displayed and their score will remain the same.

Screenshot 22



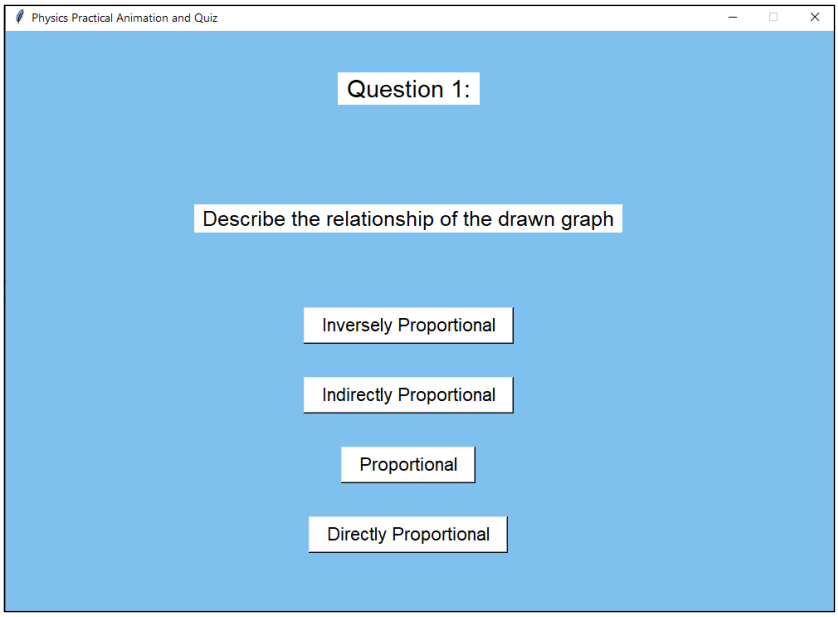
After the user sees whether they have got the previous question correct or not, the next question will be displayed. This will continue for the rest of the questions and answers as per my development tests 5.3 and 5.4.

Screenshot 23



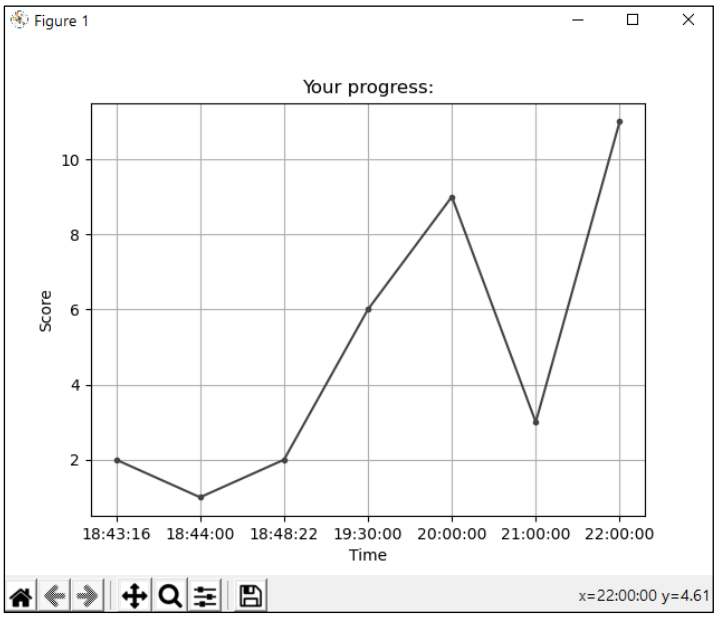
When the user finishes the quiz, their score is displayed onscreen.

Screenshot 24



The quiz has been repeated shown by how the answers are in a different order. Arranging the answers in random positions is a successful usability feature as it prevents the user from guessing the answers.

Screenshot 25



A new window is opened containing the graph of the user's progress over time

Additionally, this screenshot illustrates my program's robustness as the quiz has been repeated multiple times successfully.

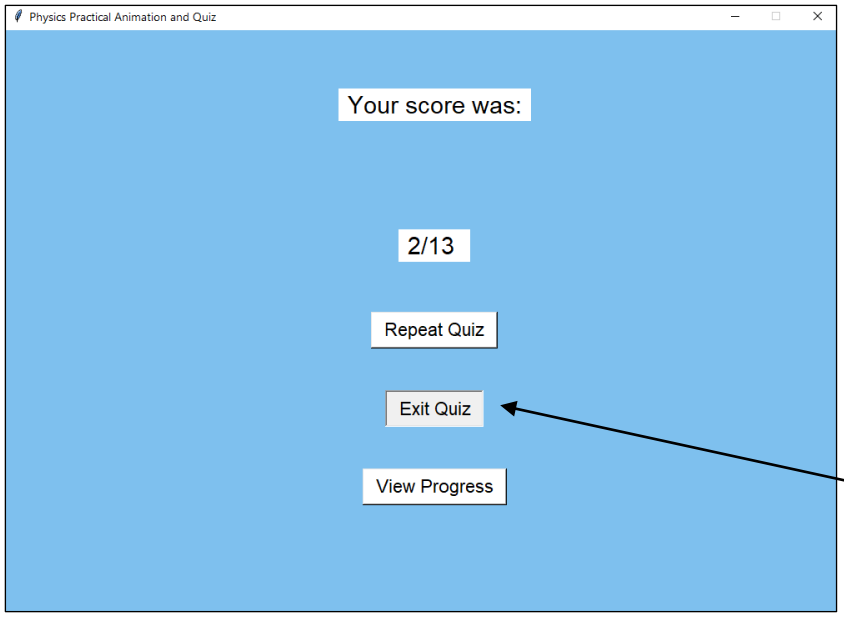
Screenshot 26

Unfortunately, I did not get time to implement this so this test was not a success.

Screenshot 27

Unfortunately, I did not get time to implement this so this test was not a success.

Screenshot 28



When the exit button was pressed, all windows and processes that were part of the program were closed

MEETING THE REQUIREMENTS:

I went through my success criteria to see what my application achieved.

The program must have a functional login system that allows students to sign in.
I can say that my program has a functional login system. The student can successfully sign in provided they have entered the right details. The login screen is easy to use and secure, an example being the use of asterisks in the password entry box. This item in the success criteria was a success determined by development tests 1.0 to 1.7 and post-development screenshots 1 and 2

There should be a validation system so the student can try again but there should a limit to this of around 5 attempts.
My program validates the user's inputs meaning that it is robust. For example, I included a validation system where the user must re-enter their details if they enter them incorrectly. Additionally, there is a countdown timer which makes the user wait 30 seconds if they enter their details incorrectly more than 5 times in a row. This is shown by development tests 1.1 and 1.4 and post-development screenshots 3 and 4.

The student must be able to create a new account with which they can log in.
The student can successfully create a new account with which they can log in. This is stored in a CSV file meaning that they will always have access to the program, even when they close and reopen it. I have validated the user's inputs as blank entries are rejected by the program. This is shown by development tests 2.0 to 2.4 and post-development screenshots 5, 6 and 7.

The program must use CSV files to store student data, so it is well organised and easy to manipulate.
I have used CSV files throughout my program to store student and question data. This is shown by development tests 2.1, 2.3 and 5.2.

The practical animation should have interactive buttons and sliders. This is demonstrated in my primary research as my stakeholder was keen on having this ability.
I have achieved this goal as the student can use buttons and keys to interact with the animation. I have shown this through development tests 4.0 to 4.8 and post-development screenshots 11, 12 and 13.

The animation must have labels, images and sound effects.
My program partially achieves this goal. I have included labels and images of the equipment needed in my animation. This is demonstrated by development test 4.0 and post-development screenshots 10 and 11. However, I did not get the chance to put sound effects in my animation. I feel that this would have made the experience more enjoyable but labels and images would have been more useful for a student's revision.

The user should be able to view a graph within the animation linking to my secondary research.
My program allows the student to view a graph of plotted points which links to the height of the light gates and time taken for the ball to fall. This graph can be updated with new points depending on how many times the animation is repeated. Moreover, I have included validation where the animation must be repeated at least 4 times to ensure enough data points are plotted. This is shown by development test 4.7 and post-development screenshot 15.

The user will be able to view the animation as many times as they want.
The user can repeat the animation as many times as they want by pressing the spacebar. This is shown by development test 4.4 and post-development screenshot 14. However, this does not repeat the part of the animation where the images are displayed. Therefore, in further development, I could incorporate this.

The student should have the option to only watch the animation and skip the quiz
I have not met this item of the success criteria. When the animation is closed, the quiz is immediately opened. I could have added a screen where the user could choose whether they wanted to do the quiz. However, when the quiz screen is displayed, the user still has to decide whether they want to do the quiz. If they do not want to do it, they can close the program.

The quiz should ask no more than 16 multiple choice questions to keep it from becoming too tedious.
My quiz asks 13 multiple choice questions which meet the requirement set out by my stakeholders. This is shown by development test 5.2.

The questions must be of reasonable difficulty and be based on aspects of the practical process.
The questions asked are of reasonable difficulty since they mimic exam-style questions. When testing my program, my stakeholders found them challenging which they liked. They are also based

on aspects of the practical. For example, questions were asked on the Matplotlib graph that was plotted. This is shown by development tests 5.2, 5.3 and 5.4.

There should be a working timer, so the students must complete questions in an allocated time. However, I must ensure that there is ample time to reduce stress
Unfortunately, I did not get round to implementing a timer in my program. My stakeholders did not mind this though as they felt that the quality of the questions was more important. Later, I will discuss how this is a feature that I could implement in the future.

The correct answer that should have been pressed should be displayed if the student gets the question wrong.
When the student answers a question incorrectly, the correct answer will be displayed. This is shown by development test 5.4 and post-development screenshot 21.

Once they have finished the questions, a graph of a student's progress must be plotted for them to easily visualise their progress.
I met this goal as a graph of the student's scores over time was plotted correctly. This was demonstrated by development test 6.0 and post-development screenshot 25. While this was a success, I feel that the program should be able to show this progress over a larger period of time. Currently, the user can only see their progress over time if they repeat the quiz.

There should be an option for the user to have another go at the quiz.
The user can repeat the quiz as many times as they want to, demonstrating the robustness of my program. This is shown by development test 5.8 and post-development screenshot 24

Having a drop-down menu with the student's details such as past scores or wrongly answered questions.
I did not have enough time to implement a menu which included this information. However, the student is still able to see their past scores through the plotted graph.

Once finished, the student can have the option to log out and return to the main menu.
Due to the problems I faced in my development, I was not able to include this feature. However, an easy alternative would be for the user to close and reopen the program. Since all their details will be stored in files, they can log in again.

I went through my program showing my stakeholder every feature so I could gather feedback.

End-User feedback:

Stakeholders: Joe and Abbas

1. Does my application meet your criteria on a successful learning tool for A-Level Physics Practicals?

   Joe: "Yes, it does. I liked the range and number of questions and there were plenty of buttons to play with. While I initially requested for a countdown timer, I didn't even notice the absence of it. Moreover, the progress tracker feature was a good substitution for having a page where I could view my past scores."

   Abbas: "As a result of being easy to use, as well as consisting of complex features, this application generally meets my criteria."

2. Would you incorporate this tool into your learning when revising?

   Joe: "Absolutely. The difficulty of the questions was just right as they were not too easy or too inaccessible making it ideal for revision."

   Abbas: "With more practicals, I would definitely incorporate this into my revision."

3. Did you like the user interface of the program?

   Joe: "I liked the choice of colours although I wouldn't call it sleek."

   Abbas: "Interactions with the program were easy but needs to be smoother. For example, when showing the apparatus needed for the practical, I would like the option of having a skip button rather than automatic transitions."

4. Was the program easy to use?

   Joe: "Yes. Interacting with it such as clicking buttons was easy and the layout was well organised and wasn't too complicated."

   Abbas: "Very easy to use."

5. Which features did you particularly like?

   Joe: "My favourite aspect was the fact that each time the ball is dropped, a new data point was added to the graph. It made it feel much more realistic."

   Abbas: "The graphs and complex features like changing the heights were particularly nice."

6. Which features did you feel needed improving?

   Joe: "The correct and incorrect screens lasted too long. A skip button could be added to quicken this up."

   Abbas: "For each component of the practical, there should be skip buttons rather than automatic transitions"

7. Are there any features you would like to see added?

   Joe: "Noises would be nice. For example, when clicking buttons or when the ball falls to the ground."

   Abbas: "More practicals and more variables that can be changed by the user."

8. Any other comments?

   Joe: "No. I enjoyed the program and would use it again."

   Abbas: "A nice and easy to use application"

Analysis:

1. *Overall, my stakeholders thought that I had built a useful program that caters to a student's learning requirement.*
2. *My stakeholders both said that they would incorporate my tool into their revision meaning that my solution meets its target audience as stated in my analysis section. However, for my solution to be the most beneficial, I would need to add more practicals.*
3. *Both stakeholders liked the user interface but though improvements could be made. Joe thought the interface could be made more modern.*
4. *Both stakeholders found the program easy to use meaning that my usability features were successful.*
5. *The ability to change the height of the light gates and viewing the graph were enjoyed by Joe and Abbas.*
6. *This was one aspect of usability that I didn't meet as the program could become quite repetitive at times. With further development, I would fix this.*
7. *As said, adding more practicals for a user to view would be necessary to add in further development.*

Further Development:

Abbas wanted more practicals and variables that could be changed by the user. This would be fairly simple to do. For example, I would have to add additional classes which would contain attributes for these new practicals. When an option is chosen from the drop-down menu, the relevant class will be instantiated. There would be no limit to how many animations that I can add. The user could even view multiple animations at the same time if each animation uses a different window.

Along with this, with more time, I would create a unified CSV file which would store the user's login details and score. This would have allowed me to display a user's past scores from different days or months more easily. My stakeholder Joe thought this would have been more useful than my current implementation where the user can see only see their progress over one session.

Having a unified CSV file would make adding a drop-down menu with a list of the student's past scores and answered questions a lot easier. This is because all the data belonging to the student would be in one place.

An improvement I would make would be adding a countdown timer in my quiz. I would use iteration to count down or up to a certain time. While I have used a while loop to do this for my login screen, this may not work for the quiz. In the login screen, the timer counts down from 30 to 0 seconds indefinitely, which in the quiz, would prevent the user from clicking answers.

Additionally, adding an option for the user to return to the login screen when they have finished is something I want to include. In my development, I struggled with this as I had to destroy the login window and create a new window exclusively for the quiz. A solution to this could be using the os module to automatically restart the Python script when the user wants to return to the login screen.

Limitations:

With this further development, the program will be storing quite a lot of data about the student. If the student forgets their login details, then they will lose access to their progress. In the features of my proposed solution, I suggested that the user create a new account but a limitation of this would be that lots of redundant data would be created in the CSV file. To create a system for the user to recover their account, I would need to ask for the user's email when they first create an account.

Then if they forget their details, an email could be sent letting them reset their details. Using emails would prevent the unauthorised change of a user's details. A potential limitation of this is that I will be storing private data so encryption may need to be implemented. This would increase the complexity of my program.

When reflecting on my analysis section, I realised that there were features which would have been nice to include. An example is a leader board where the top scorers in the quiz would be displayed. However, a limitation of this would be making sure appropriate usernames are displayed. I would need some sort of validation to make sure appropriate usernames are chosen when creating an account. This would be hard to do considering that a user could type in anything for their username.

Meeting usability features:

The main usability features of my program were met. Based on user testing, logging into the program, interacting with the animation and attempting the quiz were successful as it could be done easily. However, some aspects could have been improved.

While both my stakeholders found the interface easy to navigate, they wanted it to be more modern. This is quite important as my program is targeted at students. Having a modern interface would likely make the program experience more enjoyable and would encourage students to use it more. With further development, this could be solved by using rounded shapes or using a different font.

Both users found the program quite repetitive after the first time experiencing it. Abbas thought a skip button could be implemented when displaying images of the apparatus as it only needs to be watched once or twice. Joe thought the screens telling the user whether they have got the answer correct or not were too long, therefore, a skip button could be placed there as well. With further development, I could implement this feature.

Joe stated that noises could be added such as when the ball hits the bottom. This is a nice feature to include as it makes the animation more involving. Since I am already detecting when the ball hits the bottom, I could simply add a line of code which plays a noise.

A usability feature that was successfully met was interaction as both stakeholders enjoyed the use of buttons to control various aspects of the animation. While I was not able to include a slider in the animation, my stakeholders still found the arrow buttons engaging.

Maintenance:

There will be some work to keep the program maintained.

The code is well commented and is documented making it easier for other developers to make changes to it. Since I am using object-orientated programming it will be easy to add additional classes or methods for different practicals, which was requested by my stakeholder.

I feel that I could have improved the readability of my code though. An example is calling a method that redraws the Pygame screen rather than writing out all the objects every time. Moreover, I could have used more classes or methods to improve the modularity of my code. Since I have so many coordinates to keep track of, it can be confusing to know which one is which.

The questions that I have used may not be relevant to the user in a couple of years. Therefore, these questions and answers will need to be updated regularly so that they are tailored to the student's

requirements. This can be done by editing the CSV files – the fact that these files can be viewed in Microsoft Excel makes it easy for anyone to add or remove questions.

MY SOLUTION

```
#A Level Programming Project by  Mayur Shankar

import pygame #--Importing all the required modules
import csv
import time
import sys
import os
import matplotlib.pyplot as plt
import random
import numpy as np
from numpy.polynomial.polynomial import polyfit
pygame.init() #--Initialises pygame

#--Setting up the Tkinter window
from tkinter import*
window = Tk()
window.title("Physics Practical Animation and Quiz")#--Name of the window
window.geometry("1000x700") #--Sets the height and width of the Tkinter window
window.configure(bg="SkyBlue2") #--Sets the background colour of the window
window.resizable(False, False) #--Tkinter window is not resizable

loginTries = 0

class Format:#--Class for all the formatting that will be used throughout the program
  def __init__(self, white, black, blue, background, red, quizred, grey, green,font, bd, fontSizeLarge, fontSizeLargeQuiz, fontSizeMedium, fontSizeSmall):
    self.white = white
    self.black = black
    self.blue = blue
    self.background = background
    self.red = red
    self.quizred = quizred
    self.grey = grey
    self.green = green
    self.font = font
    self.bd = bd
    self.fontSizeLarge = fontSizeLarge
    self.fontSizeLargeQuiz = fontSizeLargeQuiz
    self.fontSizeMedium = fontSizeMedium
    self.fontSizeSmall = fontSizeSmall

form = Format("white", "black", "medium blue", "SkyBlue2", "firebrick1", "IndianRed2" ,"grey40", "seagreen3", "Arial", 1, 20, 17,15, 13)
#--This will be expanded as needed

class Login:#--Initialising the first class that will be used for the login screen
  def __init__(self):#--Key attributes labelled
    self.credentialsList = [] #--List of all the student details to be used later
    #--Tkinter widgets
    self.welcomeText = Label(window, text="Welcome, to get started please log in", fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
    self.accountText = Label(window, text="Don't have an account? Create one!", fg=form.grey,bg=form.white,font=(form.font, form.fontSizeSmall))
    self.username = Label(text="Username:", fg=form.red, bg=form.white, font=(form.font, form.fontSizeMedium))
    self.password = Label(text="Password:", fg=form.blue, bg=form.white, font=(form.font, form.fontSizeMedium))
    self.entryUsername = Entry(window, bd = form.bd, font=(form.font, form.fontSizeSmall))
    self.entryPassword = Entry(window, bd = form.bd, font=(form.font, form.fontSizeSmall), show='*') #--This is where the user will enter their details
    self.submit = Button(window, text="Submit", fg=form.black, bg=form.white,font=(form.font, form.fontSizeMedium), command=self.checkCredentials)
    #--Checks the entered credentials by using the checkCredentials method
    self.createButton = Button(window, text="Create a new account", bd = form.bd, fg = form.black, bg=form.white, font=(form.font, form.fontSizeMedium),
                               command=self.createAccount)
    #--Program will move onto the createAccount method

    #--Widgets used for creating a new account
    self.welcomeText.grid()
    self.createText = Label(text="Please create a new account",fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
    self.newUsername = Label(text="New Username:", fg=form.red, bg=form.white, font=(form.font, form.fontSizeMedium))
    self.newPassword = Label(text="New Password:", fg=form.blue, bg=form.white, font=(form.font, form.fontSizeMedium))
    #---This is where the user will enter their new username and password
    self.entryNewUsername = Entry(window, bd = form.bd, font=(form.font, form.fontSizeSmall))
    self.entryNewPassword = Entry(window, bd = form.bd, font=(form.font, form.fontSizeSmall), show='*')
    self.submitNew = Button(window, text="Submit", fg=form.black, font=(form.font, form.fontSizeMedium), command=self.addCredentials)

    #--Text that will be displayed to cope with erroneous inputs
    self.enterText = Label(window, text="Please enter a valid username/password", fg=form.grey,bg=form.white,font=(form.font, form.fontSizeSmall))
    self.attemptText = Label(window, text="You have attempted more than 5 tries, please wait 30 seconds",
                             fg=form.grey,bg=form.white,font=(form.font, form.fontSizeSmall))
    self.retryText = Label(window, text="Wrong credentials, please try again", fg=form.grey,bg=form.white,font=(form.font, form.fontSizeSmall))

    #--Widget Positioning
    self.welcomeText.grid(row=1, columnspan=9, pady = 100, padx = 280) #--Main widget in the window
    self.accountText.grid(row=5, column=4, pady = 50)
    self.username.grid(row=3, column=3, sticky = E) #--Sticky determines which side the widget will bind to
    self.password.grid(row=4, column=3, sticky = E)
    self.entryUsername.grid(row=3, column=4)
    self.entryPassword.grid(row=4, column=4)
    self.submit.grid(row=4, column = 5, sticky = W)
    self.createButton.grid(row=6, column = 4)

  def createAccount(self): #--Method to create a new user account
    self.accountText.grid_forget() #--Removing widgets that are no longer needed
    self.createButton.grid_forget()
    self.retryText.grid_forget()
    self.welcomeText.grid_forget()

    #--Positioning of widgets as per my screen designs
    self.createText.grid(row=1, columnspan=9, pady = 130, padx = 330)
    self.newUsername.grid(row=3, column=3, sticky = E)
    self.newPassword.grid(row=4, column=3, sticky = E)
    self.entryNewUsername.grid(row=3, column=4)
    self.entryNewPassword.grid(row=4, column=4)
```

```
    self.submitNew.grid(row=4, column = 5, sticky = W)

  def addCredentials(self): #--Appending entered credentials to a CSV file
    #--Part of validation where at least one character has to be entered to be saved
    if len(self.entryNewUsername.get()) == 0 or len(self.entryNewPassword.get()) == 0:
      self.enterText.grid(row=2, column = 4)
      self.entryNewUsername.delete(0, 'end') #--Deletes content in widgets
      self.entryNewPassword.delete(0, 'end')
      command = self.addCredentials #--Repeats Method
    else:
      rows = [[self.entryNewUsername.get(),self.entryNewPassword.get()]] #--Items that will be added
      #--Appending to the file as I want the original contents to be stored
      with open('studentCredentials.csv','a', newline='') as studentCredentials:
        writer = csv.writer(studentCredentials)
        writer.writerows(rows) #--Rows written
        self.newUsername.grid_forget()
        self.newPassword.grid_forget()
        self.enterText.grid_forget()
        self.createText.grid_forget()
        self.retryText.grid_forget()
        self.accountText.grid_forget()
        self.username.grid_forget()
        self.password.grid_forget()
        self.entryUsername.grid_forget()
        self.entryPassword.grid_forget()
        self.welcomeText.grid_forget()
        self.createButton.grid_forget()
        self.submit.grid_forget()
        self.entryNewUsername.grid_forget()
        self.entryNewPassword.grid_forget()
        self.submitNew.grid_forget()
        command = Login() #--Taking user back to main login screen

  def checkCredentials(self): #--Method to check the inputted details
    global loginTries #--Global variable which keeps track of how many attempts the user has had
    if loginTries > 4: #--Giving the user 5 attempts as per my success criteria
      self.retryText.grid_forget()
      self.attemptText.grid(row=2, column=4)
      second=StringVar()
      second.set("30")
      #--Timer that will be displayed
      timeEntryLabel = Label(window, text="30", width=3, bg = form.white, font=(form.font, form.fontSizeLarge,""),textvariable=second)
      timeEntryLabel.grid(row = 1, column = 3)
      self.entryUsername.delete(0, 'end') #--Deletes content in widgets
      self.entryPassword.delete(0, 'end')
      loop = 30 #--Sets the total time in seconds
      while loop >-1: #--Loop allows timer to iterate
        mins,secs = divmod(loop,60)
        second.set("{0:2d}".format(secs))
        second.set("{0:2d}".format(secs))
        window.update() #--Updates window as time changes
        time.sleep(1)
        if (loop == 0):
          timeEntryLabel.grid_forget()
          self.enterText.grid_forget()
          self.attemptText.grid_forget()
          loginTries = 0 #--Resets variable so user can enter their details again
          command = self.removeWidgets #--Sends user back to the login screen
        loop -= 1

    #--Makes sure usernames and passwords with no characters cannot be entered
    if len(self.entryUsername.get()) == 0 or len(self.entryPassword.get()) == 0:
        self.retryText.grid_forget()
        self.enterText.grid(row=2, column = 4)

    else:
      self.correct = False #--Attribute to determine whether access is given
      with open('studentCredentials.csv', 'r') as studentCredentials:
        reader = csv.reader(studentCredentials)
        for row in reader: #--File contents are appended as a 2d array
          self.credentialsList.append(row)

        for x in self.credentialsList: #--Goes through each array in the 2d array
          if self.entryUsername.get() in x and self.entryPassword.get() in x:
            self.accountText.grid_forget()
            self.username.grid_forget()
            self.password.grid_forget()
            self.entryUsername.grid_forget()
            self.entryPassword.grid_forget()
            self.entryNewUsername.grid_forget()
            self.entryNewPassword.grid_forget()
            self.welcomeText.grid_forget()
            self.createButton.grid_forget()
            self.submit.grid_forget()
            self.retryText.grid_forget()
            self.enterText.grid_forget()
            self.correct = True #--Sets attribute to true so access is granted
            command = ChoosePractical()

        if self.correct !=True:
          loginTries += 1 #--Adds one to the loginTries count
          self.credentialsList.clear()
          self.entryUsername.delete(0, 'end') #--Deletes content in widgets
          self.entryPassword.delete(0, 'end')
          self.enterText.grid_forget()
          self.retryText.grid(row = 2, column = 4)
          command = self.removeWidgets #--Sends user back to the login screen
'''
  #--Method which removes widgets from the window when user enters wrong details
  def removeWidgets(self):
    self.retryText.grid_forget()
    self.enterText.grid_forget()
    command = Login() #--Takes user back to the login screen

class ChoosePractical:
  def __init__(self):
    #--Main heading in the window
    self.headingText = Label(window, text="Please choose a practical to view",fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
    self.confirm_Button2 = Button(text="Confirm?",fg=form.black, font=(form.font, form.fontSizeMedium), command = self.practical)
    #--List of options in the drop-down menu
    self.OptionList = ["Click to choose Practical:", "Determination of 'g' by a freefall method"]
    self.menu = StringVar(window) #--Sets up a Tkinter variable
    self.menu.set(self.OptionList[0]) #--First option will be 'Choose Practical'
    self.dropDown = OptionMenu(window, self.menu, *self.OptionList)
    #--Formats and positions the drop-down menu
    self.dropDown.config(width=32, font=(form.font, form.fontSizeMedium))
    self.dropDown.grid(row=3, column=2)
    self.headingText.grid(row=1, columnspan=9, pady = 120, padx = 250)

    def change_dropdown(*args): #--Function which runs when the user clicks an option
      if self.menu.get() == "Determination of 'g' by a freefall method":
        #--Calls 'confirmButton' method
        command = self.confirmButton()
      elif self.menu.get() == "Click to choose Practical:":
        self.confirm_Button2.grid_forget()
    self.menu.trace("w", change_dropdown) #--Allows the function to be run

  def confirmButton(self): #--Allows the user to confirm that they want to view the practical
    self.confirm_Button2.grid(row=3, column=4)

  def practical(self): #--Instantiates the 'PracticalAnimation' class which creates a new window
    window.destroy() #--Destroys old Tkinter window
    command = PracticalAnimation()

class PracticalAttributes: #--Main class to store attributes
  def __init__(self, WHITE, GREY, BLACK, BLUE, ORANGE, PURPLE, RED, fontObj, fontObj_medium, fontObj_mediumsmall, fontObj_small):
    self.WHITE = WHITE #--Examples of colours
    self.GREY = GREY
    self.BLACK = BLACK
    self.BLUE = BLUE
    self.ORANGE = ORANGE
    self.PURPLE = PURPLE
    self.RED = RED
    self.fontObj = fontObj #--Used to set a font
    self.fontObj_medium = fontObj_medium
    self.fontObj_mediumsmall = fontObj_mediumsmall
```

```
    self.fontObj_small = fontObj_small
    self.width = 100 #--Values to set the size of shapes
    self.height = 50
    self.x_beginText = 315 #--Coordinates needed to position objects in the window
    self.y_beginText = 305
    self.x_heading = 100
    self.y_heading = 150
    self.x_beginButton = 310
    self.y_beginButton = 300
    self.m = 1000
    self.m2 = 1000
    self.condition = True #--Allows future loops to be run and broken
    self.condition2 = True
    self.condition_lightgatebutton = True
    self.condition_lightgatebutton2 = True
pract = PracticalAttributes((255,255,255),(220,220,220),(0,0,0),(57,100,240), #--Class is instantiated
                            (255,181,127),(240,0,250),(255,0,0),pygame.font.SysFont('arial', 34), pygame.font.SysFont('arial', 26),
                            pygame.font.SysFont('arial', 21),pygame.font.SysFont('arial', 20))

class Apparatus: #--Class which stores attributes for the apparatus part of the practical
  def __init__(self):
    self.time_delay = 2000 #--Time delays that will be used
    self.time_delay2 = 4000
    self.time_delay3 = 6000
    self.time_delay4 = 8000
    self.time_delay5 = 11000
    #--List of text to be rendered and displayed onscreen using attributes from the 'PracticalAttributes' class
    self.equipment_text = pract.fontObj_medium.render("This is the equipment you will need:", True, pract.BLACK, pract.WHITE)
    self.clampStand_text = pract.fontObj_medium.render("Stand and Clamp", True, pract.BLACK, pract.WHITE)
    self.magnet_text = pract.fontObj_medium.render("Electromagnet", True, pract.BLACK, pract.WHITE)
    self.bearings_text = pract.fontObj_medium.render("Steel ball bearings", True, pract.BLACK, pract.WHITE)
    self.gates_text = pract.fontObj_medium.render("Light Gates", True, pract.BLACK, pract.WHITE)
    self.stopwatch_text = pract.fontObj_medium.render("Stopwatch", True, pract.BLACK, pract.WHITE)
    #--List of images to be projected, saved as jpg files
    self.img_clamp = pygame.image.load('Clamp Stand.jpg')
    self.img_magnet = pygame.image.load('Electromagnet.jpeg')
    self.img_bearings = pygame.image.load('Ball bearings.jpg')
    self.img_gates = pygame.image.load('Light Gates.jpg')
    self.img_stopwatch = pygame.image.load('Stopwatch.jpg')
    #--Setting the overall size of each image
    self.img_clamp = pygame.transform.scale(self.img_clamp, (150,150))
    self.img_magnet = pygame.transform.scale(self.img_magnet, (150,150))
    self.img_bearings = pygame.transform.scale(self.img_bearings, (150,150))
    self.img_gates = pygame.transform.scale(self.img_gates, (150,150))
    self.img_stopwatch = pygame.transform.scale(self.img_stopwatch, (150,150))
    #--Coordinates of the text and images that will be displayed
    self.x_apparatusHeading = 10
    self.y_apparatusHeading = 10
    self.x_apparatusText = 10
    self.x_apparatusPicture = 190
    self.x2_apparatusText = 380
    self.x2_apparatusPicture = 510
    self.y_apparatusRow1 = 60
    self.y_apparatusRow2 = 230
    self.y_apparatusRow3 = 400
apparatus = Apparatus() #--Class is instantiated

class MovingAnimation: #--Class for all the attributes needed to create the animation
  def __init__(self):
    self.x_stand = 70 #--These attributes are for the x and y coordinate of every object in the screen
    self.y_stand = 50
    self.x_base = 10
    self.y_base = 530
    self.x_clamp = 70
    self.y_clamp = 50
    self.x_pad = 150
    self.y_pad = 515
    self.x_counterweight = 15
    self.y_counterweight = 500
    self.x_electromagnet = 175
    self.y_electromagnet = 62
    self.x_ball = 192
    self.y_ball = 96
    self.x_lightgate = 60
    self.y_lightgate = 150
    self.x_lightgate2 = 60
    self.y_lightgate2 = 450
    self.x_lightray = 120
    self.y_lightray = 156
    self.x_lightray2 = 120
    self.y_lightray2 = 456
    self.y_lightgate2_075m = 375
    self.y_lightray2_075m = 381
    self.y_lightgate2_050m = 300
    self.y_lightray2_050m = 306
    self.y_lightgate2_025m = 225
    self.y_lightray2_025m = 231
    self.x_graphButton = 320
    self.y_graphButton = 300
    self.x_slider = 420
    self.y_slider = 400

    #--Since many of my objects are rectangles, I have included a height and width.
    self.width_stand = 12
    self.height_stand = 480
    self.width_base = 250
    self.height_base = 12
    self.width_clamp = 140
    self.height_clamp = 12
    self.width_pad = 80
    self.height_pad = 16
    self.width_counterweight = 40
    self.height_counterweight = 30
    self.width_electromagnet = 35
    self.height_electromagnet = 20
    self.radius_ball = 15
    self.width_lightgate = 50
    self.height_lightgate = 12
    self.width_lightgate2 = 50
    self.height_lightgate2 = 12
    self.width_lightray = 150
    self.height_lightray = 1
    self.width_lightray2  = 150
    self.height_lightray2 = 1
    self.width_graphButton = 150
    self.height_graphButton = 150
    self.width_slider = 150
    self.height_slider = 150

    #--Labels for objects in the screen
    self.clampStand_label = pract.fontObj_small.render("Clamp Stand", True, pract.BLACK, pract.WHITE)
    self.counterweight_label = pract.fontObj_small.render("Counterweight", True, pract.BLACK, pract.WHITE)
    self.lightgates_label = pract.fontObj_small.render("Light gate", True, pract.BLACK, pract.WHITE)
    self.magnet_label = pract.fontObj_small.render("Electromagnet connected to low voltage DC supply", True, pract.BLACK, pract.WHITE)
    self.bearings_label = pract.fontObj_small.render("Ball bearing", True, pract.BLACK, pract.WHITE)
    self.press_spacebar_label = pract.fontObj_mediumsmall.render("Press the spacebar to drop the ball.", True, pract.BLACK, pract.WHITE)
    self.repeat_spacebar_label = pract.fontObj_mediumsmall.render("Press the spacebar again to repeat.", True, pract.BLACK, pract.WHITE)

    #--X and Y coordinates for every label
    self.x_clampStandlabel = 30
    self.y_clampStandlabel = 300
    self.y_clampStandlabel_075m = 230
    self.x_counterweightlabel = 15
    self.y_counterweightlabel = 550
    self.x_lightgateslabel = 50
    self.y_lightgateslabel = 120
    self.x_lightgateslabel2 = 50
    self.y_lightgateslabel2 = 420
    self.y_lightgateslabel2_075m = 345
    self.y_lightgateslabel2_050m = 270
    self.y_lightgateslabel2_025m = 190
    self.x_magnetlabel = 165
    self.y_magnetlabel = 20
    self.x_bearingslabel = 210
    self.y_bearingslabel = 86
    self.x_press_spacebarlabel = 400
```

```
    self.x_press_spacebarlabel = 400
    self.y_press_spacebarlabel = 550
    self.x_displaytimer = 390
    self.y_displaytimer = 430
    self.x_timerresult = 460
    self.y_timerresult = 470

    #--Attributes for the motion of the ball
    self.pressed = False
    self.intial_motion = 0
    self.acceleration = 0
    self.acceleration2 = 1

    #--Attributes for the buttons to change the coordinates of the light gates
    self.height_05_text = pract.fontObj_mediumsmall.render("0.5 metres", True, pract.BLACK, pract.GREY)
    self.chooseheight = 0
    self.x_height_1m_text = 556
    self.y_height_1m_text = 140
    self.x_height_075m_text = 545
    self.y_height_075m_text = 140
    self.x_height_050m_text = 545
    self.y_height_050m_text = 140
    self.x_height_025m_text = 545
    self.y_height_025m_text = 140
    self.coordinates_upbutton = ((560,120),(590,90), (620,120))
    self.coordinates_downbutton = ((560,180),(590,210), (620,180))

    #--Attributes for displaying the graph
    self.graphTime = []
    self.graphHeight = []
    self.height_05_text = pract.fontObj_mediumsmall.render("0.5 metres", True, pract.BLACK, pract.GREY)
    self.x_graphButton = 555
    self.y_graphButton = 243
    self.x_graphButtonText = 565
    self.y_graphButtonText = 250
    self.width_graphButton = 110
    self.height_graphButton = 40
    self.graphButtonText = pract.fontObj_mediumsmall.render("View Graph", True, pract.BLACK, pract.WHITE)
    self.countRepeats = 0
    self.repeatText = pract.fontObj_mediumsmall.render(" Repeat the experiment with different heights! ", True, pract.BLACK, pract.WHITE)
    self.x_repeatText = 330
    self.y_repeatText = 370

    #--So the user can exit the Pygame window
    self.x_exitButton = 555
    self.y_exitButton = 300
    self.x_exitButtonText = 595
    self.y_exitButtonText = 307
    self.width_exitButton = 110
    self.height_exitButton = 40
    self.exitButtonText = pract.fontObj_mediumsmall.render("Exit", True, pract.BLACK, pract.WHITE)
#--Class is instantiated
mov = MovingAnimation()

class PracticalAnimation:
  def __init__(self):
    win = pygame.display.set_mode((700, 600)) #--Sets the size of the Pygame window
    pygame.display.set_caption('Practical Animation') #--Gives the window a name
    win.fill(pract.ORANGE) #--Allows a background colour to be set
    #--First heading in the window
    heading_text = pract.fontObj.render(" Determination of 'g' by a freefall method ", True, pract.BLACK, pract.WHITE)
    begin_text = pract.fontObj.render(' Begin ', True, pract.BLACK)
    #--Allows the user to start the animation
    beginButton = pygame.draw.rect(win, pract.WHITE, (pract.x_beginButton, pract.y_beginButton, pract.width, pract.height))
    #--Draws the text onscreen
    win.blit(heading_text, (pract.x_heading, pract.y_heading))
    win.blit(begin_text, (pract.x_beginText, pract.y_beginText))
    #--Draws border around the text
    beginButton_border = pygame.draw.rect(win, pract.BLACK, (pract.x_beginButton, pract.y_beginButton, pract.width, pract.height),1)

    #--Starts a while loop which repeatedly checks the user's mouse position
    while pract.condition == True:
      pos = pygame.mouse.get_pos()
      #--Boundaries the user's mouse position has to be in for the button to be pressed
      if pos[0] > 310 and pos[0]< 390 and pos[1] > 300 and pos[1]< 350:
        event = pygame.event.poll()
        if event.type == pygame.MOUSEBUTTONDOWN:
          pract.condition = False #--Breaks the while loop
          win.fill(pract.ORANGE) #--New backgroud which replaces everything onscreen
          previous_time = pygame.time.get_ticks() #--Measures current time
          while pract.condition2 == True: #--Starts another while loop
            win.fill(pract.ORANGE)
            #--Draws the first set of text and images onscreen
            win.blit(apparatus.equipment_text, (apparatus.x_apparatusHeading, apparatus.y_apparatusHeading))
            win.blit(apparatus.clampStand_text, (apparatus.x_apparatusText, apparatus.y_apparatusRow1))
            win.blit(apparatus.img_clamp, (apparatus.x_apparatusPicture, apparatus.y_apparatusRow1))
            clampStand_rect = pygame.draw.rect(win, pract.BLACK, (apparatus.x_apparatusPicture, apparatus.y_apparatusRow1, 150,150),1)
            current_time = pygame.time.get_ticks() #--Measures the current time relative to the previous time
            if current_time - previous_time > apparatus.time_delay: #--Decides when the next batch of text and images should be displayed
              #--Next batch of text and images are displayed when the if statement is fulfilled
              win.blit(apparatus.equipment_text, (apparatus.x_apparatusHeading, apparatus.y_apparatusHeading))
              win.blit(apparatus.magnet_text, (apparatus.x_apparatusText, apparatus.y_apparatusRow2))
              win.blit(apparatus.img_magnet, (apparatus.x_apparatusPicture, apparatus.y_apparatusRow2))
              magnet_rect = pygame.draw.rect(win, pract.BLACK, (apparatus.x_apparatusPicture, apparatus.y_apparatusRow2, 150,150),1)
              current_time = pygame.time.get_ticks() #--Measures the current time relative to the previous time
              #--Larger time gap means a larger time delay is used
              if current_time - previous_time > apparatus.time_delay2:
                win.blit(apparatus.equipment_text, (apparatus.x_apparatusHeading, apparatus.y_apparatusHeading))
                win.blit(apparatus.bearings_text, (apparatus.x_apparatusText, apparatus.y_apparatusRow3))
                win.blit(apparatus.img_bearings, (apparatus.x_apparatusPicture, apparatus.y_apparatusRow3))
                bearings_rect = pygame.draw.rect(win, pract.BLACK, (apparatus.x_apparatusPicture, apparatus.y_apparatusRow3, 150,150),1)
                current_time = pygame.time.get_ticks()
                #--Process iterates
                if current_time - previous_time > apparatus.time_delay3:
                  win.blit(apparatus.equipment_text, (apparatus.x_apparatusHeading, apparatus.y_apparatusHeading))
                  win.blit(apparatus.gates_text, (apparatus.x2_apparatusText, apparatus.y_apparatusRow1))
                  win.blit(apparatus.img_gates, (apparatus.x2_apparatusPicture, apparatus.y_apparatusRow1))
                  gates_rect = pygame.draw.rect(win, pract.BLACK, (apparatus.x2_apparatusPicture, apparatus.y_apparatusRow1, 150,150),1)
                  current_time = pygame.time.get_ticks()
                  if current_time - previous_time > apparatus.time_delay4:
                    win.blit(apparatus.equipment_text, (apparatus.x_apparatusHeading, apparatus.y_apparatusHeading))
                    win.blit(apparatus.stopwatch_text, (apparatus.x2_apparatusText, apparatus.y_apparatusRow2))
                    win.blit(apparatus.img_stopwatch, (apparatus.x2_apparatusPicture, apparatus.y_apparatusRow2))
                    stopwatch_rect = pygame.draw.rect(win, pract.BLACK, (apparatus.x2_apparatusPicture, apparatus.y_apparatusRow2, 150,150),1)
                    current_time = pygame.time.get_ticks()
                    #--Waits some time before transitioning the screen
                    if current_time - previous_time > apparatus.time_delay5:
                      command = self.AnimationInteraction() #--Calls the next method

            #--While the program is running, checks whether the user is quitting the window
            for event in pygame.event.get():
              if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            pygame.display.update() #--Updates the display
      #--Since there are two while loops, there needs to be two lots of this code
      for event in pygame.event.get():
          if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
      pygame.display.update() #--Updates the display

  def AnimationInteraction(self):
    win = pygame.display.set_mode((700, 600))
    win.fill(pract.ORANGE)
    while pract.condition == False:
      keys = pygame.key.get_pressed() #--Detects user's key inputs
      if not (mov.pressed): #--Only runs when mov.pressed is False
        if keys[pygame.K_SPACE]:
          mov.pressed = True #--If true, on the next loop, the program won't check whether the space key is being pressed
        else:
          win.fill(pract.ORANGE)
          stand = pygame.draw.rect(win, pract.GREY,(mov.x_stand, mov.y_stand, mov.width_stand, mov.height_stand))
          #--An identical rectangle is created for the border
          stand = pygame.draw.rect(win, pract.BLACK,(mov.x_stand, mov.y_stand, mov.width_stand, mov.height_stand), 1)
          #--Rectangles for the base of the stand
```

```
        base = pygame.draw.rect(win, pract.GREY,(mov.x_base, mov.y_base, mov.width_base, mov.height_base))
        base = pygame.draw.rect(win, pract.BLACK,(mov.x_base, mov.y_base, mov.width_base, mov.height_base), 1)
        #--Rectangles for the top of the screen
        clamp = pygame.draw.rect(win, pract.GREY,(mov.x_clamp, mov.y_clamp, mov.width_clamp, mov.height_clamp))
        clamp = pygame.draw.rect(win, pract.BLACK,(mov.x_clamp, mov.y_clamp, mov.width_clamp, mov.height_clamp), 1)
        #--Rectangles for the pad and counterweight
        pad = pygame.draw.rect(win, pract.GREY,(mov.x_pad, mov.y_pad, mov.width_pad, mov.height_pad))
        pad = pygame.draw.rect(win, pract.BLACK,(mov.x_pad, mov.y_pad, mov.width_pad, mov.height_pad), 1)
        counterweight = pygame.draw.rect(win, pract.GREY,(mov.x_counterweight, mov.y_counterweight, mov.width_counterweight, mov.height_counterweight))
        counterweight = pygame.draw.rect(win, pract.BLACK,(mov.x_counterweight, mov.y_counterweight, mov.width_counterweight, mov.height_counterweight), 1)
        #--Drawing light gates as rectangles
        lightgate = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate))
        lightgate = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate), 1)
        lightgate2 = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate2, mov.y_lightgate2, mov.width_lightgate2, mov.height_lightgate2))
        lightgate2 = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate2, mov.y_lightgate2, mov.width_lightgate2, mov.height_lightgate2),1)
        #--Treating these as rectangles but with very small heights
        lightray = pygame.draw.rect(win, pract.RED,(mov.x_lightray, mov.y_lightray, mov.width_lightray, mov.height_lightray))
        lightray2 = pygame.draw.rect(win, pract.RED,(mov.x_lightray2, mov.y_lightray2, mov.width_lightray2, mov.height_lightray2))
        #--Drawing electromagnets as rectangles
        electromagnet = pygame.draw.rect(win, pract.GREY,(mov.x_electromagnet, mov.y_electromagnet, mov.width_electromagnet,mov.height_electromagnet))
        electromagnet = pygame.draw.rect(win, pract.BLACK,(mov.x_electromagnet, mov.y_electromagnet, mov.width_electromagnet,mov.height_electromagnet), 1)

        #--Drawing to the screen the various labels at the specified x and y coordinates
        win.blit(mov.clampStand_label, (mov.x_clampStandlabel, mov.y_clampStandlabel))
        win.blit(mov.counterweight_label, (mov.x_counterweightlabel, mov.y_counterweightlabel))
        win.blit(mov.lightgates_label, (mov.x_lightgateslabel, mov.y_lightgateslabel))
        win.blit(mov.lightgates_label, (mov.x_lightgateslabel2, mov.y_lightgateslabel2))
        win.blit(mov.magnet_label, (mov.x_magnetlabel, mov.y_magnetlabel))
        win.blit(mov.bearings_label, (mov.x_bearingslabel, mov.y_bearingslabel))
        win.blit(mov.press_spacebar_label, (mov.x_press_spacebarlabel, mov.y_press_spacebarlabel))

        #--Will be drawn as a circle
        ball_bearing = pygame.draw.circle(win, pract.GREY,(mov.x_ball, mov.y_ball), mov.radius_ball)
        ball_bearing = pygame.draw.circle(win, pract.BLACK,(mov.x_ball, mov.y_ball), mov.radius_ball,1)

        #--Buttons the user can press to change the position of the light gates
        increaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_upbutton,2)
        decreaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_downbutton,2)
        height_1m_text = pract.fontObj_mediumsmall.render(' 1 Metre ', True, pract.BLACK, pract.WHITE)
        win.blit(height_1m_text, (mov.x_height_1m_text, mov.y_height_1m_text))

        #--Detects whether this button has been pressed
        while pract.condition_lightgatebutton == True:
          keys = pygame.key.get_pressed() #--Receives any key press
          pos = pygame.mouse.get_pos() #--Receives any mouse press
          #--Boundaries the user's mouse position has to be in for the bottom button to be pressed
          if pos[0] > 560 and pos[0]< 620  and pos[1] > 160 and pos[1]< 230:
            event = pygame.event.poll()
            #--The mouse button has to also be pressed
            #--the mouse button has to also be pressed
            if event.type == pygame.MOUSEBUTTONDOWN:
              mov.chooseheight -=1
              #--Protects against repeated clicking of the button
              if mov.chooseheight < -3:
                mov.chooseheight = -3
              #--If the button is pressed once, the diagram for a height of 0.75m will be displayed.
              if mov.chooseheight == -1:
                win.fill(pract.ORANGE)
                stand = pygame.draw.rect(win, pract.GREY,(mov.x_stand, mov.y_stand, mov.width_stand, mov.height_stand))
                #--An identical rectangle is created for the border
                stand = pygame.draw.rect(win, pract.BLACK,(mov.x_stand, mov.y_stand, mov.width_stand, mov.height_stand), 1)
                #--Rectangles for the base of the stand
                base = pygame.draw.rect(win, pract.GREY,(mov.x_base, mov.y_base, mov.width_base, mov.height_base))
                base = pygame.draw.rect(win, pract.BLACK,(mov.x_base, mov.y_base, mov.width_base, mov.height_base), 1)
                #--Rectangles for the top of the screen
                clamp = pygame.draw.rect(win, pract.GREY,(mov.x_clamp, mov.y_clamp, mov.width_clamp, mov.height_clamp))
                clamp = pygame.draw.rect(win, pract.BLACK,(mov.x_clamp, mov.y_clamp, mov.width_clamp, mov.height_clamp), 1)
                #--Rectangles for the pad and counterweight
                pad = pygame.draw.rect(win, pract.GREY,(mov.x_pad, mov.y_pad, mov.width_pad, mov.height_pad))
                pad = pygame.draw.rect(win, pract.BLACK,(mov.x_pad, mov.y_pad, mov.width_pad, mov.height_pad), 1)
                counterweight = pygame.draw.rect(win, pract.GREY,(mov.x_counterweight, mov.y_counterweight, mov.width_counterweight, mov.height_counterweight))
                counterweight = pygame.draw.rect(win, pract.BLACK,(mov.x_counterweight, mov.y_counterweight, mov.width_counterweight, mov.height_counterweight), 1)
                #--Drawing electromagnets as rectangles
                electromagnet = pygame.draw.rect(win, pract.GREY,(mov.x_electromagnet, mov.y_electromagnet, mov.width_electromagnet,mov.height_electromagnet))
                electromagnet = pygame.draw.rect(win, pract.BLACK,(mov.x_electromagnet, mov.y_electromagnet, mov.width_electromagnet,mov.height_electromagnet), 1)
                #--Will be drawn as a circle
                ball_bearing = pygame.draw.circle(win, pract.GREY,(mov.x_ball, mov.y_ball), mov.radius_ball)
                ball_bearing = pygame.draw.circle(win, pract.BLACK,(mov.x_ball, mov.y_ball), mov.radius_ball,1)

                #--Drawing to the screen the various labels at the specified x and y coordinates
                win.blit(mov.counterweight_label, (mov.x_counterweightlabel, mov.y_counterweightlabel))
                win.blit(mov.lightgates_label, (mov.x_lightgateslabel, mov.y_lightgateslabel))
                win.blit(mov.lightgates_label, (mov.x_lightgateslabel2, mov.y_lightgateslabel2_075m))
                win.blit(mov.magnet_label, (mov.x_magnetlabel, mov.y_magnetlabel))
                win.blit(mov.bearings_label, (mov.x_bearingslabel, mov.y_bearingslabel))
                win.blit(mov.press_spacebar_label, (mov.x_press_spacebarlabel, mov.y_press_spacebarlabel))

                #--Drawing light gates as rectangles
                lightgate = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate))
                lightgate = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate), 1)
                lightgate2_075m = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate2, mov.y_lightgate2_075m, mov.width_lightgate2, mov.height_lightgate2))
                lightgate2_075m= pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate2, mov.y_lightgate2_075m, mov.width_lightgate2, mov.height_lightgate2), 1)

                #--Treating these as rectangles but with very small heights
                lightray = pygame.draw.rect(win, pract.RED,(mov.x_lightray, mov.y_lightray, mov.width_lightray, mov.height_lightray))
                lightray2_075m = pygame.draw.rect(win, pract.RED,(mov.x_lightray2, mov.y_lightray2_075m, mov.width_lightray2, mov.height_lightray2))

                #--Buttons the user can press to change the position of the light gates
                increaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_upbutton,2)
                decreaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_downbutton,2)
                height_075m_text = pract.fontObj_mediumsmall.render(' 0.75 Metres ', True, pract.BLACK, pract.WHITE)
                win.blit(height_075m_text, (mov.x_height_075m_text, mov.y_height_075m_text))

              #--If the button is pressed twice, the diagram for a height of 0.50m will be displayed.
              if mov.chooseheight == -2:
                win.fill(pract.ORANGE)
                stand = pygame.draw.rect(win, pract.GREY,(mov.x_stand, mov.y_stand, mov.width_stand, mov.height_stand))
                #--An identical rectangle is created for the border
                stand = pygame.draw.rect(win, pract.BLACK,(mov.x_stand, mov.y_stand, mov.width_stand, mov.height_stand), 1)
                #--Rectangles for the base of the stand
                base = pygame.draw.rect(win, pract.GREY,(mov.x_base, mov.y_base, mov.width_base, mov.height_base))
                base = pygame.draw.rect(win, pract.BLACK,(mov.x_base, mov.y_base, mov.width_base, mov.height_base), 1)
                #--Rectangles for the top of the screen
                clamp = pygame.draw.rect(win, pract.GREY,(mov.x_clamp, mov.y_clamp, mov.width_clamp, mov.height_clamp))
                clamp = pygame.draw.rect(win, pract.BLACK,(mov.x_clamp, mov.y_clamp, mov.width_clamp, mov.height_clamp), 1)
                #--Rectangles for the pad and counterweight
                pad = pygame.draw.rect(win, pract.GREY,(mov.x_pad, mov.y_pad, mov.width_pad, mov.height_pad))
                pad = pygame.draw.rect(win, pract.BLACK,(mov.x_pad, mov.y_pad, mov.width_pad, mov.height_pad), 1)
                counterweight = pygame.draw.rect(win, pract.GREY,(mov.x_counterweight, mov.y_counterweight, mov.width_counterweight, mov.height_counterweight))
                counterweight = pygame.draw.rect(win, pract.BLACK,(mov.x_counterweight, mov.y_counterweight, mov.width_counterweight, mov.height_counterweight), 1)
                #--Drawing electromagnets as rectangles
                electromagnet = pygame.draw.rect(win, pract.GREY,(mov.x_electromagnet, mov.y_electromagnet, mov.width_electromagnet,mov.height_electromagnet))
                electromagnet = pygame.draw.rect(win, pract.BLACK,(mov.x_electromagnet, mov.y_electromagnet, mov.width_electromagnet,mov.height_electromagnet), 1)
                #--Will be drawn as a circle
                ball_bearing = pygame.draw.circle(win, pract.GREY,(mov.x_ball, mov.y_ball), mov.radius_ball)
                ball_bearing = pygame.draw.circle(win, pract.BLACK,(mov.x_ball, mov.y_ball), mov.radius_ball,1)
                #--Drawing to the screen the various labels at the specified x and y coordinates
                win.blit(mov.counterweight_label, (mov.x_counterweightlabel, mov.y_counterweightlabel))
                win.blit(mov.lightgates_label, (mov.x_lightgateslabel, mov.y_lightgateslabel))
                #--New position of the light gate label
                win.blit(mov.lightgates_label, (mov.x_lightgateslabel2, mov.y_lightgateslabel2_050m))
                win.blit(mov.magnet_label, (mov.x_magnetlabel, mov.y_magnetlabel))
                win.blit(mov.bearings_label, (mov.x_bearingslabel, mov.y_bearingslabel))
                win.blit(mov.press_spacebar_label, (mov.x_press_spacebarlabel, mov.y_press_spacebarlabel))

                lightgate = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate))
                lightgate = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate), 1)
                #--New coordinates for the light gates
                lightgate2_050m = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate2, mov.y_lightgate2_050m, mov.width_lightgate2, mov.height_lightgate2))
                lightgate2_050m= pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate2, mov.y_lightgate2_050m, mov.width_lightgate2, mov.height_lightgate2), 1)

                #--Treating these as rectangles but with very small heights
                lightray = pygame.draw.rect(win, pract.RED,(mov.x_lightray, mov.y_lightray, mov.width_lightray, mov.height_lightray))
                lightray2_050m = pygame.draw.rect(win, pract.RED,(mov.x_lightray2, mov.y_lightray2_050m, mov.width_lightray2, mov.height_lightray2))

                #--Buttons the user can press to change the position of the light gates
```

```
                #--Buttons the user can press to change the position of the light gates
                increaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_upbutton,2)
                decreaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_downbutton,2)
                #--New label for the height of 0.5 metres
                height_050m_text = pract.fontObj_mediumsmall.render(' 0.5 Metres ', True, pract.BLACK, pract.WHITE)
                win.blit(height_050m_text, (mov.x_height_050m_text, mov.y_height_050m_text))

              #--If the button is pressed thrice, the diagram for a height of 0.25m will be displayed.
              if mov.chooseheight == -3:
                win.fill(pract.ORANGE)
                stand = pygame.draw.rect(win, pract.GREY,(mov.x_stand, mov.y_stand, mov.width_stand, mov.height_stand))
                #--An identical rectangle is created for the border
                stand = pygame.draw.rect(win, pract.BLACK,(mov.x_stand, mov.y_stand, mov.width_stand, mov.height_stand), 1)
                #--Rectangles for the base of the stand
                base = pygame.draw.rect(win, pract.GREY,(mov.x_base, mov.y_base, mov.width_base, mov.height_base))
                base = pygame.draw.rect(win, pract.BLACK,(mov.x_base, mov.y_base, mov.width_base, mov.height_base), 1)
                #--Rectangles for the top of the screen
                clamp = pygame.draw.rect(win, pract.GREY,(mov.x_clamp, mov.y_clamp, mov.width_clamp, mov.height_clamp))
                clamp = pygame.draw.rect(win, pract.BLACK,(mov.x_clamp, mov.y_clamp, mov.width_clamp, mov.height_clamp), 1)
                #--Rectangles for the pad and counterweight
                pad = pygame.draw.rect(win, pract.GREY,(mov.x_pad, mov.y_pad, mov.width_pad, mov.height_pad))
                pad = pygame.draw.rect(win, pract.BLACK,(mov.x_pad, mov.y_pad, mov.width_pad, mov.height_pad), 1)
                counterweight = pygame.draw.rect(win, pract.GREY,(mov.x_counterweight, mov.y_counterweight, mov.width_counterweight, mov.height_counterweight))
                counterweight = pygame.draw.rect(win, pract.BLACK,(mov.x_counterweight, mov.y_counterweight, mov.width_counterweight, mov.height_counterweight), 1)
                #--Drawing electromagnets as rectangles
                electromagnet = pygame.draw.rect(win, pract.GREY,(mov.x_electromagnet, mov.y_electromagnet, mov.width_electromagnet,mov.height_electromagnet))
                electromagnet = pygame.draw.rect(win, pract.BLACK,(mov.x_electromagnet, mov.y_electromagnet, mov.width_electromagnet,mov.height_electromagnet), 1)
                #--Will be drawn as a circle
                ball_bearing = pygame.draw.circle(win, pract.GREY,(mov.x_ball, mov.y_ball), mov.radius_ball)
                ball_bearing = pygame.draw.circle(win, pract.BLACK,(mov.x_ball, mov.y_ball), mov.radius_ball,1)
                #--Drawing to the screen the various labels at the specified x and y coordinates
                win.blit(mov.counterweight_label, (mov.x_counterweightlabel, mov.y_counterweightlabel))
                win.blit(mov.lightgates_label, (mov.x_lightgateslabel, mov.y_lightgateslabel))
                #--New position of the light gate label
                win.blit(mov.lightgates_label, (mov.x_lightgateslabel2, mov.y_lightgateslabel2_025m))
                win.blit(mov.magnet_label, (mov.x_magnetlabel, mov.y_magnetlabel))
                win.blit(mov.bearings_label, (mov.x_bearingslabel, mov.y_bearingslabel))
                win.blit(mov.press_spacebar_label, (mov.x_press_spacebarlabel, mov.y_press_spacebarlabel))

                lightgate = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate))
                lightgate = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate), 1)
                #--New coordinates for the light gate
                lightgate2_025m = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate2, mov.y_lightgate2_025m, mov.width_lightgate2, mov.height_lightgate2))
                lightgate2_025m= pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate2, mov.y_lightgate2_025m, mov.width_lightgate2, mov.height_lightgate2), 1)
                #--Treating these as rectangles but with very small heights
                lightray = pygame.draw.rect(win, pract.RED,(mov.x_lightray, mov.y_lightray, mov.width_lightray, mov.height_lightray))
                lightray2_025m = pygame.draw.rect(win, pract.RED,(mov.x_lightray2, mov.y_lightray2_025m, mov.width_lightray2, mov.height_lightray2))

                #--Buttons the user can press to change the position of the light gates
                increaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_upbutton,2)
                decreaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_downbutton,2)
                #--New label for the height of 0.25 metres
                height_025m_text = pract.fontObj_mediumsmall.render(' 0.25 Metres ', True, pract.BLACK, pract.WHITE)
                win.blit(height_025m_text, (mov.x_height_025m_text, mov.y_height_025m_text))

            if pos[0] > 560 and pos[0]< 620  and pos[1] > 70 and pos[1]< 140: #--Boundaries for the up button
              event = pygame.event.poll()
              #--The mouse button has to also be pressed
              if event.type == pygame.MOUSEBUTTONDOWN:
                mov.chooseheight += 1 #--Will increment this attribute by 1
                if mov.chooseheight > 0:
                  mov.chooseheight = 0 #--Protects against repeated clicking of the up button
                else:
                  mov.chooseheight = 0
                  #--Displays window with new objects
                  win.fill(pract.ORANGE)
                  stand = pygame.draw.rect(win, pract.GREY,(mov.x_stand, mov.y_stand, mov.width_stand, mov.height_stand))
                  #--An identical rectangle is created for the border
                  stand = pygame.draw.rect(win, pract.BLACK,(mov.x_stand, mov.y_stand, mov.width_stand, mov.height_stand), 1)
                  #--Rectangles for the base of the stand
                  base = pygame.draw.rect(win, pract.GREY,(mov.x_base, mov.y_base, mov.width_base, mov.height_base))
                  base = pygame.draw.rect(win, pract.BLACK,(mov.x_base, mov.y_base, mov.width_base, mov.height_base), 1)
                  #--Rectangles for the top of the screen
                  clamp = pygame.draw.rect(win, pract.GREY,(mov.x_clamp, mov.y_clamp, mov.width_clamp, mov.height_clamp))
                  clamp = pygame.draw.rect(win, pract.BLACK,(mov.x_clamp, mov.y_clamp, mov.width_clamp, mov.height_clamp), 1)
                  #--Rectangles for the pad and counterweight
                  pad = pygame.draw.rect(win, pract.GREY,(mov.x_pad, mov.y_pad, mov.width_pad, mov.height_pad))
                  pad = pygame.draw.rect(win, pract.BLACK,(mov.x_pad, mov.y_pad, mov.width_pad, mov.height_pad), 1)
                  counterweight = pygame.draw.rect(win, pract.GREY,(mov.x_counterweight, mov.y_counterweight, mov.width_counterweight, mov.height_counterweight))
                  counterweight = pygame.draw.rect(win, pract.BLACK,(mov.x_counterweight, mov.y_counterweight, mov.width_counterweight, mov.height_counterweight), 1)
                  #--Drawing light gates as rectangles
                  lightgate = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate))
                  lightgate = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate), 1)
                  lightgate2 = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate2, mov.y_lightgate2, mov.width_lightgate2, mov.height_lightgate2))
                  lightgate2 = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate2, mov.y_lightgate2, mov.width_lightgate2, mov.height_lightgate2),1)
                  #--Treating these as rectangles but with very small heights
                  lightray = pygame.draw.rect(win, pract.RED,(mov.x_lightray, mov.y_lightray, mov.width_lightray, mov.height_lightray))
                  lightray2 = pygame.draw.rect(win, pract.RED,(mov.x_lightray2, mov.y_lightray2, mov.width_lightray2, mov.height_lightray2))
                  #--Drawing electromagnets as rectangles
                  electromagnet = pygame.draw.rect(win, pract.GREY,(mov.x_electromagnet, mov.y_electromagnet, mov.width_electromagnet,mov.height_electromagnet))
                  electromagnet = pygame.draw.rect(win, pract.BLACK,(mov.x_electromagnet, mov.y_electromagnet, mov.width_electromagnet,mov.height_electromagnet), 1)

                  #--Drawing to the screen the various labels at the specified x and y coordinates
                  win.blit(mov.clampStand_label, (mov.x_clampStandlabel, mov.y_clampStandlabel))
                  win.blit(mov.counterweight_label, (mov.x_counterweightlabel, mov.y_counterweightlabel))
                  win.blit(mov.lightgates_label, (mov.x_lightgateslabel, mov.y_lightgateslabel))
                  win.blit(mov.lightgates_label, (mov.x_lightgateslabel2, mov.y_lightgateslabel2))
                  win.blit(mov.magnet_label, (mov.x_magnetlabel, mov.y_magnetlabel))
                  win.blit(mov.bearings_label, (mov.x_bearingslabel, mov.y_bearingslabel))
                  win.blit(mov.press_spacebar_label, (mov.x_press_spacebarlabel, mov.y_press_spacebarlabel))

                  #--Will be drawn as a circle
                  ball_bearing = pygame.draw.circle(win, pract.GREY,(mov.x_ball, mov.y_ball), mov.radius_ball)
                  ball_bearing = pygame.draw.circle(win, pract.BLACK,(mov.x_ball, mov.y_ball), mov.radius_ball,1)

                  #--Buttons the user can press to change the position of the light gates
                  increaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_upbutton,2)
                  decreaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_downbutton,2)
                  height_1m_text = pract.fontObj_mediumsmall.render(' 1 Metre ', True, pract.BLACK, pract.WHITE)
                  win.blit(height_1m_text, (mov.x_height_1m_text, mov.y_height_1m_text))

              if keys[pygame.K_SPACE]: #--Checks whether the space bar has been pressed
                pract.condition_lightgatebutton = False #--If so, this while loop will be ended

              #--While the program is running, checks whether the user is quitting the window
              for event in pygame.event.get():
                if event.type == pygame.QUIT:
                  pygame.quit()
                  sys.exit()
              pygame.display.update() #--Updates the display

            else: #--Previous if not statement is bypassed
              win.fill(pract.ORANGE)
              #--Practical diagram is redrawn
              stand = pygame.draw.rect(win, pract.GREY,(mov.x_stand, mov.y_stand, mov.width_stand, mov.height_stand))
              #--An identical rectangle is created for the border
              stand = pygame.draw.rect(win, pract.BLACK,(mov.x_stand, mov.y_stand, mov.width_stand, mov.height_stand), 1)
              #--Rectangles for the base of the stand
              base = pygame.draw.rect(win, pract.GREY,(mov.x_base, mov.y_base, mov.width_base, mov.height_base))
              base = pygame.draw.rect(win, pract.BLACK,(mov.x_base, mov.y_base, mov.width_base, mov.height_base), 1)
              #--Rectangles for the top of the stand
              clamp = pygame.draw.rect(win, pract.GREY,(mov.x_clamp, mov.y_clamp, mov.width_clamp, mov.height_clamp))
              clamp = pygame.draw.rect(win, pract.BLACK,(mov.x_clamp, mov.y_clamp, mov.width_clamp, mov.height_clamp), 1)
              #--Rectangles for the pad and counterweight
              pad = pygame.draw.rect(win, pract.GREY,(mov.x_pad, mov.y_pad, mov.width_pad, mov.height_pad))
              pad = pygame.draw.rect(win, pract.BLACK,(mov.x_pad, mov.y_pad, mov.width_pad, mov.height_pad), 1)
              counterweight = pygame.draw.rect(win, pract.GREY,(mov.x_counterweight, mov.y_counterweight, mov.width_counterweight, mov.height_counterweight))
              counterweight = pygame.draw.rect(win, pract.BLACK,(mov.x_counterweight, mov.y_counterweight, mov.width_counterweight, mov.height_counterweight), 1)

              if mov.chooseheight == 0: #--When the up or down button is not pressed
                #--Drawing the original light gates
                lightgate = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate))
                lightgate = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate), 1)
                lightgate2 = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate2, mov.y_lightgate2, mov.width_lightgate2, mov.height_lightgate2))
                lightgate2 = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate2, mov.y_lightgate2, mov.width_lightgate2, mov.height_lightgate2),1)
                #--Treating these as rectangles but with very small heights
```

```
        lightray = pygame.draw.rect(win, pract.RED,(mov.x_lightray, mov.y_lightray, mov.width_lightray, mov.height_lightray))
        lightray2 = pygame.draw.rect(win, pract.RED,(mov.x_lightray2, mov.y_lightray2, mov.width_lightray2, mov.height_lightray2))
        #--Drawing light gate label
        win.blit(mov.lightgates_label, (mov.x_lightgateslabel2, mov.y_lightgateslabel2))
        #--Drawing clamp stand label
        win.blit(mov.clampStand_label, (mov.x_clampStandlabel, mov.y_clampStandlabel))

      if mov.chooseheight == -1: #--When the down button is pressed once
        #--Drawing the new light gates corresponding to a height of 0.75 metres
        lightgate = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate))
        lightgate = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate), 1)
        lightgate2_075m = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate2, mov.y_lightgate2_075m, mov.width_lightgate2, mov.height_lightgate2))
        lightgate2_075m = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate2, mov.y_lightgate2_075m, mov.width_lightgate2, mov.height_lightgate2), 1)
        #--Treating these as rectangles but with very small heights
        lightray = pygame.draw.rect(win, pract.RED,(mov.x_lightray, mov.y_lightray, mov.width_lightray, mov.height_lightray))
        lightray2_075m = pygame.draw.rect(win, pract.RED,(mov.x_lightray2, mov.y_lightray2_075m, mov.width_lightray2, mov.height_lightray2))
        #--Drawing light gate label
        win.blit(mov.lightgates_label, (mov.x_lightgateslabel2, mov.y_lightgateslabel2_075m))

      if mov.chooseheight == -2: #--When the down button is pressed twice
        #--Drawing the new light gates corresponding to a height of 0.5 metres
        lightgate = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate))
        lightgate = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate), 1)
        lightgate2_050m = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate2, mov.y_lightgate2_050m, mov.width_lightgate2, mov.height_lightgate2))
        lightgate2_050m = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate2, mov.y_lightgate2_050m, mov.width_lightgate2, mov.height_lightgate2), 1)
        #--Treating these as rectangles but with very small heights
        lightray = pygame.draw.rect(win, pract.RED,(mov.x_lightray, mov.y_lightray, mov.width_lightray, mov.height_lightray))
        lightray2_050m = pygame.draw.rect(win, pract.RED,(mov.x_lightray2, mov.y_lightray2_050m, mov.width_lightray2, mov.height_lightray2))
        #--Drawing light gate label
        win.blit(mov.lightgates_label, (mov.x_lightgateslabel2, mov.y_lightgateslabel2_050m))

      if mov.chooseheight == -3: #--When the down button is pressed thrice
        #--Drawing the new light gates corresponding to a height of 0.25 metres
        lightgate = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate))
        lightgate = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate), 1)
        lightgate2_025m = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate2, mov.y_lightgate2_025m, mov.width_lightgate2, mov.height_lightgate2))
        lightgate2_025m = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate2, mov.y_lightgate2_025m, mov.width_lightgate2, mov.height_lightgate2), 1)
        #--Treating these as rectangles but with very small heights
        lightray = pygame.draw.rect(win, pract.RED,(mov.x_lightray, mov.y_lightray, mov.width_lightray, mov.height_lightray))
        lightray2_025m = pygame.draw.rect(win, pract.RED,(mov.x_lightray2, mov.y_lightray2_025m, mov.width_lightray2, mov.height_lightray2))
        #--Drawing light gate label
        win.blit(mov.lightgates_label, (mov.x_lightgateslabel2, mov.y_lightgateslabel2_025m))

      #--Drawing electromagnets as rectangles
      electromagnet = pygame.draw.rect(win, pract.GREY,(mov.x_electromagnet, mov.y_electromagnet, mov.width_electromagnet,mov.height_electromagnet))
      electromagnet = pygame.draw.rect(win, pract.BLACK,(mov.x_electromagnet, mov.y_electromagnet, mov.width_electromagnet,mov.height_electromagnet), 1)
      #--Will be drawn as a circle
      ball_bearing = pygame.draw.circle(win, pract.GREY,(mov.x_ball, int(mov.y_ball)), mov.radius_ball)
      ball_bearing = pygame.draw.circle(win, pract.BLACK,(mov.x_ball, int(mov.y_ball)), mov.radius_ball,1)

      #--Drawing to the screen the various labels at the specified x and y coordinates
      win.blit(mov.counterweight_label, (mov.x_counterweightlabel, mov.y_counterweightlabel))
      win.blit(mov.lightgates_label, (mov.x_lightgateslabel, mov.y_lightgateslabel))
      win.blit(mov.magnet_label, (mov.x_magnetlabel, mov.y_magnetlabel))
      win.blit(mov.bearings_label, (mov.x_bearingslabel, mov.y_bearingslabel))
      win.blit(mov.press_spacebar_label, (mov.x_press_spacebarlabel, mov.y_press_spacebarlabel))

      if mov.chooseheight == 0: #--If the light gates are in their original positions
        if mov.y_ball > mov.y_lightray - 1 and mov.y_ball < mov.y_lightray + 1: #--When ball passes the first light gate
          self.lightgate_detect = pygame.time.get_ticks() #--Will record time

        if mov.y_ball > mov.y_lightray2 - 1 and mov.y_ball < mov.y_lightray2 + 1: #--When ball passes the second light gate
          self.lightgate2_detect = pygame.time.get_ticks()
          self.result_time = self.lightgate2_detect - self.lightgate_detect #--Stores the calculated time
          #--Appending time and height to lists
          mov.graphTime.append(((self.result_time)**2)/1000**2)
          mov.graphHeight.append(1)

      if mov.chooseheight == -1: #--If the light gates are separated by a distance of 0.75 metres
        if mov.y_ball > mov.y_lightray - 1 and mov.y_ball < mov.y_lightray + 1: #--When ball passes the first light gate
          self.lightgate_detect = pygame.time.get_ticks() #--Will record time

        if mov.y_ball > mov.y_lightray2_075m - 1 and mov.y_ball < mov.y_lightray2_075m + 1: #--When ball passes the second light gate
          self.lightgate2_detect = pygame.time.get_ticks()
          self.result_time = self.lightgate2_detect - self.lightgate_detect #--Stores the calculated time
          #--Appending time and height to lists
          mov.graphTime.append(((self.result_time)**2)/1000**2)
          mov.graphHeight.append(0.75)

      if mov.chooseheight == -2: #--If the light gates are separated by a distance of 0.5 metres
        if mov.y_ball > mov.y_lightray - 1 and mov.y_ball < mov.y_lightray + 1: #--When ball passes the first light gate
          self.lightgate_detect = pygame.time.get_ticks() #--Will record time

        if mov.y_ball > mov.y_lightray2_050m - 1 and mov.y_ball < mov.y_lightray2_050m + 1: #--When ball passes the second light gate
          self.lightgate2_detect = pygame.time.get_ticks()
          self.result_time = self.lightgate2_detect - self.lightgate_detect #--Stores the calculated time
          #--Appending time and height to lists
          mov.graphTime.append(((self.result_time)**2)/1000**2)
          mov.graphHeight.append(0.5)

      if mov.chooseheight == -3: #--If the light gates are separated by a distance of 0.25 metres
        if mov.y_ball > mov.y_lightray - 1 and mov.y_ball < mov.y_lightray + 1: #--When ball passes the first light gate
          self.lightgate_detect = pygame.time.get_ticks() #--Will record time

        if mov.y_ball > mov.y_lightray2_025m - 1 and mov.y_ball < mov.y_lightray2_025m + 1: #--When ball passes the second light gate
          self.lightgate2_detect = pygame.time.get_ticks()
          self.result_time = self.lightgate2_detect - self.lightgate_detect #--Stores the calculated time
          #--Appending time and height to lists
          mov.graphTime.append(((self.result_time)**2)/1000**2)
          mov.graphHeight.append(0.25)

      if mov.y_ball >= 500: #--If the ball goes past the pad, the while loop will be broken
        #--Displays the time taken for the ball to fall
        self.display_timer = pract.fontObj_mediumsmall.render((f'Time taken to fall between light gates:'), True, pract.BLACK, pract.WHITE)
        self.time_result = pract.fontObj_medium.render(str(f'{self.result_time} milliseconds'), True, pract.BLACK, pract.WHITE)
        win.blit(self.display_timer, (mov.x_displaytimer,mov.y_displaytimer))
        win.blit(self.time_result, (mov.x_timerresult,mov.y_timerresult))
        #--Prompts the user to repeat the practical
        win.blit(mov.repeat_spacebar_label, (mov.x_press_spacebarlabel, mov.y_press_spacebarlabel))
        #--So the user can change the height of the light gates once the ball has fallen
        increaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_upbutton,2)
        decreaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_downbutton,2)
        #--So the user can open a graph of results
        graphButton = pygame.draw.rect(win, pract.WHITE, (mov.x_graphButton, mov.y_graphButton, mov.width_graphButton, mov.height_graphButton))
        graphButton = pygame.draw.rect(win, pract.RED, (mov.x_graphButton, mov.y_graphButton, mov.width_graphButton, mov.height_graphButton),1)
        win.blit(mov.graphButtonText,(mov.x_graphButtonText, mov.y_graphButtonText))

        #--Allows the user to exit the Pygame window
        exitButton = pygame.draw.rect(win, pract.WHITE, (mov.x_exitButton, mov.y_exitButton, mov.width_exitButton, mov.height_exitButton))
        exitButton = pygame.draw.rect(win, pract.RED, (mov.x_exitButton, mov.y_exitButton, mov.width_exitButton, mov.height_exitButton),1)
        win.blit(mov.exitButtonText,(mov.x_exitButtonText, mov.y_exitButtonText))

        #--Makes sure enough data points are plotted on the graph
        mov.countRepeats += 1

        #--Displays the current height between the light gates
        if mov.chooseheight == 0:
          height_1m_text = pract.fontObj_mediumsmall.render(' 1 Metre ', True, pract.BLACK, pract.WHITE)
          win.blit(height_1m_text, (mov.x_height_1m_text, mov.y_height_1m_text))
        elif mov.chooseheight == -1:
          height_075m_text = pract.fontObj_mediumsmall.render(' 0.75 Metres ', True, pract.BLACK, pract.WHITE)
          win.blit(height_075m_text, (mov.x_height_075m_text, mov.y_height_075m_text))
        elif mov.chooseheight == -2:
          height_050m_text = pract.fontObj_mediumsmall.render(' 0.5 Metres ', True, pract.BLACK, pract.WHITE)
          win.blit(height_050m_text, (mov.x_height_050m_text, mov.y_height_050m_text))
        elif mov.chooseheight == -3:
          height_050m_text = pract.fontObj_mediumsmall.render(' 0.25 Metres ', True, pract.BLACK, pract.WHITE)
          win.blit(height_050m_text, (mov.x_height_025m_text, mov.y_height_025m_text))

        #--Sets this condition to true so the while loop can always run
        pract.condition_lightgatebutton2 = True
        #--Detects whether the up or down button has been pressed
        while pract.condition_lightgatebutton2 == True:
          keys = pygame.key.get_pressed() #--Receives any key press
          pos = pygame.mouse.get_pos() #--Receives any mouse press
          #--Boundaries the user's mouse position has to be in for the bottom button to be pressed
```

```
            #--Boundaries the user's mouse position has to be in for the bottom button to be pressed
            if pos[0] > 560 and pos[0]< 620  and pos[1] > 160 and pos[1]< 230:
              event = pygame.event.poll()
              #--The mouse button has to be pressed
              if event.type == pygame.MOUSEBUTTONDOWN:
                mov.chooseheight -=1
                if mov.chooseheight < -3:
                  mov.chooseheight = -3 #--Protects against repeated clicking of the down button
                if mov.chooseheight == -1: #--Displaying the buttons and text for a height of 0.75 metres
                  increaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_upbutton,2)
                  decreaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_downbutton,2)
                  height_075m_text = pract.fontObj_mediumsmall.render(' 0.75 Metres ', True, pract.BLACK, pract.WHITE)
                  win.blit(height_075m_text, (mov.x_height_075m_text, mov.y_height_075m_text))

                if mov.chooseheight == -2: #--Displaying the buttons and text for a height of 0.5 metres
                  increaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_upbutton,2)
                  decreaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_downbutton,2)
                  height_050m_text = pract.fontObj_mediumsmall.render('  0.5 Metres  ', True, pract.BLACK, pract.WHITE)
                  win.blit(height_050m_text, (mov.x_height_050m_text, mov.y_height_050m_text))

                if mov.chooseheight == -3: #--Displaying the buttons and text for a height of 0.25 metres
                  increaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_upbutton,2)
                  decreaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_downbutton,2)
                  height_025m_text = pract.fontObj_mediumsmall.render(' 0.25 Metres  ', True, pract.BLACK, pract.WHITE)
                  win.blit(height_025m_text, (mov.x_height_025m_text, mov.y_height_025m_text))

            if pos[0] > 560 and pos[0]< 620  and pos[1] > 70 and pos[1]< 140: #--Boundaries for the up button
              event = pygame.event.poll()
              #--The mouse button has to be pressed
              if event.type == pygame.MOUSEBUTTONDOWN:
                mov.chooseheight += 1 #--Will increment this attribute by 1
                if mov.chooseheight > 0:
                  mov.chooseheight = 0 #--Protects against repeated clicking of the up button
                else:
                  mov.chooseheight = 0
                  win.fill(pract.ORANGE)
                  stand = pygame.draw.rect(win, pract.GREY,(mov.x_stand, mov.y_stand, mov.width_stand, mov.height_stand))
                  #--An identical rectangle is created for the border
                  stand = pygame.draw.rect(win, pract.BLACK,(mov.x_stand, mov.y_stand, mov.width_stand, mov.height_stand), 1)
                  #--Rectangles for the base of the stand
                  base = pygame.draw.rect(win, pract.GREY,(mov.x_base, mov.y_base, mov.width_base, mov.height_base))
                  base = pygame.draw.rect(win, pract.BLACK,(mov.x_base, mov.y_base, mov.width_base, mov.height_base), 1)
                  #--Rectangles for the top of the screen
                  clamp = pygame.draw.rect(win, pract.GREY,(mov.x_clamp, mov.y_clamp, mov.width_clamp, mov.height_clamp))
                  clamp = pygame.draw.rect(win, pract.BLACK,(mov.x_clamp, mov.y_clamp, mov.width_clamp, mov.height_clamp), 1)
                  #--Rectangles for the pad and counterweight
                  pad = pygame.draw.rect(win, pract.GREY,(mov.x_pad, mov.y_pad, mov.width_pad, mov.height_pad))
                  pad = pygame.draw.rect(win, pract.BLACK,(mov.x_pad, mov.y_pad, mov.width_pad, mov.height_pad), 1)
                  counterweight = pygame.draw.rect(win, pract.GREY,(mov.x_counterweight, mov.y_counterweight, mov.width_counterweight, mov.height_counterweight))
                  counterweight = pygame.draw.rect(win, pract.BLACK,(mov.x_counterweight, mov.y_counterweight, mov.width_counterweight, mov.height_counterweight), 1)
                  #--Drawing light gates as rectangles
                  lightgate = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate))
                  lightgate = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate, mov.y_lightgate, mov.width_lightgate, mov.height_lightgate), 1)
                  lightgate2 = pygame.draw.rect(win, pract.GREY,(mov.x_lightgate2, mov.y_lightgate2, mov.width_lightgate2, mov.height_lightgate2))
                  lightgate2 = pygame.draw.rect(win, pract.BLACK,(mov.x_lightgate2, mov.y_lightgate2, mov.width_lightgate2, mov.height_lightgate2),1)
                  #--Treating these as rectangles but with very small heights
                  lightray = pygame.draw.rect(win, pract.RED,(mov.x_lightray, mov.y_lightray, mov.width_lightray, mov.height_lightray))
                  lightray2 = pygame.draw.rect(win, pract.RED,(mov.x_lightray2, mov.y_lightray2, mov.width_lightray2, mov.height_lightray2))
                  #--Drawing electromagnets as rectangles
                  electromagnet = pygame.draw.rect(win, pract.GREY,(mov.x_electromagnet, mov.y_electromagnet, mov.width_electromagnet,mov.height_electromagnet))
                  electromagnet = pygame.draw.rect(win, pract.BLACK,(mov.x_electromagnet, mov.y_electromagnet, mov.width_electromagnet,mov.height_electromagnet), 1)
                  #--Drawing to the screen the various labels at the specified x and y coordinates
                  win.blit(mov.clampStand_label, (mov.x_clampStandlabel, mov.y_clampStandlabel))
                  win.blit(mov.counterweight_label, (mov.x_counterweightlabel, mov.y_counterweightlabel))
                  win.blit(mov.lightgates_label, (mov.x_lightgateslabel, mov.y_lightgateslabel))
                  win.blit(mov.lightgates_label, (mov.x_lightgateslabel2, mov.y_lightgateslabel2))
                  win.blit(mov.magnet_label, (mov.x_magnetlabel, mov.y_magnetlabel))
                  win.blit(mov.bearings_label, (mov.x_bearingslabel, mov.y_bearingslabel))
                  win.blit(mov.press_spacebar_label, (mov.x_press_spacebarlabel, mov.y_press_spacebarlabel))
                  win.blit(self.display_timer, (mov.x_displaytimer,mov.y_displaytimer))
                  win.blit(self.time_result, (mov.x_timerresult,mov.y_timerresult))
                  #--Prompts the user to repeat the practical
                  win.blit(mov.repeat_spacebar_label, (mov.x_press_spacebarlabel, mov.y_press_spacebarlabel))
                  #--Draws the ball bearing onscreen again
                  ball_bearing = pygame.draw.circle(win, pract.GREY,(mov.x_ball, int(mov.y_ball)), mov.radius_ball)
                  ball_bearing = pygame.draw.circle(win, pract.BLACK,(mov.x_ball, int(mov.y_ball)), mov.radius_ball,1)

                  #--Displaying the buttons and text again
                  increaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_upbutton,2)
                  decreaseheightbutton = pygame.draw.polygon(win, pract.RED, mov.coordinates_downbutton,2)
                  height_1m_text = pract.fontObj_mediumsmall.render(' 1 Metre ', True, pract.BLACK, pract.WHITE)
                  win.blit(height_1m_text, (mov.x_height_1m_text, mov.y_height_1m_text))

            #--Boundaries for the graph button
            if pos[0] > 555 and pos[0]< 665 and pos[1] > 243 and pos[1]< 283:
              event = pygame.event.poll()
              #--The mouse button has to be pressed
              if event.type == pygame.MOUSEBUTTONDOWN:
                #--So enough data points are plotted on the graph
                if mov.countRepeats > 3:
                  #--Plots graph
                  plt.scatter(mov.graphTime,mov.graphHeight)
                  plt.title('Spot the anomalies and work out the gradient!')
                  plt.xlabel('Time Squared (s)^2')
                  plt.ylabel('Height (m)')
                  plt.grid(True)
                  #--Line of best fit
                  plt.plot(np.unique(mov.graphTime), np.polyld(np.polyfit(mov.graphTime, mov.graphHeight, 1))
                                                                          (np.unique(mov.graphTime)))
                  plt.show()
                else:
                  #--Prompts the user to repeat the practical
                  win.blit(mov.repeatText,(mov.x_repeatText,mov.y_repeatText))

            #--Boundaries for the exit button
            if pos[0] > 555 and pos[0]< 665 and pos[1] > 300 and pos[1]< 340:
              event = pygame.event.poll()
              if event.type == pygame.MOUSEBUTTONDOWN:
                #--Exits the window
                command = Quiz()
                pygame.quit()
                sys.exit()

            if keys[pygame.K_SPACE]: #--Checks whether the space bar has been pressed
              pract.condition_lightgatebutton2 = False #--If so, this while loop will be ended
              mov.acceleration = 0 #--Resets acceleration and position of the ball
              mov.y_ball = 96

            for event in pygame.event.get():
              if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            pygame.display.update() #--Updates the display

          if keys[pygame.K_SPACE]: #--Checks whether the space bar has been pressed (only when the up or down button has not been pressed)
            pract.condition_lightgatebutton2 = False #--If so, this while loop will be ended
            mov.acceleration = 0 #--Resets acceleration and position of the ball
            mov.y_ball = 96

        else:
          mov.y_ball += mov.acceleration#--Moves the ball downwards
          mov.acceleration += 0.00161 #--Allows the acceleration of the ball to increase
          #--Displays the ball bearing again but with new coordinates
          ball_bearing = pygame.draw.circle(win, pract.GREY,(mov.x_ball, int(mov.y_ball)), mov.radius_ball)
          ball_bearing = pygame.draw.circle(win, pract.BLACK,(mov.x_ball, int(mov.y_ball)), mov.radius_ball,1)

        for event in pygame.event.get():
          if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
        pygame.display.update() #--Updates the display

      for event in pygame.event.get():
        if event.type == pygame.QUIT:
          pygame.quit()
          sys.exit()
```

```
            sys.exit()
        pygame.display.update() #--Updates the display

class Quiz(): #--New class which will be used for the quiz
    def __init__(self):
        #--Creating a new window which will be identical to the old one
        self.window2 = Tk()
        self.window2.title("Physics Practical Animation and Quiz")#--Name of the window
        self.window2.geometry("1000x700") #--Sets the height and width of the tkinter window
        self.window2.configure(bg="SkyBlue2") #--Sets the background colour of the window
        self.window2.resizable(False, False) #-Tkinter window is not resizable

        #--Initial screen which will allow the user to click a button to begin the practical
        self.quizMaintext = Label(self.window2, text="Quiz: Determination of 'g' by a free-fall method",
                                  fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
        self.startText = Label(self.window2, text="You will now answer 13 multiple choice questions",
                                  fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium))
        self.startButton = Button(self.window2, text="Click to begin!", fg=form.black,bg=form.white,
                                  font=(form.font, form.fontSizeMedium), command = self.LoadQuestions)

        self.quizMaintext.grid(row=0, columnspan=9, pady = 100, padx = 220) #--Main widget in the window
        self.startText.grid(row=1, column=2, columnspan = 5,pady = 80)
        self.startButton.grid(row=2, column=4)

        self.quizArray = [] #--Array of questions and answers that will be used
        self.score = 0 #--Sets score to 0
        self.repeatAnswer = 0

    def LoadQuestions(self):
        with open('Quiz_Questions&Answers.csv', 'r') as Quiz_QA:
            reader = csv.reader(Quiz_QA)
            for row in reader: #--File contents are appended as a 2d array
                self.quizArray.append(row)
        print(self.quizArray)
        self.x = 1
        self.repeatAnswer +=1
        command = self.DisplayQuiz() #--Allows the quiz to run

    def DisplayQuiz(self):
        #--Removes widgets in the window
        self.quizMaintext.grid_forget()
        self.startText.grid_forget()
        self.startButton.grid_forget()
        #--Repaints the background
        self.window2.configure(bg="SkyBlue2")
        #--Questions
        self.QuestionNumber = Label(text=f' Question {self.quizArray[self.x][0]}: ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
        self.Question = Label(text=f' {self.quizArray[self.x][1]} ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLargeQuiz))
        #--Correct answer for the first question
        if self.x == 1:
            self.CorrectAnswer = Button(text=f'  {self.quizArray[self.x][2]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
                                                                                                                command = self.CorrectAnswer)
        #--This solves my problem where the I couldn't click the correct answers when repeating the quiz
        if self.repeatAnswer > 1 and self.x == 1:
            self.CorrectAnswer = Button(text=f'  {self.quizArray[self.x][2]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
                                                                                                                command = self.CorrectAnswer2)
        #--Correct answer for the second question
        if self.x == 2:
            self.CorrectAnswer = Button(text=f'  {self.quizArray[self.x][2]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
                                                                                                                command = self.CorrectAnswer2)
        #--Correct answer for the third question
        if self.x == 3:
            self.CorrectAnswer = Button(text=f'  {self.quizArray[self.x][2]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
                                                                                                                command = self.CorrectAnswer3)
        #--Correct answer for the fourth question
        if self.x == 4:
            self.CorrectAnswer = Button(text=f'  {self.quizArray[self.x][2]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
                                                                                                                command = self.CorrectAnswer4)
        #--Correct answer for the fifth question
        if self.x == 5:
            self.CorrectAnswer = Button(text=f'  {self.quizArray[self.x][2]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
                                                                                                                command = self.CorrectAnswer5)
        #--Correct answer for the sixth question
        if self.x == 6:
            self.CorrectAnswer = Button(text=f'  {self.quizArray[self.x][2]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
                                                                                                                command = self.CorrectAnswer6)
        #--Correct answer for the seventh question
        if self.x == 7:
            self.CorrectAnswer = Button(text=f'  {self.quizArray[self.x][2]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
                                                                                                                command = self.CorrectAnswer7)
        #--Correct answer for the eighth question
        if self.x == 8:
            self.CorrectAnswer = Button(text=f'  {self.quizArray[self.x][2]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
                                                                                                                command = self.CorrectAnswer8)
        #--Correct answer for the ninth question
        if self.x == 9:
            self.CorrectAnswer = Button(text=f'  {self.quizArray[self.x][2]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
                                                                                                                command = self.CorrectAnswer9)
        #--Correct answer for the tenth question
        if self.x == 10:
            self.CorrectAnswer = Button(text=f'  {self.quizArray[self.x][2]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
                                                                                                                command = self.CorrectAnswer10)
        #--Correct answer for the eleventh question
        if self.x == 11:
            self.CorrectAnswer = Button(text=f'  {self.quizArray[self.x][2]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
                                                                                                                command = self.CorrectAnswer11)
        #--Correct answer for the twelfth question
        if self.x == 12:
            self.CorrectAnswer = Button(text=f'  {self.quizArray[self.x][2]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
                                                                                                                command = self.CorrectAnswer12)
        #--Correct answer for the thirteenth question
        if self.x == 13:
            self.CorrectAnswer = Button(text=f'  {self.quizArray[self.x][2]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
                                                                                                                command = self.CorrectAnswer13)
        #--Incorrect Answers
        self.IncorrectAnswer1 = Button(text=f'  {self.quizArray[self.x][3]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
                                                                                                                command = self.IncorrectAnswer)
        self.IncorrectAnswer2 = Button(text=f'  {self.quizArray[self.x][4]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
                                                                                                                command = self.IncorrectAnswer)
        self.IncorrectAnswer3 = Button(text=f'  {self.quizArray[self.x][5]}  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium),
                                                                                                                command = self.IncorrectAnswer)
        #--Randomises the order of the displayed answers
        rowchoice = [3,4,5,6]
        row1 = random.choice(rowchoice)
        #--Removing items from the list so the same coordinates cannot be used twice
        rowchoice.remove(row1)
        row2 = random.choice(rowchoice)
        rowchoice.remove(row2)
        row3 = random.choice(rowchoice)
        rowchoice.remove(row3)
        row4 = random.choice(rowchoice)
        rowchoice.remove(row4)

        #--Determines positioning of widgets
        self.QuestionNumber.grid(row = 1, column = 1, columnspan = 1,padx = 400, pady = 50)
        self.Question.grid(row = 2, column = 1, columnspan = 1,pady = 70)
        self.CorrectAnswer.grid(row = row1, column = 1, padx = 100,pady = 20)
        self.IncorrectAnswer1.grid(row = row2, column = 1, padx = 100,pady = 20)
        self.IncorrectAnswer2.grid(row = row3, column = 1, pady = 20)
        self.IncorrectAnswer3.grid(row = row4, column = 1, pady = 20)

    #--If the first question is answered correctly
    def CorrectAnswer(self):
        #--Removes all widgets in the window
        self.QuestionNumber.grid_forget()
        self.Question.grid_forget()
        self.CorrectAnswer.grid_forget()
        self.IncorrectAnswer1.grid_forget()
        self.IncorrectAnswer2.grid_forget()
        self.IncorrectAnswer3.grid_forget()
        #--Paints the background green
        self.window2.configure(bg=form.green)
        #--Prints text
        self.correctAnswerText = Label(text=f' Correct! You have gained a mark. ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
        self.correctAnswerText.grid(padx = 300, pady = 300)
```

```
    #--Increases the score by 1
    self.score += 1
    #--Allows the next question to be displayed
    self.x +=1
    #--Displays this window for 2 seconds before displaying the next question
    self.window2.after(2000, lambda: self.DisplayQuiz())
    #--Removes this text widget so the next lot of questions and answers can be displayed
    self.window2.after(2000, lambda: self.correctAnswerText.grid_forget())

  #--If the second question is answered correctly
  def CorrectAnswer2(self):
    #--Removes all widgets in the window
    self.QuestionNumber.grid_forget()
    self.Question.grid_forget()
    self.CorrectAnswer.grid_forget()
    self.IncorrectAnswer1.grid_forget()
    self.IncorrectAnswer2.grid_forget()
    self.IncorrectAnswer3.grid_forget()
    #--Paints the background green
    self.window2.configure(bg=form.green)
    #--Prints text
    self.correctAnswerText = Label(text=f' Correct! You have gained a mark. ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
    self.correctAnswerText.grid(padx = 300, pady = 300)
    #--Increases the score by 1
    self.score += 1
    #--Allows the next question to be displayed
    self.x +=1
    #--Displays this window for 2 seconds before displaying the next question
    self.window2.after(2000, lambda: self.DisplayQuiz())
    #--Removes this text widget so the next lot of questions and answers can be displayed
    self.window2.after(2000, lambda: self.correctAnswerText.grid_forget())

  #--If the third question is answered correctly
  def CorrectAnswer3(self):
    #--Removes all widgets in the window
    self.QuestionNumber.grid_forget()
    self.Question.grid_forget()
    self.CorrectAnswer.grid_forget()
    self.IncorrectAnswer1.grid_forget()
    self.IncorrectAnswer2.grid_forget()
    self.IncorrectAnswer3.grid_forget()
    #--Paints the background green
    self.window2.configure(bg=form.green)
    #--Prints text
    self.correctAnswerText = Label(text=f' Correct! You have gained a mark. ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
    self.correctAnswerText.grid(padx = 300, pady = 300)
    #--Increases the score by 1
    self.score += 1
    #--Allows the next question to be displayed
    self.x +=1
    #--Displays this window for 2 seconds before displaying the next question
    self.window2.after(2000, lambda: self.DisplayQuiz())
    #--Removes this text widget so the next lot of questions and answers can be displayed
    self.window2.after(2000, lambda: self.correctAnswerText.grid_forget())

  #--If the fourth question is answered correctly
  def CorrectAnswer4(self):
    #--Removes all widgets in the window
    self.QuestionNumber.grid_forget()
    self.Question.grid_forget()
    self.CorrectAnswer.grid_forget()
    self.IncorrectAnswer1.grid_forget()
    self.IncorrectAnswer2.grid_forget()
    self.IncorrectAnswer3.grid_forget()
    #--Paints the background green
    self.window2.configure(bg=form.green)
    #--Prints text
    self.correctAnswerText = Label(text=f' Correct! You have gained a mark. ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
    self.correctAnswerText.grid(padx = 300, pady = 300)
    #--Increases the score by 1
    self.score += 1
    #--Allows the next question to be displayed
    self.x +=1
    #--Displays this window for 2 seconds before displaying the next question
    self.window2.after(2000, lambda: self.DisplayQuiz())
    #--Removes this text widget so the next lot of questions and answers can be displayed
    self.window2.after(2000, lambda: self.correctAnswerText.grid_forget())

  #--If the fifth question is answered correctly
  def CorrectAnswer5(self):
    #--Removes all widgets in the window
    self.QuestionNumber.grid_forget()
    self.Question.grid_forget()
    self.CorrectAnswer.grid_forget()
    self.IncorrectAnswer1.grid_forget()
    self.IncorrectAnswer2.grid_forget()
    self.IncorrectAnswer3.grid_forget()
    #--Paints the background green
    self.window2.configure(bg=form.green)
    #--Prints text
    self.correctAnswerText = Label(text=f' Correct! You have gained a mark. ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
    self.correctAnswerText.grid(padx = 300, pady = 300)
    #--Increases the score by 1
    self.score += 1
    #--Allows the next question to be displayed
    self.x +=1
    #--Displays this window for 2 seconds before displaying the next question
    self.window2.after(2000, lambda: self.DisplayQuiz())
    #--Removes this text widget so the next lot of questions and answers can be displayed
    self.window2.after(2000, lambda: self.correctAnswerText.grid_forget())

  #--If the sixth question is answered correctly
  def CorrectAnswer6(self):
    #--Removes all widgets in the window
    self.QuestionNumber.grid_forget()
    self.Question.grid_forget()
    self.CorrectAnswer.grid_forget()
    self.IncorrectAnswer1.grid_forget()
    self.IncorrectAnswer2.grid_forget()
    self.IncorrectAnswer3.grid_forget()
    #--Paints the background green
    self.window2.configure(bg=form.green)
    #--Prints text
    self.correctAnswerText = Label(text=f' Correct! You have gained a mark. ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
    self.correctAnswerText.grid(padx = 300, pady = 300)
    #--Increases the score by 1
    self.score += 1
    #--Allows the next question to be displayed
    self.x +=1
    #--Displays this window for 2 seconds before displaying the next question
    self.window2.after(2000, lambda: self.DisplayQuiz())
    #--Removes this text widget so the next lot of questions and answers can be displayed
    self.window2.after(2000, lambda: self.correctAnswerText.grid_forget())

  #--If the seventh question is answered correctly
  def CorrectAnswer7(self):
    #--Removes all widgets in the window
    self.QuestionNumber.grid_forget()
    self.Question.grid_forget()
    self.CorrectAnswer.grid_forget()
    self.IncorrectAnswer1.grid_forget()
    self.IncorrectAnswer2.grid_forget()
    self.IncorrectAnswer3.grid_forget()
    #--Paints the background green
    self.window2.configure(bg=form.green)
    #--Prints text
    self.correctAnswerText = Label(text=f' Correct! You have gained a mark. ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
    self.correctAnswerText.grid(padx = 300, pady = 300)
    #--Increases the score by 1
    self.score += 1
    #--Allows the next question to be displayed
    self.x +=1
    #--Displays this window for 2 seconds before displaying the next question
    self.window2.after(2000, lambda: self.DisplayQuiz())
    #--Removes this text widget so the next lot of questions and answers can be displayed
```

```
    self.window2.after(2000, lambda: self.correctAnswerText.grid_forget())

  #--If the eighth question is answered correctly
  def CorrectAnswer8(self):
    #--Removes all widgets in the window
    self.QuestionNumber.grid_forget()
    self.Question.grid_forget()
    self.CorrectAnswer.grid_forget()
    self.IncorrectAnswer1.grid_forget()
    self.IncorrectAnswer2.grid_forget()
    self.IncorrectAnswer3.grid_forget()
    #--Paints the background green
    self.window2.configure(bg=form.green)
    #--Prints text
    self.correctAnswerText = Label(text=f' Correct! You have gained a mark. ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
    self.correctAnswerText.grid(padx = 300, pady = 300)
    #--Increases the score by 1
    self.score += 1
    #--Allows the next question to be displayed
    self.x +=1
    #--Displays this window for 2 seconds before displaying the next question
    self.window2.after(2000, lambda: self.DisplayQuiz())
    #--Removes this text widget so the next lot of questions and answers can be displayed
    self.window2.after(2000, lambda: self.correctAnswerText.grid_forget())

  #--If the ninth question is answered correctly
  def CorrectAnswer9(self):
    #--Removes all widgets in the window
    self.QuestionNumber.grid_forget()
    self.Question.grid_forget()
    self.CorrectAnswer.grid_forget()
    self.IncorrectAnswer1.grid_forget()
    self.IncorrectAnswer2.grid_forget()
    self.IncorrectAnswer3.grid_forget()
    #--Paints the background green
    self.window2.configure(bg=form.green)
    #--Prints text
    self.correctAnswerText = Label(text=f' Correct! You have gained a mark. ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
    self.correctAnswerText.grid(padx = 300, pady = 300)
    #--Increases the score by 1
    self.score += 1
    #--Allows the next question to be displayed
    self.x +=1
    #--Displays this window for 2 seconds before displaying the next question
    self.window2.after(2000, lambda: self.DisplayQuiz())
    #--Removes this text widget so the next lot of questions and answers can be displayed
    self.window2.after(2000, lambda: self.correctAnswerText.grid_forget())

  #--If the tenth question is answered correctly
  def CorrectAnswer10(self):
    #--Removes all widgets in the window
    self.QuestionNumber.grid_forget()
    self.Question.grid_forget()
    self.CorrectAnswer.grid_forget()
    self.IncorrectAnswer1.grid_forget()
    self.IncorrectAnswer2.grid_forget()
    self.IncorrectAnswer3.grid_forget()
    #--Paints the background green
    self.window2.configure(bg=form.green)
    #--Prints text
    self.correctAnswerText = Label(text=f' Correct! You have gained a mark. ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
    self.correctAnswerText.grid(padx = 300, pady = 300)
    #--Increases the score by 1
    self.score += 1
    #--Allows the next question to be displayed
    self.x +=1
    #--Displays this window for 2 seconds before displaying the next question
    self.window2.after(2000, lambda: self.DisplayQuiz())
    #--Removes this text widget so the next lot of questions and answers can be displayed
    self.window2.after(2000, lambda: self.correctAnswerText.grid_forget())

  #--If the eleventh question is answered correctly
  def CorrectAnswer11(self):
    #--Removes all widgets in the window
    self.QuestionNumber.grid_forget()
    self.Question.grid_forget()
    self.CorrectAnswer.grid_forget()
    self.IncorrectAnswer1.grid_forget()
    self.IncorrectAnswer2.grid_forget()
    self.IncorrectAnswer3.grid_forget()
    #--Paints the background green
    self.window2.configure(bg=form.green)
    #--Prints text
    self.correctAnswerText = Label(text=f' Correct! You have gained a mark. ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
    self.correctAnswerText.grid(padx = 300, pady = 300)
    #--Increases the score by 1
    self.score += 1
    #--Allows the next question to be displayed
    self.x +=1
    #--Displays this window for 2 seconds before displaying the next question
    self.window2.after(2000, lambda: self.DisplayQuiz())
    #--Removes this text widget so the next lot of questions and answers can be displayed
    self.window2.after(2000, lambda: self.correctAnswerText.grid_forget())

  #--If the twelfth question is answered correctly
  def CorrectAnswer12(self):
    #--Removes all widgets in the window
    self.QuestionNumber.grid_forget()
    self.Question.grid_forget()
    self.CorrectAnswer.grid_forget()
    self.IncorrectAnswer1.grid_forget()
    self.IncorrectAnswer2.grid_forget()
    self.IncorrectAnswer3.grid_forget()
    #--Paints the background green
    self.window2.configure(bg=form.green)
    #--Prints text
    self.correctAnswerText = Label(text=f' Correct! You have gained a mark. ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
    self.correctAnswerText.grid(padx = 300, pady = 300)
    #--Increases the score by 1
    self.score += 1
    #--Allows the next question to be displayed
    self.x +=1
    #--Displays this window for 2 seconds before displaying the next question
    self.window2.after(2000, lambda: self.DisplayQuiz())
    #--Removes this text widget so the next lot of questions and answers can be displayed
    self.window2.after(2000, lambda: self.correctAnswerText.grid_forget())

  #--If the thirteenth question is answered correctly
  def CorrectAnswer13(self):
    #--Removes all widgets in the window
    self.QuestionNumber.grid_forget()
    self.Question.grid_forget()
    self.CorrectAnswer.grid_forget()
    self.IncorrectAnswer1.grid_forget()
    self.IncorrectAnswer2.grid_forget()
    self.IncorrectAnswer3.grid_forget()
    #--Paints the background green
    self.window2.configure(bg=form.green)
    #--Prints text
    self.correctAnswerText = Label(text=f' Correct! You have gained a mark. ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
    self.correctAnswerText.grid(padx = 300, pady = 300)
    #--Increases the score by 1
    self.score += 1
    #--Displays this window for 2 seconds before finishing the quiz
    self.window2.after(2000, lambda: self.EndQuiz())
    #--Removes this text widget so the next screen can be displayed
    self.window2.after(2000, lambda: self.correctAnswerText.grid_forget())

  #--If the user answers the question incorrectly
  def IncorrectAnswer(self):
    #--Removes all widgets in the window
    self.QuestionNumber.grid_forget()
    self.Question.grid_forget()
    self.CorrectAnswer.grid_forget()
    self.IncorrectAnswer1.grid_forget()
    self.IncorrectAnswer2.grid_forget()
```

```
    self.IncorrectAnswer2.grid_forget()
    self.IncorrectAnswer3.grid_forget()
    #--Paints the background red
    self.window2.configure(bg=form.quizred)
    #--Prints the correct answer
    self.incorrectAnswerText = Label(text=f' Incorrect. The correct answer was: ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
    self.correctAnswer = Label(text=f' {self.quizArray[self.x][2]}', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
    self.incorrectAnswerText.grid(row = 1, column = 2, padx = 300, pady = 150)
    self.correctAnswer.grid(row = 2, column = 2)
    #--If the user is on the final question
    if self.x == 13:
      #--Displays this window for 3 seconds before finishing the quiz
      self.window2.after(3000, lambda: self.EndQuiz())
    #--Allows the next question to be displayed
    self.x +=1
    #--Displays this window for 3 seconds before moving onto the next question
    self.window2.after(3000, lambda: self.DisplayQuiz())
    #--Removes these text widgets so the next screen can be displayed
    self.window2.after(3000, lambda: self.incorrectAnswerText.grid_forget())
    self.window2.after(3000, lambda: self.correctAnswer.grid_forget())

  #--Will run when the user answers the last question
  def EndQuiz(self):
    #--Repaints the window
    self.window2.configure(bg="SkyBlue2")
    #--Displays the user's score
    self.scoreText = Label(text=f' Your score was: ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
    self.scoreInteger = Label(text=f' {self.score}/13  ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeLarge))
    #--Positions widgets
    self.scoreText.grid(row = 1, column = 2, padx = 400, pady = 70)
    self.scoreInteger.grid(row = 2, column = 2, pady = 60)

    #--Buttons that the user can press
    self.repeatQuiz = Button(text=f' Repeat Quiz ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium), command = self.RepeatQuiz)
    self.exitQuiz = Button(text=f' Exit Quiz ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium), command = self.ExitQuiz)
    self.graphProgress = Button(text=f' View Progress ', fg=form.black,bg=form.white,font=(form.font, form.fontSizeMedium), command = self.GraphProgress)
    #--Positions widgets
    self.repeatQuiz.grid(row = 4, column = 2, pady = 0)
    self.exitQuiz.grid(row = 5, column = 2, pady = 50)
    self.graphProgress.grid(row = 6, column = 2)

  #--Allows the user to repeat the quiz
  def RepeatQuiz(self):
    #--Removes widgets from the display
    self.repeatQuiz.grid_forget()
    self.exitQuiz.grid_forget()
    self.graphProgress.grid_forget()
    self.scoreText.grid_forget()
    self.scoreInteger.grid_forget()
    #--Clears the contents of the array so the quiz can be repeated
    self.quizArray.clear()
    #--Resets the score
    self.score = 0
    #--Repeats the quiz
    command = self.LoadQuestions()

  #--Will exit the program
  def ExitQuiz(self):
    self.window2.destroy()
    print("Program exited")
    #--Deletes this file so each student has their own progress chart
    os.remove('ScoreProgress.csv')

  #--Allows the user to view their progress over time
  def GraphProgress(self):
    #--Allows program to record the current time
    import datetime
    from datetime import date
    #--Records the current time
    t = time.localtime()
    current_time = time.strftime("%H:%M:%S", t)
    #--List of times and scores
    self.progressList = []

    #--Appends the current time and user's score to a CSV file
    rows = [[current_time, self.score]]
    with open('ScoreProgress.csv','a', newline='') as ScoreProgress:
            writer = csv.writer(ScoreProgress)
            writer.writerows(rows) #--Rows written

    #--Writes contents of this file to a list
    with open('ScoreProgress.csv','r', newline='') as ScoreProgress:
            reader = csv.reader(ScoreProgress)
            for row in reader: #--File contents are appended as a 2d array
              self.progressList.append(row)

    del self.progressList[0] #--Deletes first item in list since this will be blank
    self.length = len(self.progressList)
    command = self.DisplayGraph()

  def DisplayGraph(self):
    #--Additional lists that data will be appended to
    Time_Scores = []
    Scores = []
    #--These variables will allow me to iterate through lists
    x = 0
    u = 0
    z = 0

    for z in range(self.length+1):
      #--Appends all the times to a list
      Time_Scores.append(self.progressList[u][0])
      #--Appends all the scores to a list
      Scores.append(self.progressList[u][1])
      #--Iterates through list
      u+=1
      if len(Time_Scores) == len(self.progressList):
        #--Converts the list of scores from strings to integers
        Scores = list(map(int, Scores))
        #--Orders the two lists so they can be plotted
        Time_Scores, Scores = zip(*sorted(zip(Time_Scores, Scores)))

        #--Graph is plotted
        plt.plot(Time_Scores, Scores, color='#444444', marker = '.', linestyle= '-')
        plt.xlabel(f"Time")
        plt.ylabel(f"Score")
        plt.title(f"Your progress:")
        plt.grid(True)
        plt.show()

        print("Graph drawn")
        #--For loop is broken
        break

login = Login() #--Instantiates the main class
window.mainloop()
```

BIBLIOGRAPHY

- Various sources from StackOverflow: https://stackoverflow.com/
- Tech with Tim Pygame tutorial: https://www.youtube.com/watch?v=i6xMBig-pP4&list=PLzMcBGfZo4-lp3jAExUCewBfMx3UZFkh5
- Various sources from w3schools: https://www.w3schools.com/python/default.asp
- Grid geometry manager: https://effbot.org/