Adam Satvilker

Computer science project- 8308

A-Level computer science

25/04/19

Adam Satvilker

Analysis

Problem identification:

Stakeholders: This solution is aimed at students aged 16-18 who are studying A level mathematics, these will be the students in my maths class who at the end will act as beta testers as they are real users in a real environment (classroom). They are all currently studying the course so will be able to give an accurate response.

Defining the problem:
A level mathematics is a very hard and challenging A level and as a result of this, many students struggle in understanding particular ideas and concepts. One in particular is mechanics in maths. To do this I aim to create a solution which is interactive and allows the user to test different outcomes and see the result whilst also providing a visual representation of what is happening to allow for greater understanding. Also I aim to use the data which they have inputted and display key graphs whilst providing explanations of what each one shows and how all the concepts link together.
The main menu will allow for users to enter a name which will be added to the database and will be connected to the other sections of the solution such as the 'Testing', 'graphs' and 'simulator' which will all be different modules.
The testing section will branch to a further set of modules which will allow the user to chose a topic a followed by this enter answers to the given questions. This will be followed by a results page to indicate how well they did.
The graphing module will be a way of users entering data and be able to view specific graphs, again each graph will have a different module that will be branched to once selected and data is inputted by the user.
The simulator page will be separate module that will take input from the user and the a simulation according to that data will be displayed on the screen. The user can do this for many different data entries and see the visual difference each time.

Software and hardware:
My Program will run using the language python 3.4 and a specific library called Matplotlib will be used, this an external library that must be downloaded. The GUI will also be created using python, specifically the library TKinter. The program will be stored as a regular python (.py) file. Also there will be a separate python file to store the database and hold all data. To run this program a desktop or a laptop is required. If a desktop then a mouse, monitor and keyboard are required. The desktop or laptop must have the external library mentioned above installed as well as the latest version of python. A standard computer is required with a minimum of 3GB of RAM and either windows, macOS and linux to run the solution. The solution will not be able to use on any phone or tablet.

Justification of how the problem is solved by computational methods:

Abstraction:
The solution I'm creating involves recreating a real life model of what is actually happening. My program however is aimed at improving and understanding, therefore keeping it as simple as possible is extremely important and so it will not consider air resistance or the mass of the object as well as friction. These are not required to calculate in the specification for learning this therefore it is not required for me to involve as it will over complicate the program and make it harder to understand the concept properly. My solution also won't display all calculations and work out everything, it will simply calculate what the user asks for and display the equation required and the final answer along with appropriate units. The actual working out and the manipulating of equations wont be displayed as this will look overwhelming and appear much complicated when the main aim of the program is to understand what to do and recognise what is actually happening so the equation that is required in that situation to solve is enough whilst the final answer can allow them to check that they have done it right. The student will not see the database of all their data, they will only be able to view the graphs which are drawn from their data. Seeing the raw data will again look confusing, but also will have no relevance until it is put

into a graph. This is because the trends of the data are not apparent and we are only looking to analyse those trends in order to help understand what is happening in that particular scenario.

Thinking ahead:

-Inputs:

My solution will have various different input points. First being their name and the during the test section they must enter integer data types in the answer boxes. They will also have to enter values for which the graph calculator use to create the graph such as time taken and velocity. Using the simulator will require inputs of integer data type again in order to create the simulation.

-Outputs:

The outputs for the test section will be a results page that is displayed after a set of inputs (answers) are entered for the set of questions. It will display whether each question was answered correctly or incorrectly as well as a total score.
The output for the graphs section will simply be the graph that is created.
The simulation side will have an output of a new path of flight being drawn during the simulation. This will be a visual output that the user can see the ball moving on the screen and the path it has taken.

-Pre conditions:

When using the test side to the solution entry boxes cannot be of data types other than integer. If no answer is inputted, the result is automatically marked as incorrect.
For the simulator, the value for the angle of elevation must be between 0 and 180, the value for the velocity must not be negative and an upper limit of 100 to allow for sensible use.
The graphs section cannot take any data type other than integer.

Thinking procedurally:

My solution will be broken down into many sub sections, however it will be first split into 3 main sections:

-Testing:

Within the test section there will be many repeated sequences. For each particular topic there will be a procedure that runs to display the questions and then a procedure to take the inputs and verify them, creating the results for the user. This will be consistent for each topic in the test section.

-Graphing:

The inputting of values and then selecting particular graph will be a procedure and the output of this procedure will be fed as a parameter into one of two other procedure. The other two will be the procedure for creating each type of graph and the relevant calculation. The one which is called depends on which was selected by user. This allows us to break down the problems even further as the user has to make a decision which effectively creates a branch in the solution where a slightly different procedure is needed so having separate procedures is necessary

-Simulating:

The simulator will be of 2 procedures, the first being where the user can input values and then this will bu used as the parameters of the second procedure which will use the relevant calculations and graphics to create the simulation. The two procedures are necessary as one feeds into the other and this is efficient when re entering values

Thinking logically:

-Functions:

The solution will most certainly contain functions. Each procedure will have its own function and particularly whenever the user is required to input or make a decision there will be a branching to a new function. The functions will also feed outputs to other functions as inputs through the use of parameters, for example in the simulator when values are entered

-Iterations:

The solution will also include iterations or repeated sections as the user will constantly be changing values and the calculations must be performed again each time. There also may be iterations in the code when creating the graphs as they will require data to be constantly added until there is no more data being inputted meaning there will be a lot of repetition and loops here. There will be for loops used in checking answers to test questions too.

-Selection/decisions:
There will be selection at all points of user input. When a user answers questions, conditional statements will be used to check of answer is correct against one stored in database. There will also be conditions for checking of a condition has been met in order to stop a loop such as when creating the graphs, when the time inputted has been reached, the calculation loop must stop automatically. Also in the simulator, once the 'ball' reaches the 'floor' the simulation loop is to stop when the condition is met.

Thinking concurrently:
My solution will have multiple functions running concurrently as data will constantly need to be fed to other parts of the solution. For example the simulator will make use of concurrent processing. The function which is displaying the interface and allowing for user input will run whilst the function which is calculating and creating the simulation does. So the user will stilll be able to interact with the interface and input new values whilst the previous simulation is running. In the test section, when the results are being calculated, the solution will be verifying each answer and adding to the score simultaneously. The graph section is similar to the simulateion in that one function will feed into the other so likewise, when one set of input values are being used in the second function to create the graph, you can still enter new values and interact with the interface.

Limitations:
A limitation to my solution is that the visual representation won't be very complex and quite simplistic as I do not have more complicated software that would allow for more complicated simulations to be created. Also to spend time creating this isn't worth it as my solution is to help build a clear understanding. Having a very complicated simulation may confuse the user so it is not worth investing the time into it. The limitations will also be that it can only be used by one user at a time.

Essential features:
Simulation:
The simulations/visual representations which will demonstrate what is happening each time, which is a key part in helping the user to understand the concept through an alternative visual perspective. This will help to make it clear about what is happening and for the user to make sense of it as they can picture it.

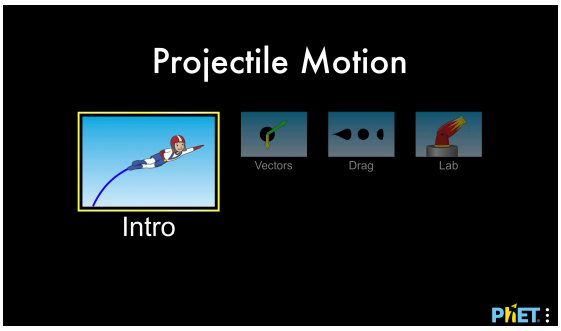
Graphs:
The solution must have graphs which use the data gathered from the user and the calculations performed to create certain graphs. These graphs are very important in the topic and it is important that the user can interpret them properly and gather information/conclusions from them therefore this feature is very important and must be in my solution.

Database:
A database must be part of the solution in order to store all the raw data which is inputted and gathered throughout. Without this the graphs can't be formed as the data wont be saved in an organised way for the graphs to take the information and plot it. Therefore a database is necessary for my solution to work.

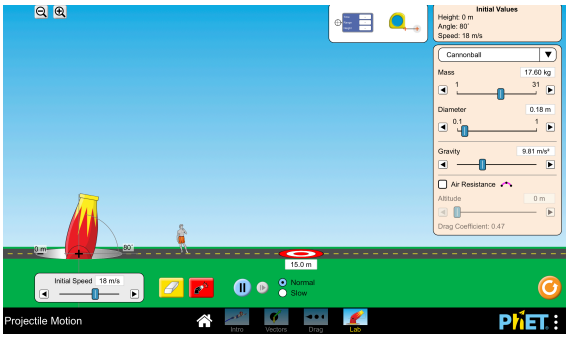
GUI:
There will be a lot of different sections to my solution as there are multiple concepts to cover and various features such as the simulator, the graphs and questions/explanations for the concepts so it must all be played out extremely clearly and be simple enough to use and navigate. A GUI like this is essential as it will add to the simplicity of understanding the concepts as the easier the solution to interact with, the easier it will be to use and therefore learn from.

Research:
A similar solution to the one I intend to make is:
https://phet.colorado.edu/sims/html/projectile-motion/latest/projectile-motion_en.html
This website is a projectile motion simulator which allows you to change values and test what happens in each case, being represented by an animation each time. You can use the simulator to analyse drag and vectors. This is allows you to create situations and see the outcome as it works by using an algorithm that takes data you enter to calculate everything that is missing (what you are looking for)

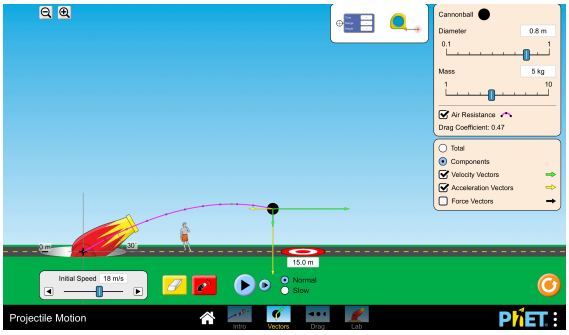
How it works:



The home page gives you your options of using the simulator to analyse either drag or vectors as well as an open lab for using it for either of the two simultaneously and the introduction page. The use of a “lab” is a very good idea as it lets you take what you’ve discovered and freely test anything you want which means it is more beneficial as there is a wider range of tests that can be carried out.



The standard layout for the GUI shows the starting point as a canon and a straight path which the particle will move across when the red fire button is pressed after they use the toggles which are labelled to alter the different values. Each time the toggles are changed the simulator calculates new values and accordingly adjust the simulation of the particle’s movement (for example the length it travels or the height).



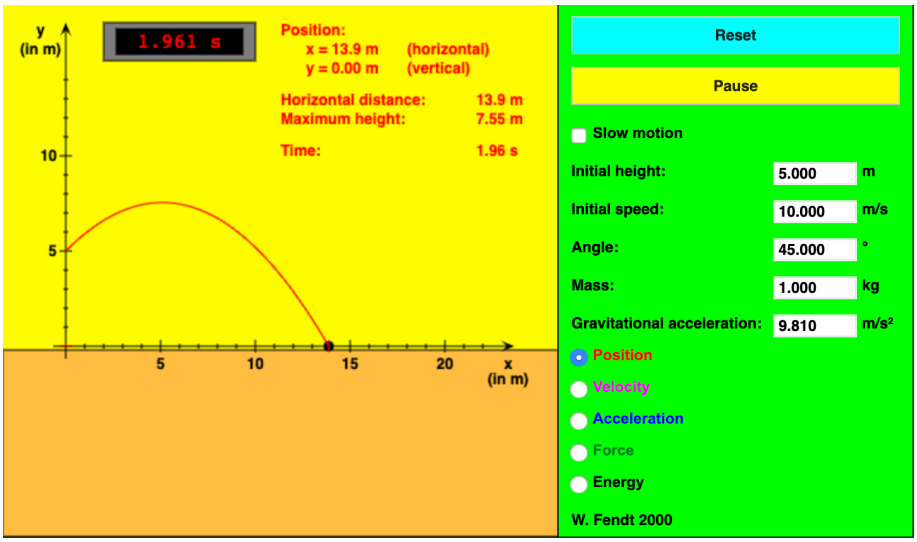
The simulator which is adapted to represent how vectors for acceleration, velocity and force vary during the flight of the particle. The simulator does this by using the length of each arrow to represent magnitude, and the direction of the arrow represents which way it is acting. This is an extremely useful learning tool as these vectors represent what is happening from first principles and therefore is breaking down the concept of what is happening, as much as possible making it very easy to understand as well as the visual representation.

This solution also takes force into consideration, however for my solution I wont include this as it is beyond what the stakeholders require to know and therefore it will add more complexity to the solution which isn’t needed as the aim is to keep it as simple as possible in order to help the user to understand the concept as much and easily as possible.

Solution 2:

https://www.walter-fendt.de/html5/phen/projectile_en.htm

This solution is more simplistic, similar to what I aim to create. Interesting it only consists of one page and it is a HTML webpage that was use to create this



Here you can see the layout is extremely simple and use of different colours make the solution more organised.
The use of a 'slow motion' feature is useful as it allows the user to observe whats happening better and that can help with understanding a feature I could potentially use.

One feature which this has is an option to change the information you view at the top. The use of simple check buttons is an effective way of doing this. My solution could potentially have the option to view different values however there is the limitation of the amount of calculation I will be carrying out. For example I wont be calculating energy or force

Similarly to the first solution there are values such as mass which are being considered that I wont be however this solution is a good example of a simple way of laying it out and having the essential features which are easy to use.

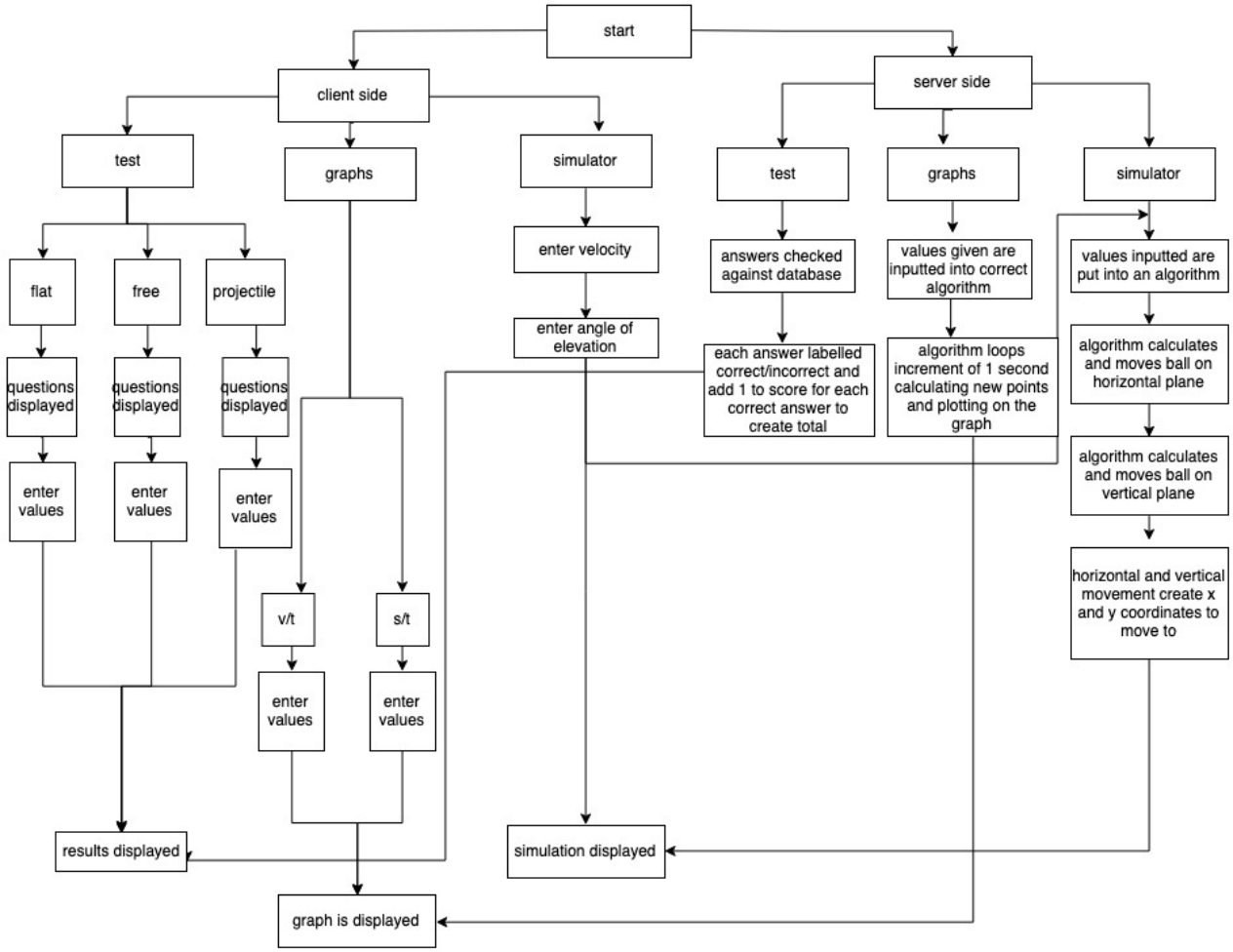
Success criteria:

Criteria	How it will be measured	Justification of criteria
Must be able to use suvat equations to calculate missing values	This criteria is met when the user can input values and the program output back a correct simulation or graph. The program will calculate the time taken and display it in the shell so I can see if this matches and therefore the algorithm works	The main point of the solution is to use the these equations and understand them so it is essential that it can correctly use the equations.
Allowed to change a value at any time in the simulator and receive a new output	If the user enters a value and get an output, to test this the user should be able to change one of the values again and the get a new output	The idea is to add multiple different inputs and get multiple different outcomes which can then be used to plot a graph. So this is very important as a graph cannot be formed correctly or accurately with just one piece of data
User can view all results they have collected in the form of a graph	To test this, when the user has entered data, they can click the option to view all their result and a graph with all the data plots should be displayed	The graphs are an important part of the topic and the user must know about them therefore it is very important that they can view their data in these graphs
A database which can interact with the main algorithm, can be viewed or manipulated by the algorithm	The data for the user which is stored can be viewed and then compared to when the algorithm runs and see what data is outputted from the algorithm and compare if they are the same	If my database cannot interact with my algorithms then the data inputted by the user can no longer be used for further analysis (the graphs and further calculations). This is an important part of the solution and must be here or a lot of my solution will not work as there is no data for the algorithm to manipulate
A simple and easy to use GUI that is user friendly	If the GUI has complimentary colours and buttons which are big and clearly visible to the user, to measure this these questions are given in the form of a questionnaire, the solution must score at least 70% which is rated by the stakeholder	As there is a simulation and graphs involved there are a lot of things beings displayed and this can get very complex meaning it can quickly become hard for the user to you making it hard for them to learn the concepts which is the aim of this program.
user can choose from 3 topics to be tested on.	There will be a menu with the option to select one of the 3.	It is important to ensure that there are 3 options as this means the necessary content has been covered, if not my solution would not be sufficient as it would be leaving gaps.

Criteria	How it will be measured	Justification of criteria
User must be able to view questions and input an answer, the program should be able to correct it/mark it	The user can type and answer to the question and enter the equation they used, if it it right the a tick will appear next to it and if it is wrong a an “X” appears and the correct answer is displayed underneath	The aim is to help the user improve their skills and knowledge in the topic and so it is important that the solution allows the user to check and evaluate their skills at any point in order for them to improve so being able to answer questions and getting feedback is essential.

Design

My solution will aim to break down the problem as follows…



Here you can see the problem decomposition, initially broken down into the ‘server side’ and ‘client side’. The client side is where all the user interaction occurs and the server side is what happens in the background while running which users don’t see. As you can see there are some arrows which cross over and this is to represent where the background calculations or algorithms are used and how they fit into the users experience.

Sub problem	Description
Creating the 3 different planes for user to interact with	The three different scenarios which the user must be familiar and know about, these need to be recreated in the form of a simulation
Calculating values/manipulating equations	All values which need to be calculated using the SUVAT mechanics equations. Also equations must be manipulated/rearranged to find a certain value
Creating graphs	The data and results collected will be used to form specific graphs (distance/time, velocity/time, displacement/time) these are then used to analyse results (e.g.finding gradient and what this means)
Making a simulator	Each plane or scenario needs to perform animations according to input values, will show what happens in each case on a proportional scale so as you increase a certain value the animation will show this change visually.
Creating a database	All values that are calculated for each time the users uses the solution it must be stored in a database where the other parts such s the the graph section can access it and manipulate this data

Functions	Description
Asking user questions	Questions are displayed and then the user can input an answer
Verifying if correct	After entering an answer this function will check to see if its correct, if not will tell the user to try again. Maximum of 5 tries and then answer with working out is displayed
Identifying missing value	This function will identify which values are given and which is missing, selects correct equation to use
Solving equation	Equation is rearranged to find missing value and then calculations are carried out, finding missing value
Creating graph	Each graph will have a basic layout and then when data is available from database this is used to create the graph
Display analysis	An analysis page with explanations of what graphs represent, data such as gradient (calculated from users data) is used to explain concepts.
Main GUI	The main menu that the user will interact with, can select and navigate the software. Will be able to choose which plane they want to focus on and whether they want to go through the tutorial or straight to experimenting with the simulator

Functions	Description
Simulator GUI	This will be where the user interacts with the animation. Can select different options and input values
Animations	When the user inputs a value calculations are carried out and in this function those results and values are translated to movements/animations.

Classes	What it holds	Justification
SUVAT equations	The attributes of this class will be: S-store the value of displacement U-store the value of initial velocity V-store the value of velocity A-store the value of acceleration T-store the value of time Equations- stores all 5 equations Functions included will be identifying missing value and solving equation.	Each calculation carried out will have all of the attributes as they are the different values required for the equation as well as the different equations.
Graphs	All graphs will hold the following attributes: X_title- where the name of the value that is being plotted on the x-axis is stored. Y-title- where the name of the value that is being plotted on the y-axis is stored. Domain- A list which stores all input(x-values) Range- A list which stores all outputs(y-values) They will all have the procedure: display analysis	Each graph will follow the same format and so if all the graphs were a class they can all follow the same sub-procedures which will format the graph and then they can add in specific data. Also they will all follow the same calculations to find the gradient and area.
Main GUI	the main GUI class will hold all functions that allow the user to interact with the solution	Each different page will have a class and the super class will be the basic format/layout of each page and using inheritance the child classes can inherit this and add their own specific attributes.

Data:
The following data will be pre-loaded and will stay the same, just applied where necessary, will be present from the start.

- Suvat equations:

The SUVAT equations are set and must be used as they are so they are just saved from the start. This allows the solution to just check which is appropriate and then use it.

- Value of gravity:

The value of gravity is needed and never changes so this is data which will be manually inserted as it is important and is the same all the time. This keeps the solution more

simple for the user as they don't have to worry about accounting for this in their calculations.

• Questions and answers:
The questions will be set from the start and stored in the database from the start as well as corresponding answers. This is necessary as it would make the solution a lot more complicated if it had to create questions too. The corresponding answer is saved too as it improves efficiency since it doesn't have to go through the calculation process. This keeps the solution simple and quicker

• Analysis information:
The pages of analysis and explanations will also be stored as a text file. This makes sense as there will be a lot of text already complete without being affected by the actual program so it does no need to be apart of the code but a text file which is simply displayed when required, this keeps the code less complicated.

Usability:

The usability aspect of my solution is to create a solution which the user can interact with in the necessary ways and are able to easily use the features of the solution whilst also being as simple, clear and easy to navigate as possible.

The graphical user interface I'm using to do this is TKinter which is a python library and it has many usability features, the main ones I intend to use are:

Feature	Description	Justification
Progress bar	A bar which represents the progress made at a given point of a lengthy process	Will provide user an indication of they're progress, will be useful to include in the tutorial section so they can see how far they are when answering questions. Can estimate the time they need to finish it.
Scale	Can choose a numerical value by directly dragging/sliding the bar across.	When changing values in the simulator this will be much more efficient for the user as they don't have to keep re entering data. Also it takes out the possibility of inputting incorrect datatypes, less for user to do helps keep it more user friendly
Button	Basic user interaction, can display text or image but when clicked by user can perform an action	Will be used constantly through out when entering data or selecting options and navigating the thkinter GUI

Feature	Description	Justification
Label	Displays text or image, user can only view this	Wherever in my interface that requires instructions or information to be displayed this feature is necessary as it doesn't allow user to interact with it just view it which is the only purpose for these particular parts of the solution.
Scroll bar	Allows the user to see all parts of a widget where the contents of that page are significantly more than the rest so doesn't fit size of the widget. Can drag bar horizontally or vertically.	This adds to the ease of use of the solution, will be useful to indicate that there is more information or content to that page and they can scroll down to view it with is very simple and easy to do.
Check button	Has all attributes of a regular button but can also hold a binary value	This will allow me to add binary values so I can use this in my solution when asking the user to select a value from a given group of values and apply this in the simulator or use it in calculating missing values
Combo box	Like an entry box however whilst being able to enter text, there are also options displayed. Can choose from a set of values already provided. Or can still enter a string of text instead.	when answering a question which requires a worded answer to ensure it matches the answer already stored in the database, the options can be displayed below. Giveng the user guidance on how to enter there answer which improves usability and is needed to keep the solution simple and easy to use for the user
Menu bar	A regular menu bar which consists of buttons that carry out different actions. Consistent on all pages of GUI. (e.g. home, back or quit)	This is important as it improves efficiency since it only has to be created once and will be applied to all pages as apposed to repeating the same buttons on each page each time a new one is created

Ease of use:

Borders:
A usability feature is adding borders, this allows you to add an outline or ‘raise’ that particular widget compared to surroundings. This helps improve the layout and keep the layout clear and easy to read

Colours:
I am going to use basic colours to keep it simple and user friendly. Basic colours such as green and red will be used for ‘correct’ and ‘incorrect’ or ‘start’ and ‘stop’.

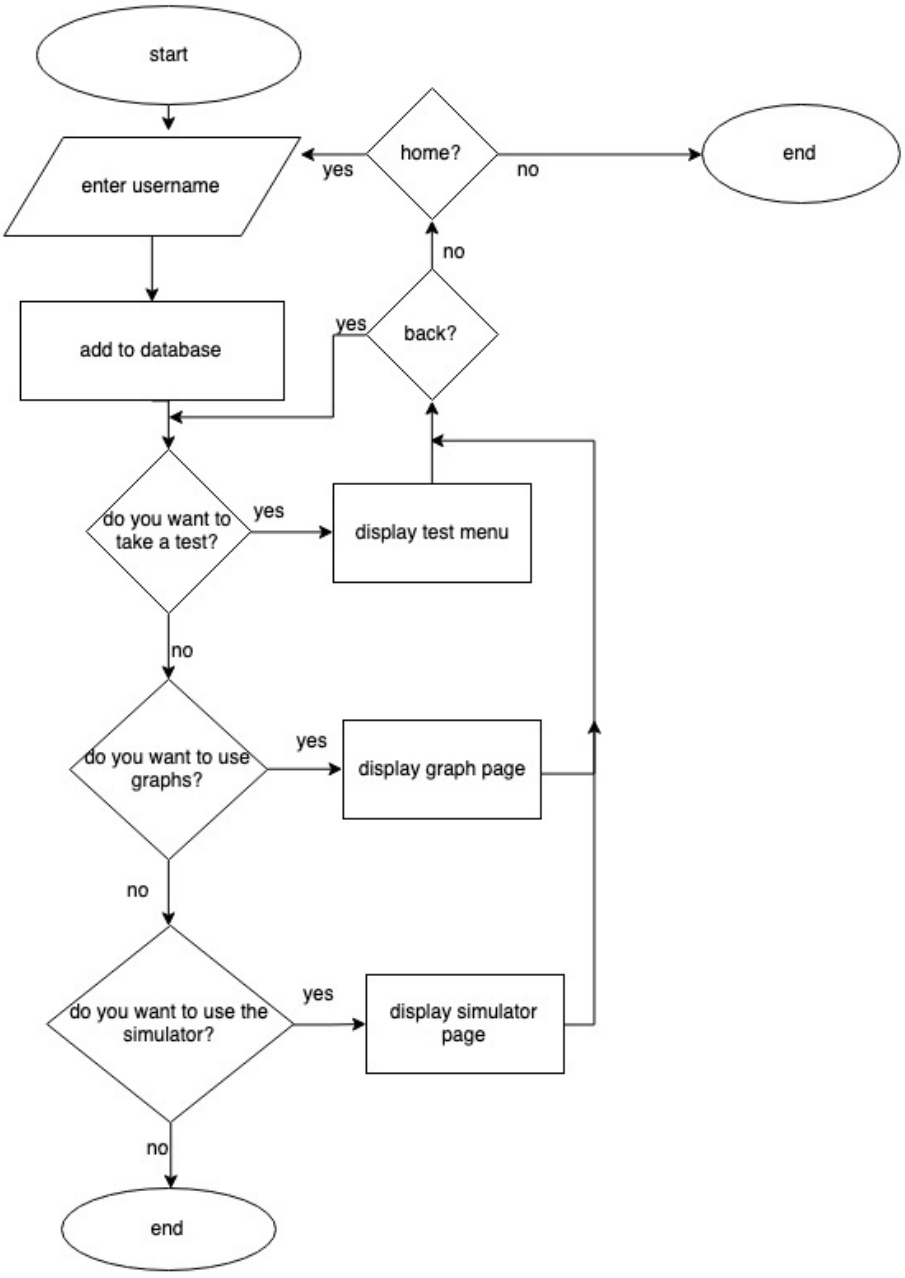
Fonts:
A standard font will be used of a standard to big size. This is to keep the interface as easy to interpret as possible.

Layout:
The overall layout will be simple and tidy as possible(no over congested widgets). Also avoiding having many different ways of interacting with the software all in space as this can be confusing and overwhelming, making the solution hared to use and more complicated.

Flowcharts:

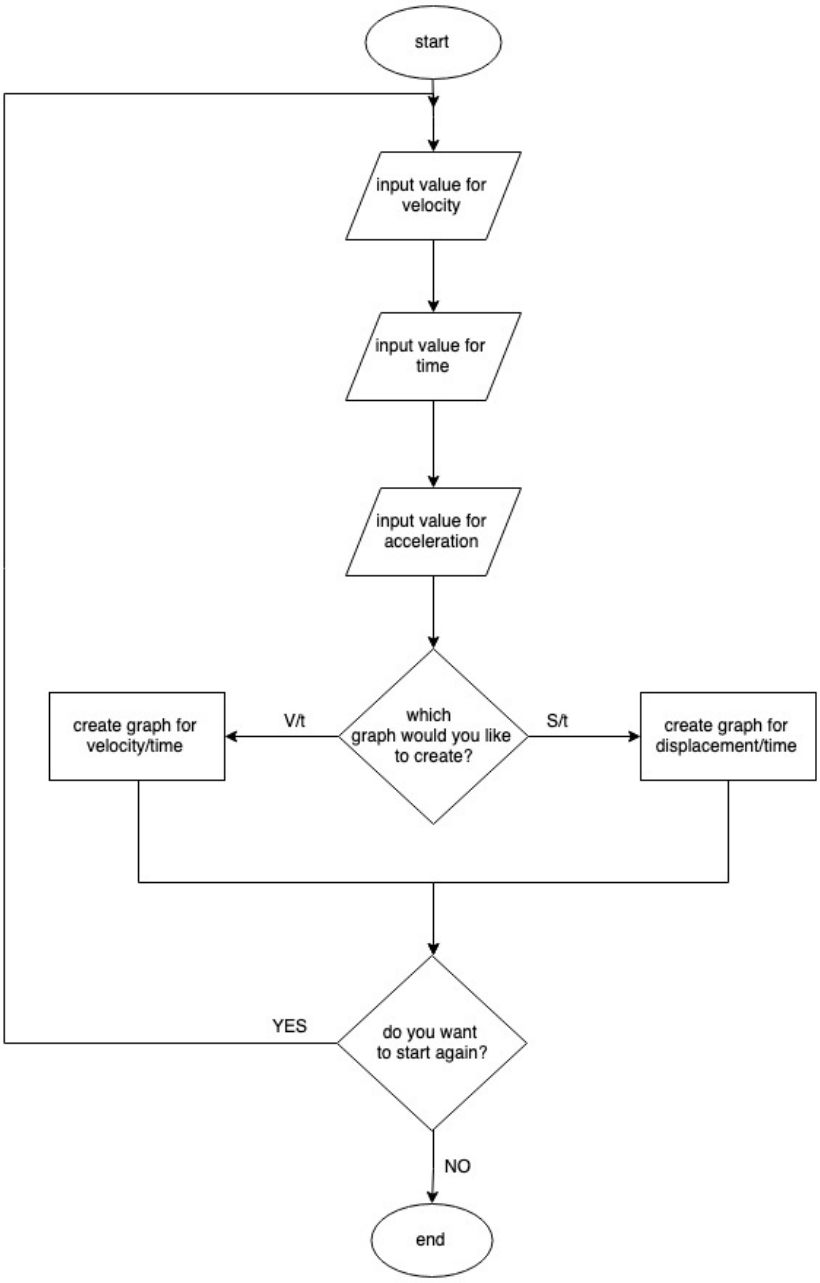
The following flowcharts represent the development and planning of each section of my code before I begin as well examples of algorithms which will be involved in my solution.

MAIN MENU:



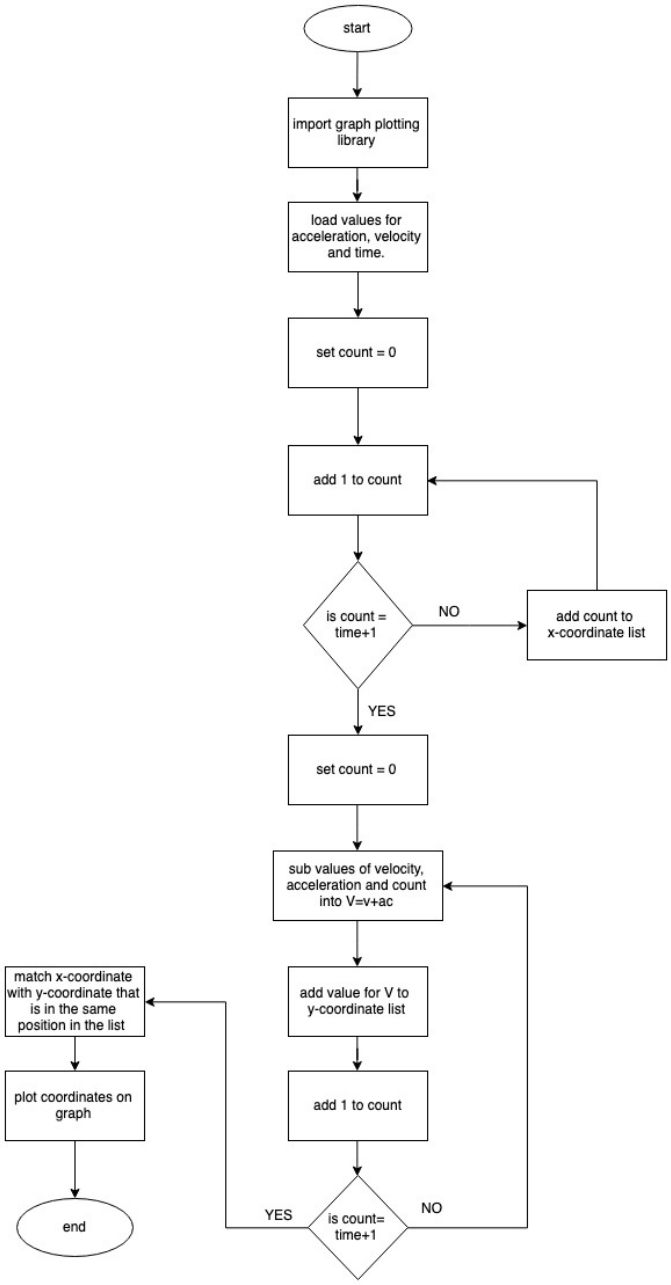
This flowchart is to show the layout of the solution. Without going into detail of how each section works, it is to show how the solution is navigated. As you can see above this is relatively straight forward and simple which is intended to keep the solution simple and easy to use.

GRAPHS:



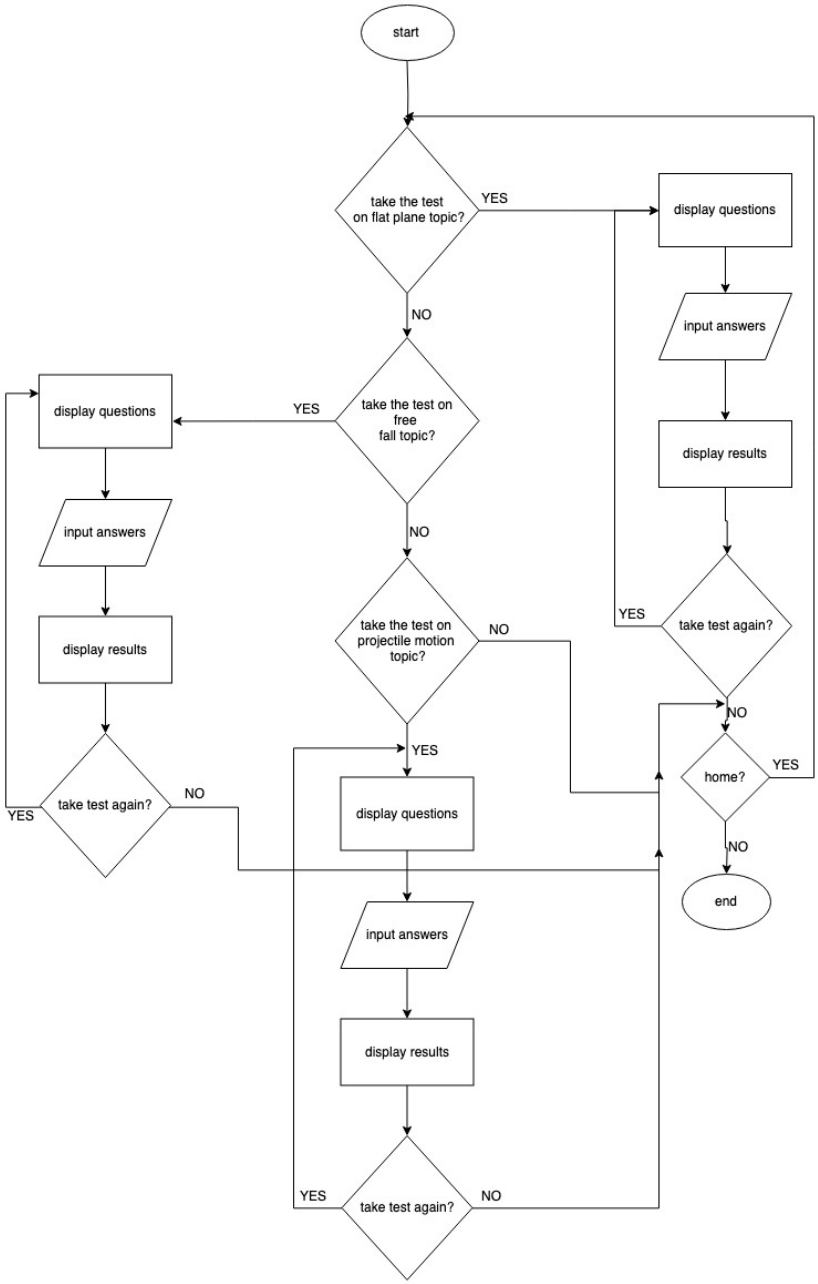
This flowchart focuses on the graphing section of my solution. It is an outline of how the user will go from entering values to being able to view a graph created by the program in accordance to what they have entered. Where it says ‘create graph’ this will be a separate section (flowchart shown next) where the program will carry out the necessary calculations, create the graph and feed it back into this function where it will be displayed to the user

CREATING GRAPH:



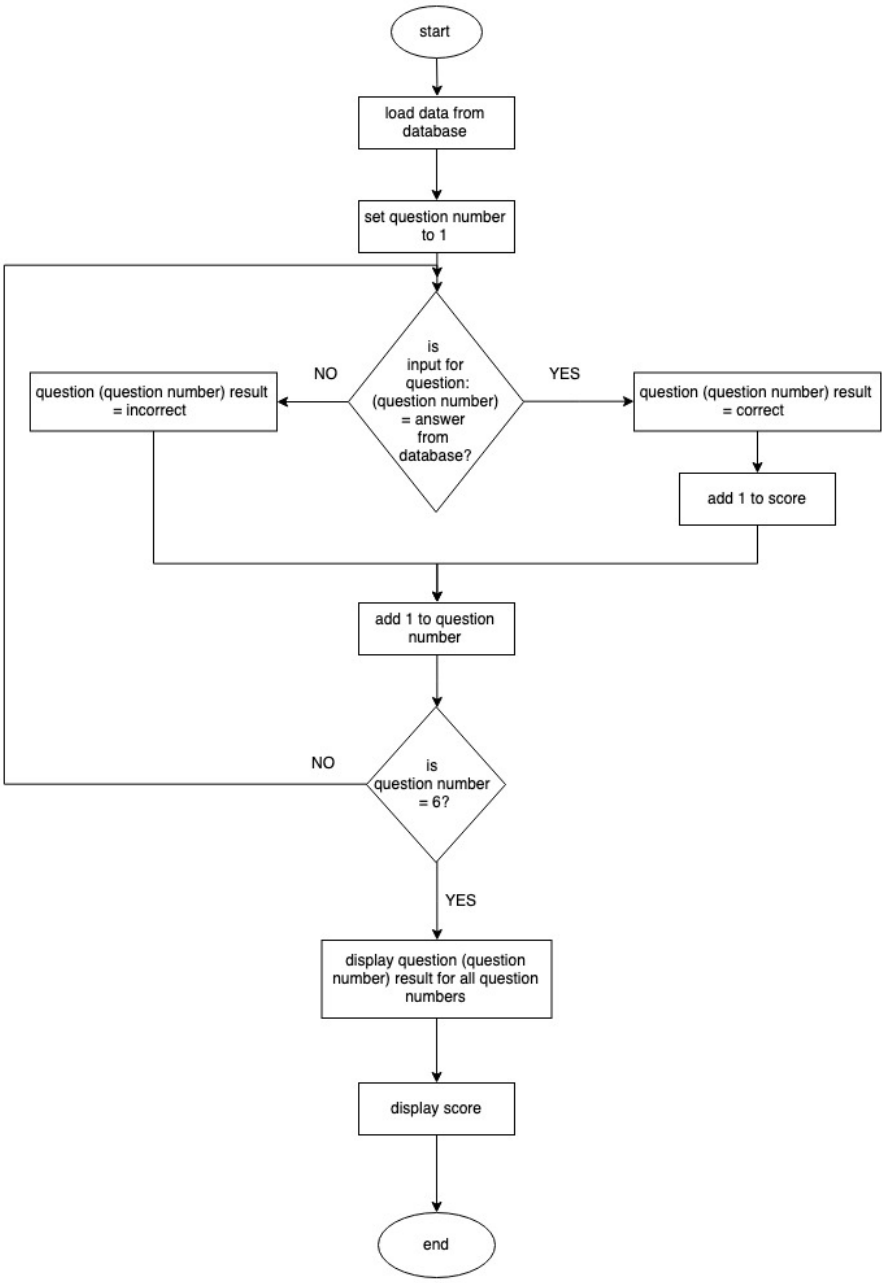
This is the flowchart to show how I will create the graphs once the user input values. As shown above the values inputted from the flowchart previous will be loaded and a graph plotting library will also be imported called matplotlib. The way this flowchart works is it first looks at x-coordinates. It loops with an increment of 1 each time to represent '1second' and this happen until the time taken (entered by user) plus 1 is the same as the count. This plus one is important as it allows the loop to carry on looping for the last value of time as it will still satisfy the loop. Each time the loop is adding this number to a list of x-coordinates. Then the count is reset to 0 to now loop and add the y-coordinates. An equation is used here which calculates the new velocity for each increment of time (1 second), this is done using the count until the time taken plus one is equal to the count. These values are added to a list of y-coordinates. The y-coordinates are matched with the x-coordinates by comparing positions in their lists. These are used to then create the graph.

TESTING SECTION:



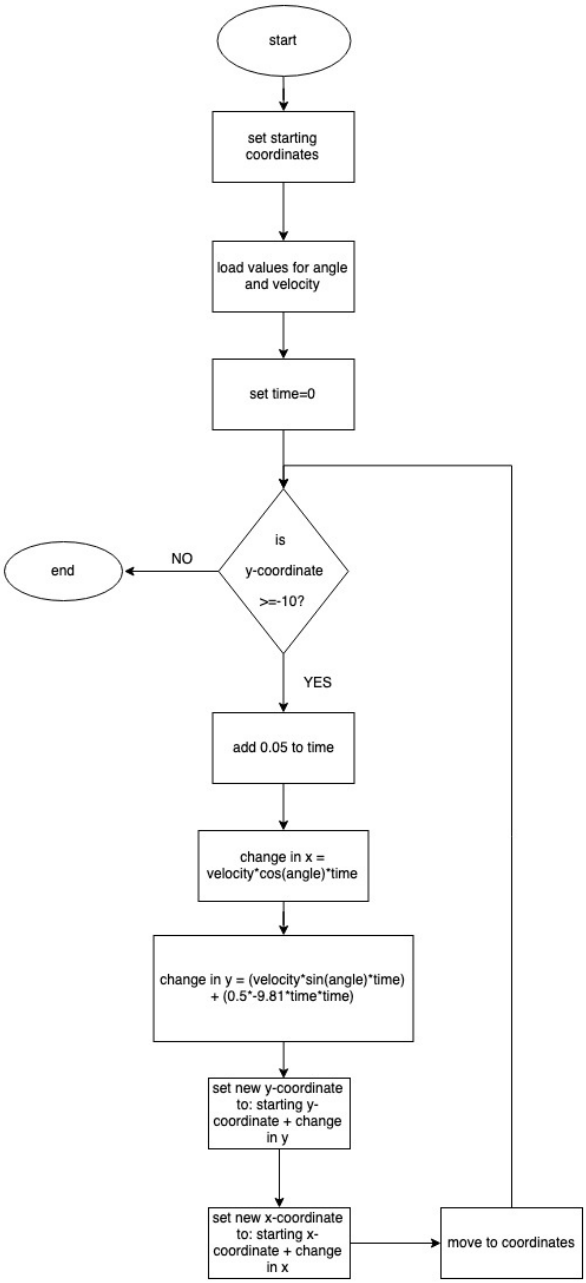
The next section is the test section, this consists of three different sections within itself although each will follow a similar layout just different data. The sections are the three topics which the user can be tested on. The above flowchart shows how each topic when selected requires inputs for the questions which are then used to create results that are displayed to the user after. The creating results part of the solution will be a separate function that is shown below.

CALCULATING RESULTS:



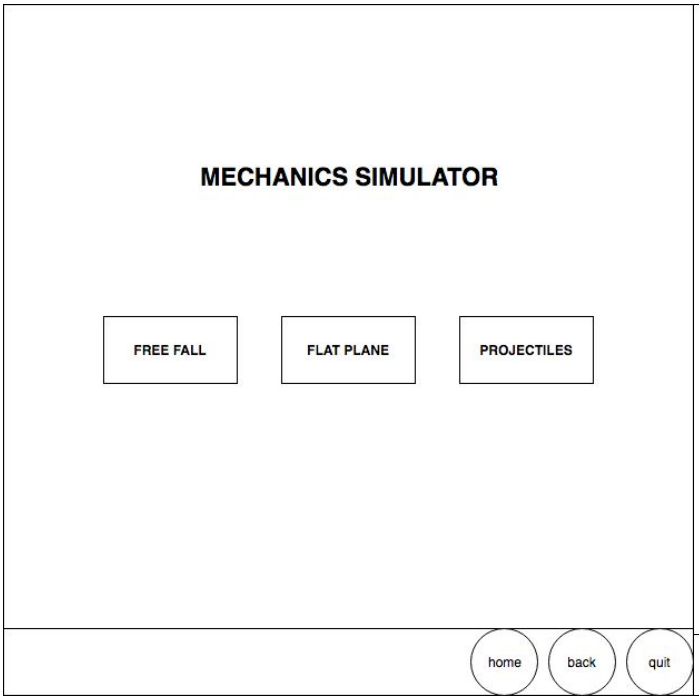
This is the flowchart to represent the way the program will calculate results for the users test. The correct answers are first loaded from the database and then for each question a loop runs to compare the answer inputted by user and the answer stored in the database. When correct the question is marked as correct and the score is increased by 1. If incorrect, it is marked as incorrect and score is not changed. The loop runs until question number = 6 as there are 5 questions and being one more than 5 allows the loop to run for the question number 5 as well then stop after. Once the loop ends, the result (correct/ incorrect) stored for each question and the total score are fed back to the function shown before this one which displays the reulsts back to the user.

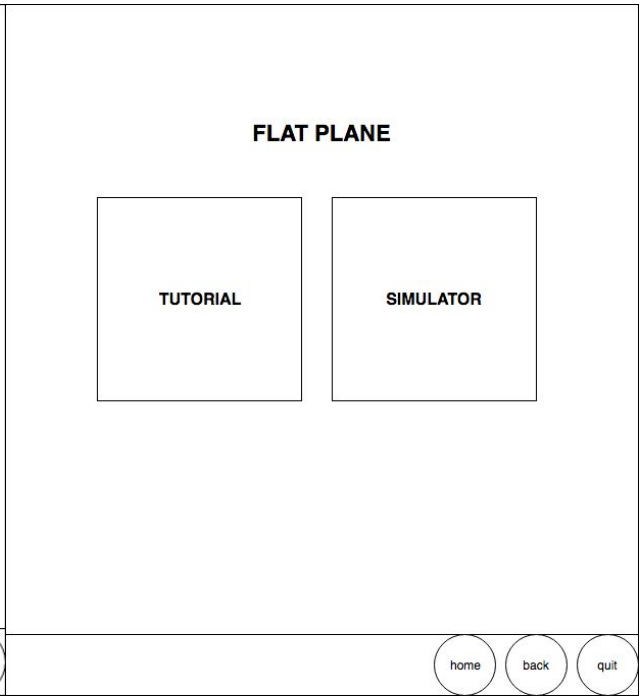
SIMULATOR:

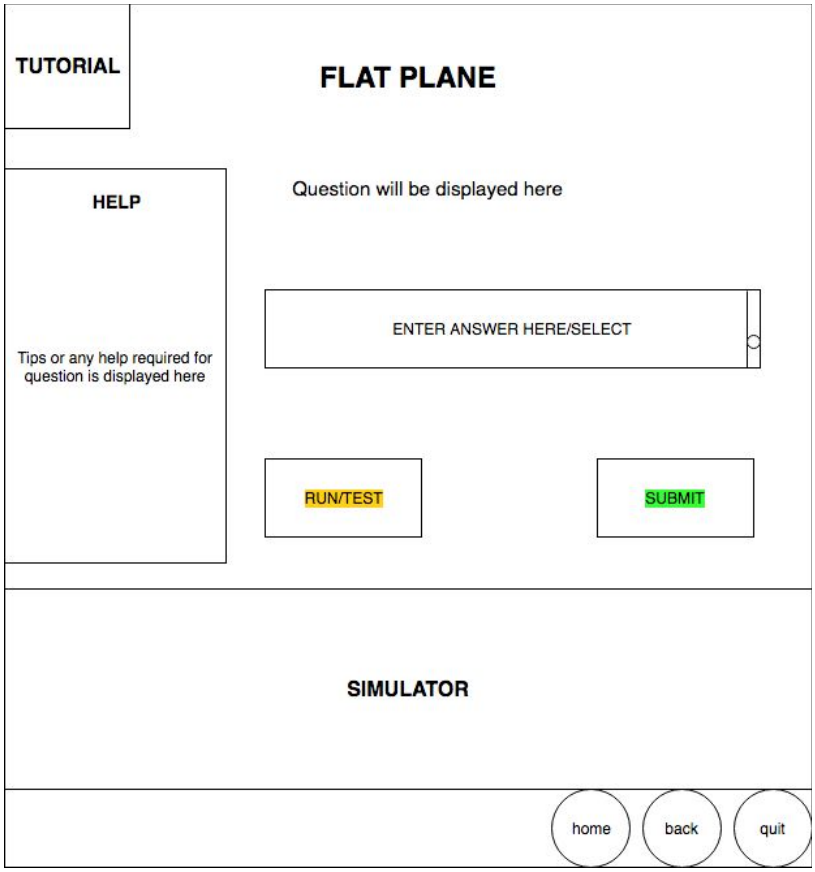


The final section is the simulator which doesn't require much input. This flowchart represents how when the user inputs values for velocity and angle, the simulation is created and displayed. This works by using a loop which runs until the object reaches the same y-coordinate it started with which signifies it has reached the surface again. Each loop increments time by 0.05 as this will increase the amount of loops therefore the simulation will be more accurate. Each time the algorithm will use an equation to calculate the change in displacement in the x direction and then a different equation for the y direction. These values are added to the previous stored x and y coordinates to create the new coordinates which the object is the moved to.

Examples of my GUI designs:







The use of buttons throughout is shown here as well as the use of a menu bar and a combo box. The menu bar is along the bottom and is show to be consistent throughout. The combo box is shown in the third example where it says “enter answer here/select” when entering an answer they can type it but can also click the toggle to view the options of answers.

TEST PLAN:

SECTION 1: home screen:

Test	Data Input	Expected outcome	Outcome as expected?	Points
A name is entered and the enter button is clicked	'John'	The name 'john' is added to the database		
The 'test' button is pressed	n/a	The screen for the test section of the program is displayed and the user can now access these functions		
The 'graphs' button is pressed	n/a	The screen for the graphs section of the program is displayed and the user can now access these functions		
The 'simulator' button is pressed	n/a	The screen for the simulator section of the program is displayed and the user can now access these functions		

SECTION 2: test screen:

Test	Data Input	Expected outcome	Outcome as expected?	Points
The flat plane button is pressed	n/a			
User can input answers under question or can be left blank	Q1) 23.4 Q2) 35 Q3) abc Q4) 4 Q5)	The first question is correct, the second is a correct answer in the database but for a different question so should be incorrect. the second is wrong but also a different data type to test if it will crash. Question 4 is simply wrong and the last question is blank so should be automatically incorrect		

Test	Data Input	Expected outcome	Outcome as expected?	Points
The enter button is pressed	n/a	The results are displayed. The answers inputted are checked against the ones stored in the database for that particular question. Points are awarded where correct.		
the back button is pressed	n/a	Returns user to previous page. The input boxes should be empty and scores reset so user can try again		
The home button is pressed	n/a	the home screen is now displayed		
The free fall button is pressed	n/a			
User can input answers under question or can be left blank	Q1)8 Q2)35 Q3)abc Q4)4 Q5)	The first question is correct, the second is a correct answer in the database but for a different question so should be incorrect. the second is wrong but also a different data type to test if it will crash. Question 4 is simply wrong and the last question is blank so should be automatically incorrect		
The enter button is pressed	n/a			
the back button is pressed	n/a			
The home button is pressed	n/a			
The projectiles button is pressed				
User can input answers under question or can be left blank	Q1) Q2) Q3) Q4) Q5)			

SECTION 3: graphs screen:

Test	Data Input	Expected outcome	Outcome as expected?	Points
Integers are entered for three values and 'velocity/ time' check box is clicked	Velocity:50 Time:30 Acceleration:5	A new screen should pop up with the graph correctly displayed. Each point on graph should be calculated using the values entered. Should be a straight line graph.		
Integers are entered for three values and 'displacement/ time' check box is clicked	Velocity:50 Time:30 Acceleration:5	A new screen should pop up with the graph correctly displayed. Each point on graph should be calculated using the values entered. Should be a straight curved graph.		
A letter is inputted instead of an integer	Velocity:A Time:30 Acceleration:5			
The home button is pressed	n/a			

SECTION 4: simulator screen:

Test	Data Input	Expected outcome	Outcome as expected?	Points
The velocity and angle are set to the lowest possible	Velocity:0 Angle:0	As the algorithm follows the laws of mechanics the ball should remain stationary.		
The velocity and angle are set to the maximum value	Velocity:100 Angle:180	Here the ball should move the exact opposite way to which its originally going however does not come off the floor of the plane so shouldn't move either as it doesn't leave the plane.		

Test	Data Input	Expected outcome	Outcome as expected?	Points
the velocity is set to a random value in between maximum and minimum	Velocity:60 Angle:45	Should expect the simulation to be a curve witch is symmetrical, when ball comes back down and reach the floor, should stop immediately		
Home button is pressed				

Iterative testing:

Test	What happens	Meet requirements?
Clicking Test		
Selecting free fall, flat plane or projectile test		
For each of three test sections have questions displayed		
For each of three test sections have text entry boxes for users to input answer		
Click submit answers		
Be able to view results and total score		
Be able to go back or return home		
Clicking Graphs		
Choose between graphs		
View example of each graph		
Clicking Simulator		
Slider for altering velocity		
Slider for altering angle of elevation		
Click go and simulation runs		
Can run simulation numerous times on user request		
Can return home from simulator page		
Inputting text for username		
Add username to database		

Test	What happens	Meet requirements?
Store data for each user and results		
Is user friendly (/10)		

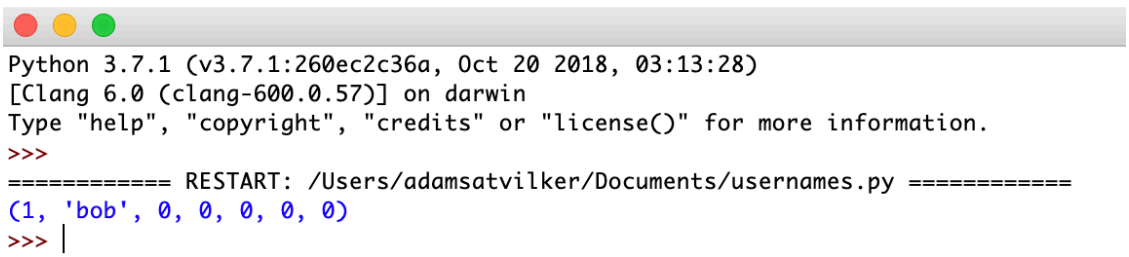
Development

To carry out the development phase of my solution I will be using a combination of the spiral method and extreme developing due to time constraints that will allow for better efficiency. Also the requirements of the solution aren't strict but are also clear, in this case it is not vital to stick to it word for word so if new improvements or changes that could potentially improve the overall solution arise, they will be implemented as they occur. The structure will have fundamental sections that will form the main chunks of the final solution, these will each be broken down into prototypes. Each prototype will be followed by a review and then the end of each section an overall review will take place along with an iterative testing process to ensure this section has successfully met the original requirements.

Section 1:
The first stage of my development was the creation of my database, designed to hold the data used in the 'Test' section of the solution. This means holding all questions to be answered as well as the answers to them. Additionally, a table for results will be used to record the current users results and scores so they can then view there total score and mark for each question straight after.

```
import sqlite3
#library for sql is imported
conn = sqlite3.connect('simulator3.db')
##creating and connecting database
c = conn.cursor()


c.execute('''CREATE TABLE IF NOT EXISTS results
             (resultID int,username text,Q1 int,Q2 int,Q3 int,Q4 int,Q5 int,total int)''')
#creates an empty table with fields above and there data type
results=[(1,'bob',0,0,0,0,0,0)]
c.executemany('INSERT INTO results VALUES (?,?,?,?,?,?,?,?)', results)
#when adding the data the '?,?' represents 8 fields are present


for row in c.execute('SELECT * FROM results'):
                     print(row)
conn.close()
```

To create the database initially I used the library sqlite3 which allowed me to create and query the database. First I began by creating an empty database 'simulator3.db'. Using the command 'CREATE TABLE' to create the table 'results' to store the data for each question and their username. The 'IF NOT EXISTS' ensures no duplicate tables are in my database To then test this was successful I used 'c.executemany' to interact with the database and 'INSERT INTO' to add one record and then queried the database again to display everything in the 'for loop' at the end.



Section 1.1:

The next step was to add the ‘questions’ table which will allow me to store all the questions for the test section of the program. Here they are first organised by the

```
import sqlite3
#library for sql is imported
conn = sqlite3.connect('simulator3.db')
##creating and connecting database
c = conn.cursor()
##creates a cursor for searching database

c.execute('''CREATE TABLE IF NOT EXISTS results
             (resultID int,username text,Q1 int,Q2 int,Q3 int,Q4 int,Q5 int,total int)''')
#creates an empty table with fields above and there data type
results=[(1,'bob',0,0,0,0,0,0)]
c.executemany('INSERT INTO results VALUES (?,?,?,?,?,?,?,?)', results)
#when adding the data the '?,?' represents 8 fields are present


c.execute('''CREATE TABLE IF NOT EXISTS Questions
             (question_type int, question_num int, question text, answer int)''')
Questions=[(1,1,'test','7'),
           (1,2,'n','n'),
           (1,3,'n','n'),
           (1,4,'n','n'),
           (1,5,'n','n'),
           (2,1,'n','n'),
           (2,2,'n','n'),
           (2,3,'n','n'),
           (2,4,'n','n'),
           (2,5,'n','n'),
           (3,1,'n','n'),
           (3,2,'n','n'),
           (3,3,'n','n'),
           (3,4,'n','n'),
           (3,5,'n','n')]
c.executemany('INSERT INTO Questions VALUES (?,?,?,?)',Questions)
for row in c.execute('SELECT * FROM Questions ORDER BY question_num AND question_type ASC'):
    print(row)

##for row in c.execute(''):
##                    print(row)
conn.commit()
##print("username: ")
##for row in c.execute('SELECT * FROM username'):
##                    print(row)
for row in c.execute('SELECT * FROM results'):
                    print(row)
conn.close()
```

question type as there are 3 different types, followed by the question number (1-5). This will later help to request the right set of questions (1-3), followed by the right question(1-5)

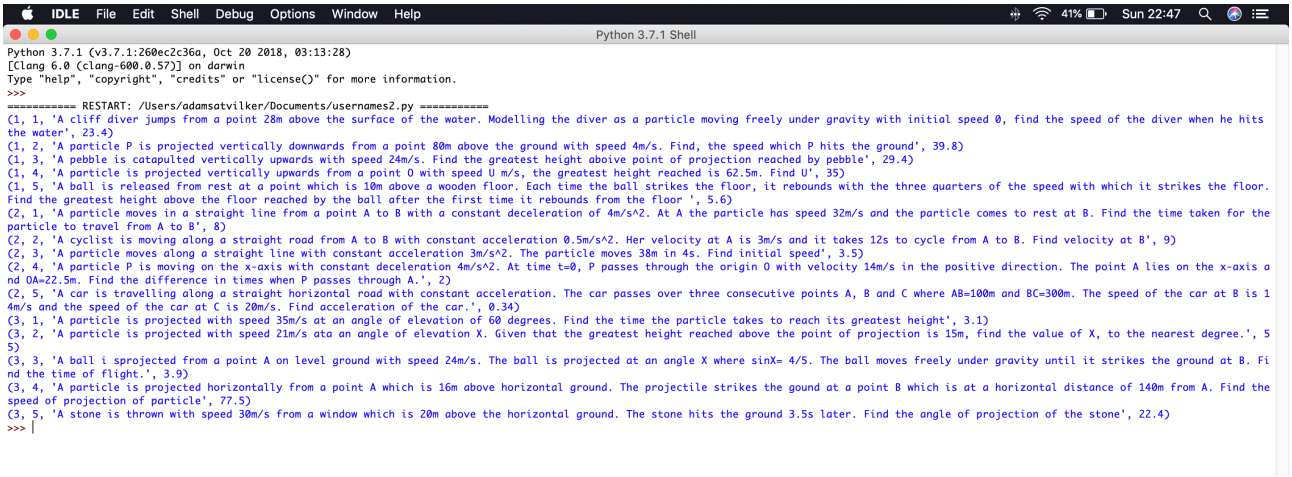
Section 1.2:

Finally to complete the database section, we needed to populate the questions table as

```
import sqlite3
#library for sql is imported
conn = sqlite3.connect('simulator2.db')
##creating and connecting database
c = conn.cursor()
##creates a cursor for searching database

c.execute('''CREATE TABLE IF NOT EXISTS results
             (resultID int,username text,Q1 int,Q2 int,Q3 int,Q4 int,Q5 int,total int)''')
#creates an empty table with fields above and there data type

#when adding the data the '?,?...' represents 8 fields are present


c.execute('''CREATE TABLE IF NOT EXISTS Questions
             (question_type int, question_num int, question text, answer int)''')
Questions=[(1,1,'A cliff diver jumps from a point 28m above the surface of the water. Modelling the diver as a particle moving freely under gravity with initial speed 0, find the speed of the diver when
           (1,2,'A particle P is projected vertically downwards from a point 80m above the ground with speed 4m/s. Find, the speed which P hits the ground','39.8'),
           (1,3,'A pebble is catapulted vertically upwards with speed 24m/s. Find the greatest height aboive point of projection reached by pebble','29.4'),
           (1,4,'A particle is projected vertically upwards from a point O with speed U m/s, the greatest height reached is 62.5m. Find U','35'),
           (1,5,'A ball is released from rest at a point which is 10m above a wooden floor. Each time the ball strikes the floor, it rebounds with the three quarters of the speed with which it strikes t
           (2,1,'A particle moves in a straight line from a point A to B with a constant deceleration of 4m/s^2. At A the particle has speed 32m/s and the particle comes to rest at B. Find the time take
           (2,2,'A cyclist is moving along a straight road from A to B with constant acceleration 0.5m/s^2. Her velocity at A is 3m/s and it takes 12s to cycle from A to B. Find velocity at B','9'),
           (2,3,'A particle moves along a straight line with constant acceleration 3m/s^2. The particle moves 38m in 4s. Find initial speed','3.5'),
           (2,4,'A particle P is moving on the x-axis with constant deceleration 4m/s^2. At time t=0, P passes through the origin O with velocity 14m/s in the positive direction. The point A lies on the
           (2,5,'A car is travelling along a straight horizontal road with constant acceleration. The car passes over three consecutive points A, B and C where AB=100m and BC=300m. The speed of the car
           (3,1,'A particle is projected with speed 35m/s at an angle of elevation of 60 degrees. Find the time the particle takes to reach its greatest height','3.1'),
           (3,2,'A particle is projected with speed 21m/s ata an angle of elevation X. Given that the greatest height reached above the point of projection is 15m, find the value of X, to the nearest de
           (3,3,'A ball i sprojected from a point A on level ground with speed 24m/s. The ball is projected at an angle X where sinX= 4/5. The ball moves freely under gravity until it strikes the ground
           (3,4,'A particle is projected horizontally from a point A which is 16m above horizontal ground. The projectile strikes the gound at a point B which is at a horizontal distance of 140m from A.
           (3,5,'A stone is thrown with speed 30m/s from a window which is 20m above the horizontal ground. The stone hits the ground 3.5s later. Find the angle of projection of the stone','22.4')]
c.executemany('INSERT INTO Questions VALUES (?,?,?,?)',Questions)
for row in c.execute('SELECT * FROM Questions'):
    print(row)

conn.commit()


conn.close()
```

the requirements require us to read the question and match the correct answer which is to be stored from the beginning.

Evidence of this working:



Section 1 review:

Overall this section was very straight forward as it was simple SQL commands being used to create and query our database which for now is also very straight forward as it is a flat file database however as we have adopted an extreme programming method we may revisit this later when we feel we can improve it. There were no specific errors or bugs at this point

Requirement	Was it met?	
Creating a database	YES	The initial database was created using the imported sql library.
Be able to store user scores	YES	A table ‘results’ was created to store user scores.
Be able to store questions with answers	YES	Another separate table was made just for this, organised also by question type. All data now saved in database.

Section 2: Creating the GUI:

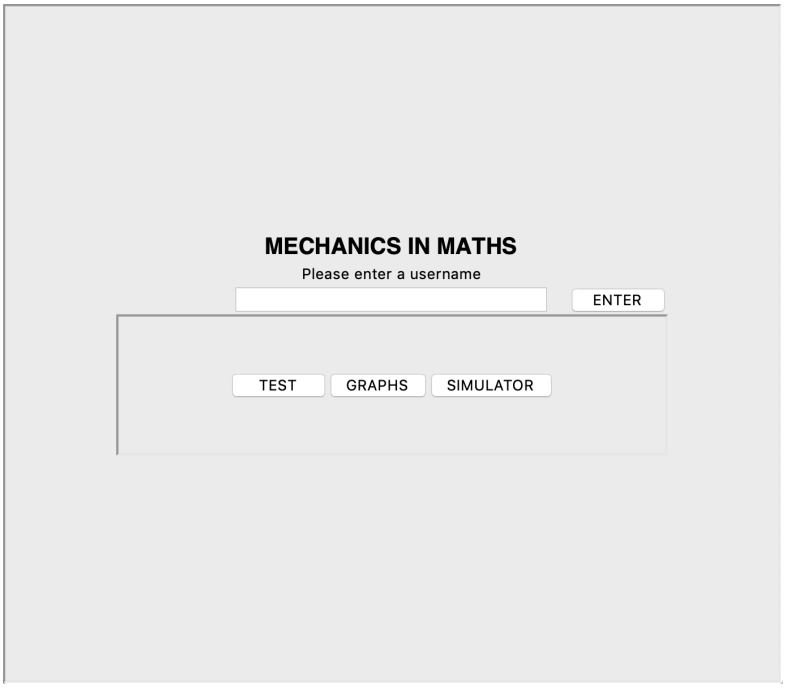
The aim of this section is to build a graphical user interface that allows the user a simple way to interact with each part of my program and to do this we will use the python library TKinter. The interface at this point will only look to be a standard one or a foundation which will then be later developed and revisited in order to integrate the different functions of the solution into one easy to use interface for the final solution.

Section 2.1:

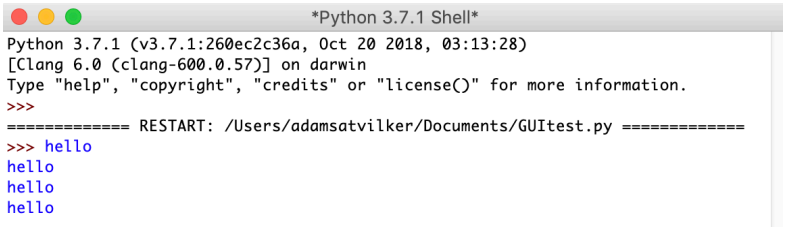
```
from tkinter import *
from tkinter import ttk
import tkinter as tk
import sqlite3
from tkinter import font
#library for sql is imported
def connect():
    print('hello')
def Test_q():
    print('hello')
def graphs():
    print('hello')
def simulator():
    print('hello')


root = tk.Tk()
#creates window
root.title("Mechanics simulator")

root.geometry("880x630")
#set the dimensions of the window

mainframe = ttk.Frame(root,padding= "100 200 100 200",relief='sunken')
subframe = ttk.Frame(mainframe,padding= "100 50 100 50",relief='sunken')
#padding adds extra space around the widget
#create the frame for the window
mainframe.grid (column=0, row=0, sticky=(N, W, E, S))
subframe.grid (column=2, row=15, sticky=(N, W, E, S))
#sticky means how the widget would line up within the grid cell
mainframe.columnconfigure(0, weight=1)
mainframe.rowconfigure(0, weight=1)
subframe.columnconfigure(0, weight=5)
subframe.rowconfigure(0, weight=5)

#column and row configure tells Tk that if the main window is resized, the frame should expand to take up extra space
name=StringVar()
#defines name as a string variable
font_change = font.Font(family='Helvetica', size=20, weight='bold')
#setting for changing font and details of font chnage are stored in this variable
ttk.Label(mainframe, text="MECHANICS IN MATHS",font=font_change).grid (column=2, row=0, sticky=(N))
#the font chnage variable above is then used here to change the font of this lable
ttk.Label(mainframe, text="Please enter a username").grid (column=2, row=5, sticky=(N))
name_entry = ttk.Entry(mainframe, width=30, textvariable=name)
name_entry.grid(column=2, row=10, sticky=(N))
#allows user a space to input text and save it in this variable
ttk.Button(mainframe, text="ENTER", command=connect).grid(column=2, row=10, sticky=E)
ttk.Button(subframe, text="TEST", command=Test_q).grid(column=1, row=10, sticky=S)
ttk.Button(subframe, text="GRAPHS", command=graphs).grid(column=2, row=10, sticky=S)
ttk.Button(subframe, text="SIMULATOR", command=simulator).grid(column=3, row=10, sticky=S)
#buttons are created to navigate to other pages which are defined as functions in the programe
```

To begin with a simple GUI is created which is plain and simple, the template to initially build on. Here we also imported ‘ttk’, ‘tk’ and ‘font’ modules from the tkinter library for

added functionality later. The ttk module allows for overall more features and better looking layout for buttons than tk so having this allows us to improve user usability.
At this point a particular struggle was the geometry and understanding how best to use it and how it works. I set the initial size of the window as well as padding which created almost an invisible border around the edge of the window. I used the .grid function as I learnt this from an online tutorial as the easiest way to keep a consistent layout which all items or objects will use so organising the multiple entities later on in the program will be easier. I included a subframe as well to contain the entities as it helped to keep all the entities organised and be neatly displayed on the window in the centre.



This is the result of running the code above and is the first prototype of the GUI. It contains the required feature of the home page however very dull and plain as yet. The buttons in the subframe will allow the user to navigate to the three sections of the solutions. The user can input text as their username and click enter.



When clicking any of the four buttons, at this point we haven't created windows for each or added any specific functionality however to test the program is working and the functions are being called, a 'hello' statement was added to each function which is shown here to be displaying when each button is pressed confirming it works so far.

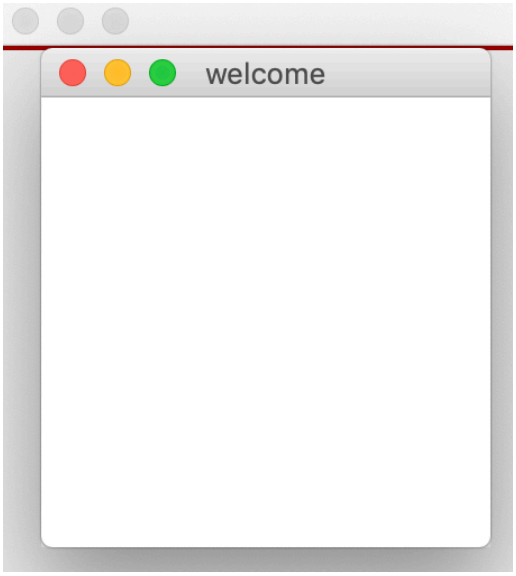
TESTING:

Test	What happens	Meet requirements?
Clicking Test	Nothing happens	No
Selecting free fall, flat plane or projectile test		
For each of three test sections have questions displayed		
For each of three test sections have text entry boxes for users to input answer		
Click submit answers		
Be able to view results and total score		
Be able to go back or return home		
Clicking Graphs	Nothing happens	No
Choose between graphs		
View example of each graph		
Clicking Simulator	Nothing happens	No
Slider for altering velocity		
Slider for altering angle of elevation		
Click go and simulation runs		
Can run simulation numerous times on user request		
Can return home from simulator page		
Inputting text for username	Data is stored in variable however when clicking enter nothing happens.	Yes
Add username to database		
Store data for each user and results		

Section 2.2:

To build on what we have we need to add functionality to the interface to allow the user to interact with it usefully. One issue I had at this point was trying to find a solution to create new windows when pressing a button. Looking at online tutorials there was one option of using the line of code below:

```
additional_window = tk.Toplevel()
additional_window.title("welcome")
```

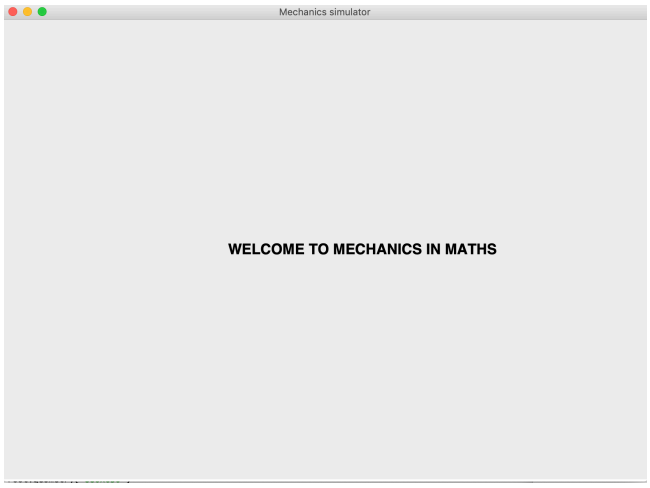


The issue with this was that it was creating a separate ‘pop-up’ window which I didn’t need at this point and wouldn’t be sufficient for this job as it is mainly a small window usually consisting of a simple pop-up message or buttons which can be used to navigate GUI. An example of what happened when I used it is on the left.

The solution I eventually found to this through an online forum was to simply create another GUI within each function that is meant to be a new page. This could simply be copied and pasted as the basic layout was universal and consistent through out the interface and then depending on the needs of that page, it could be further developed uniquely. Here is how I did this:

```
from tkinter import *
from tkinter import ttk
import tkinter as tk
import sqlite3
from tkinter import font
#library for sql is imported
def simulator(*args):
    root.geometry("880x630")
#set the dimensions of the window

    mainframe = ttk.Frame(root,padding= "300 300 300 300")
#padding adds extra space around the widget
#create the frame for the window
    mainframe.grid (column=0, row=0, sticky=(N, W, E, S))
#sticky means how the widget would line up within the grid cell
    mainframe.columnconfigure(0, weight=5)
    mainframe.rowconfigure(0, weight=5)
def graphs(*args):
    root.geometry("880x630")
#set the dimensions of the window

    mainframe = ttk.Frame(root,padding= "300 300 300 300")
#padding adds extra space around the widget
#create the frame for the window
    mainframe.grid (column=0, row=0, sticky=(N, W, E, S))
#sticky means how the widget would line up within the grid cell
    mainframe.columnconfigure(0, weight=5)
    mainframe.rowconfigure(0, weight=5)

def Test_q(*args):
    root.geometry("880x630")
#set the dimensions of the window
    mainframe = ttk.Frame(root,padding= "300 300 300 300")
#padding adds extra space around the widget
#create the frame for the window
    mainframe.grid (column=0, row=0, sticky=(N, W, E, S))
#sticky means how the widget would line up within the grid cell
    mainframe.columnconfigure(0, weight=5)
    mainframe.rowconfigure(0, weight=5)
    font_change = font.Font(family='Helvetica', size=20, weight='bold')
    ttk.Label(mainframe, text=" WELCOME TO MECHANICS IN MATHS ",font=font_change).grid (column=2, row=1, sticky=(N))

def Test_q_flat(*args):
    root.geometry("880x630")
#set the dimensions of the window

    mainframe = ttk.Frame(root,padding= "300 300 300 300")
#padding adds extra space around the widget
#create the frame for the window
    mainframe.grid (column=0, row=0, sticky=(N, W, E, S))
#sticky means how the widget would line up within the grid cell
    mainframe.columnconfigure(0, weight=5)
```

This then resulted in the following being produced if you clicked either of the three options (not including ‘enter’ button).



This has now added functionality as we have successfully found a solution to creating multiple pages. However when clicking ‘enter’ still nothing happens.

Section 2.3:
Now we’ve added actual windows with a consistent foundation GUI for each of the three main sections, we now needed to add functionality to the enter button. The purpose of this button was to take the user input and add this to a database where it is to be stored for later use throughout the program.

```
def connect(*args):
    value= "username6.db"
    conn = sqlite3.connect(value)
    c=conn.cursor()
    user=name.get()
    record = [(1,str(user),0,0,0,0,0)]
    c.executemany('INSERT INTO results VALUES (?,?,?,?,?,?,?)', record)
```

Here the button now takes the name of the database and uses the imported library sqlite3 to connect to the database and then retrieve the user input which we store as a new record in the database.

```
import sqlite3
value= "username6.db"
conn = sqlite3.connect(value)
c=conn.cursor()
for row in c.execute('SELECT * FROM results'):
        print(row)
```

```
>>> (1, 'bob', 0, 0, 0, 0, 0)
(1, 'adam', 0, 0, 0, 0, 0)
```

When clicking enter and querying the database straight away this is produced when the user inputs ‘adam’ and clicks enter. This shows that the button is storing the newly created record in the database which meets the requirement. Note that ‘bob’ is the earlier test record from **section 1** and will be deleted

However if we close the program and query the database again we get:

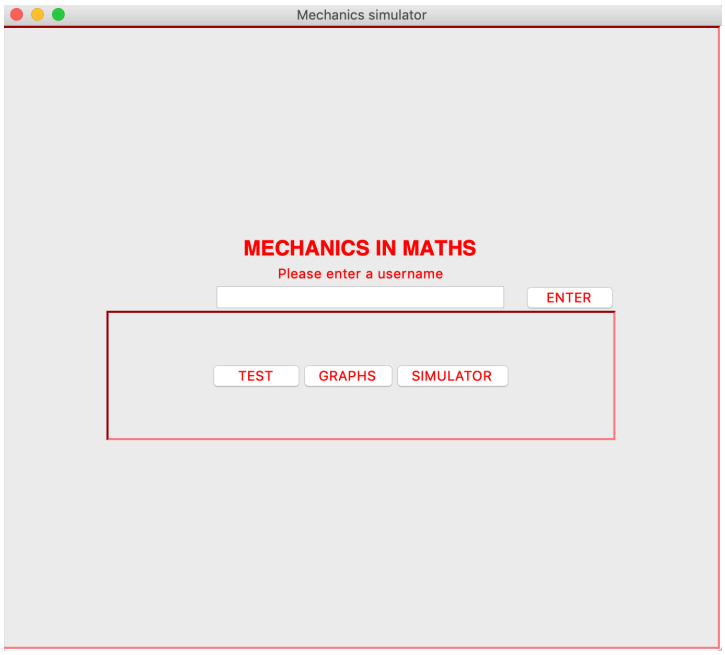
```
(1, 'bob', 0, 0, 0, 0, 0)
>>>
```

This test shows that although in the program it appears to be there when you query it, it is only being stored temporarily. This is an issue as later when using the database, this data wont be there and the user will have no data and the program will be searching for a record which essentially doest exist. The issue I eventually discovered was not a particular problem with the actual database, but that the changes weren't being saved permanently. So once added we could view it as there but these changes weren't saved so afterwards they were lost. The simple solution to this issue was to use: 'conn.commit' which saved the changes permanently. The button now fully meets the requirements.

Section 2.4:
Now having added functionality, another requirement is improving the interface to make it more user friendly. We did this by adding colour and outline as well as bold fonts of a different colour and using spaced out and organised widgets which are large and simple.

```
def home(*args):


    #set the dimensions of the window
    s = ttk.Style()
    s.configure('Mycolour.TFrame',background='red')
    s.configure('Mycolour.TButton',background='red',foreground='red')
    s.configure('Mycolour.TLabel',background='red',foreground='red')

    mainframe = ttk.Frame(root,padding= "100 200 100 200",relief='sunken',style='Mycolour.TFrame')
    subframe = ttk.Frame(mainframe,padding= "100 50 100 50",relief='sunken',style='Mycolour.TFrame')
    ##mainframe.configure(background='red')
    #padding adds extra space around the widget
    #create the frame for the window
    mainframe.grid (column=0, row=0, sticky=(N, W, E, S))
    subframe.grid (column=2, row=15, sticky=(N, W, E, S))
    #sticky means how the widget would line up within the grid cell
    mainframe.columnconfigure(0, weight=1)
    mainframe.rowconfigure(0, weight=1)
    subframe.columnconfigure(0, weight=5)
    subframe.rowconfigure(0, weight=5)

    #column and row configure tells Tk that if the main window is resized, the frame should expand to take up extra space
    global name
    name=StringVar()
    font_change = font.Font(family='Helvetica', size=20, weight='bold')
    ttk.Label(mainframe, text="MECHANICS IN MATHS",font=font_change,foreground='red').grid (column=2, row=0, sticky=(N))
    ttk.Label(mainframe, text="Please enter a username",style='Mycolour.TLabel').grid (column=2, row=5, sticky=(N))
    name_entry = ttk.Entry(mainframe, width=30, textvariable=name)
    name_entry.grid(column=2, row=10, sticky=(N))
    ttk.Button(mainframe, text="ENTER", command=connect, style='Mycolour.TButton').grid(column=2, row=10, sticky=E)
    ttk.Button(subframe, text="TEST", command=Test_q, style='Mycolour.TButton').grid(column=1, row=10, sticky=S)
    ttk.Button(subframe, text="GRAPHS", command=graphs,style='Mycolour.TButton').grid(column=2, row=10, sticky=S)

    ttk.Button(subframe, text="SIMULATOR", command=simulator,style='Mycolour.TButton').grid(column=3, row=10, sticky=S)
```

Above is the code which does this. Earlier I mentioned the use of ttk having added features and whilst this is the case, to begin with I wasn't aware of how to use it so initially I was limited in the design changes I could make. However, I researched it and found I could use 'ttk.Style' which I could then proceed to use to configure the design of frames, buttons and labels.
This added functionality has allowed me to better meet the requirements of a user friendly interface that is easy to use for those at a disadvantage due to simplicity as well as spacing and use of colour to segment sections. Below is what the changes produce:



Final review of section 2:
Overall this section has had a few problematic points in trying to permanently store in the database as well as creating new windows in the most suitable way fit for our purpose however with research we found the correct solutions. As a result at this point we have now started to meet some requirements as shown below. So far we have successfully created a platform to now add to it by adding specific functionality to the three sides of the program. This section has sorted out the basics and will now enable us to move forward and solve the more technical issues which we can then integrate into this GUI and add depth to it.

Test	What happens	Meet requirements?
Clicking Test	A new window is created	Yes
Selecting free fall, flat plane or projectile test		
For each of three test sections have questions displayed		
For each of three test sections have text entry boxes for users to input answer		
Click submit answers		
Be able to view results and total score		
Be able to go back or return home		
Clicking Graphs	A new window is created	Yes

Test	What happens	Meet requirements?
Choose between graphs		
View example of each graph		
Clicking Simulator	A new window is created	Yes
Slider for altering velocity		
Slider for altering angle of elevation		
Click go and simulation runs		
Can run simulation numerous times on user request		
Can return home from simulator page		
Inputting text for username	Data is stored in variable however when clicking enter nothing happens.	Yes
Add username to database	The record is added for the username inputted when enter button clicked	Yes
Store data for each user and results		

Section 3: Simulator manager:
The next section will focus on developing the first of the three main sections, the simulator. This in its nature is a very technical function as it is based on mathematical concepts. This meant that to create an efficient final solution, it needed to be mathematically correct as well as communicated in an efficient way in python (creating a simulation of whats happening). This is one if the most important sections and to begin with I began attempting to code a potential solution to this sub problem separately from my main code:

Section 3.1:
Using a combination of examples on the internet and my own maths knowledge this was my first attempt. I decided to use the python library turtle to do this as it is simple and clear which is what our simulation must be. The use of abstraction here is key as factors such as mass or air resistance are not considered, this is a very simplistic or ideal scenario which however does consider gravity. This is key as the solution must fit the A-level maths specification and they require you to assume these conditions, overall the solution will be more suited to the users need by keeping it simple.
To begin with I had to import the library ‘math’ and the relevant modules.
The first issue I encountered was the angle unit was wrong as I assumed degrees however it was actually in radians, meaning I had to use ‘radians()’ to convert it first.

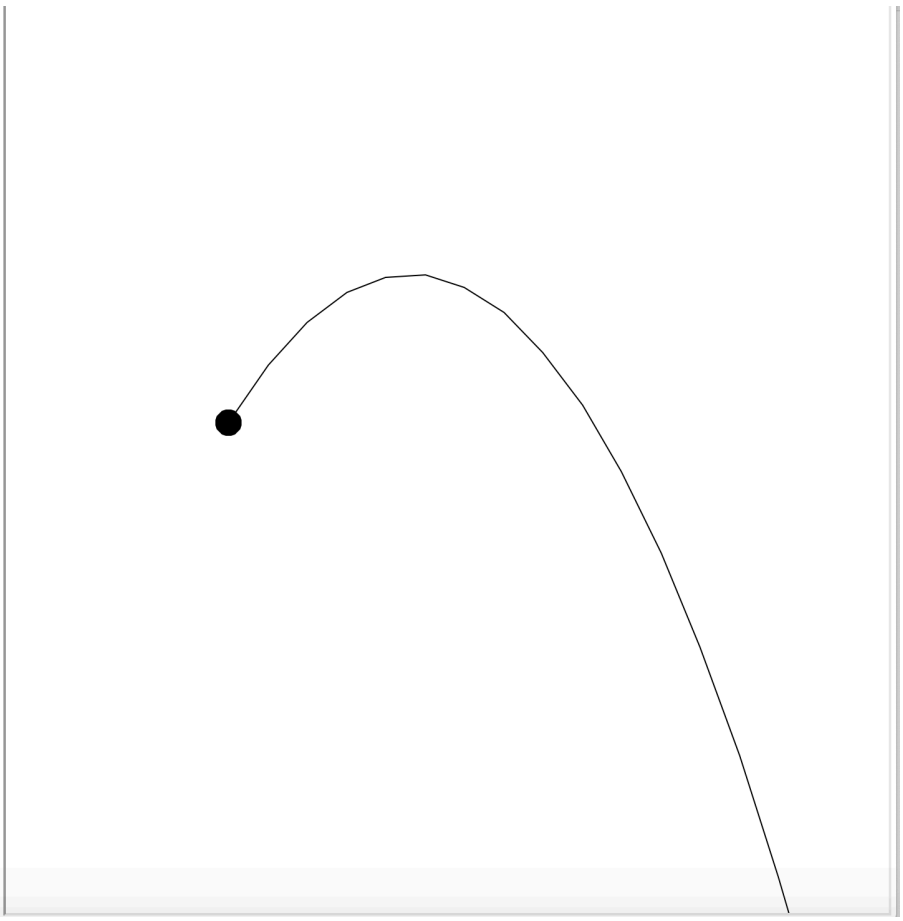
This prototype was the first attempt and the aim of the algorithm is to take an input of velocity and angle of elevation. Then, these values are substituted into appropriate equations to work out the displacement of both the horizontal and vertical components. In terms of the simulation, these values are the vectors used to move the object on the x-axis and y-axis (horizontal and vertical). This is done on a loop between the first second and the second after the time of flight.

*turtle projectile.py - /Use

```
import turtle
from math import pi , atan, sin, cos, radians
##libraries imported
x0, y0 = -180, -100  # initial location

initial_v = int(input("enter initial velocity"))  # initial velocity in units per second
angle=int(input("please enter angle of elevation"))
##asks user to input data
vx=initial_v*(cos(angle))
vy=initial_v*(sin(angle))
##works out velocity on both planes
travel_time = int(input("enter time taken"))  # seconds
##asks user for time
g = 11  # acceleration due to gravity in units per second squared
ball = turtle.Turtle(shape='circle', visible=False)
##creating turtle object
ball.penup()
ball.radians()  # to make turtle compatible with math.atan()
ball.setheading(radians(angle))  # straight up
ball.goto(x0, y0)
ball.pendown()
##will start drawing now when moved
ball.showturtle()
ball.stamp()

for t in range(1, travel_time + 1):

    x = x0 + (vx * t)
    y = y0 + (vy * t) - ((g / 2 )*(t**2))
##for every increment of time, distance moved is added and coordinates have new values
    ball.goto(x, y)
## the ball goes to the coordinates
    print(x, y)
    angle = atan((vy * t - g * t**2) / (vx * t))
    ball.setheading(angle)
##works out new angle at this value for t and adjusts the ball
```

The algorithm successfully manipulates the equations and produces the correct answers and also successfully translates this to vectors for the simulation. However, the issue with it is that the algorithm is only partially accurate as once the path of projection reaches the ‘floor’ or 0 displacement, it carries on going below this which isn’t accurate. The issue originates from the for loop. Initially, the loop was set to end exactly when the time of flight had been reached however with a for loop this meant that it would just carry on iterating every second without taking into consideration that it had returned to original displacement of 0.
An example of this error is below showing the coordinates at each stage of the iteration in the loop. You can see it does to begin to resemble projectile motion with negligible air resistance but doesn’t stop when it should suggesting the algorithm isn’t completely accurate.

```
enter initial velocity60
please enter angle of elevation45
enter time taken30
-148.4806806709362 45.55421147204711
-116.96136134187243 80.10842294409422
-85.44204201280866 103.66263441614132
-53.922722683744865 116.21684588818843
-22.403403354681075 117.77105736023555
9.115915974382688 108.32526883228263
40.63523530344648 87.87948030432977
72.15455463251027 56.43369177637686
103.67387396157403 13.987903248423947
135.19319329063785 -39.45788527952891
166.7125126197016 -103.90367380748182
198.23183194876538 -179.34946233543474
229.7511512778292 -265.79525086338765
261.27047060689296 -363.24103939134045
292.7897899359568 -471.68682791929336
324.30910926502054 -591.1326164472463
355.82842859408436 -721.5784049751992
387.34774792314806 -863.0241935031521
418.8670672522119 -1015.4699820311049
450.3863865812757 -1178.9157705590578
481.9057059103394 -1353.3615590870108
513.4250252394032 -1538.8073476149636
544.944344568467 -1735.2531361429164
576.4636638975308 -1942.6989246708695
607.9829832265946 -2161.1447131988225
639.5023025556584 -2390.590501726775
671.0216218847222 -2631.036290254728
702.5409412137859 -2882.482078782681
734.0602605428497 -3144.9278673106337
>>>
```



Section 3.1 (Attempt 2):

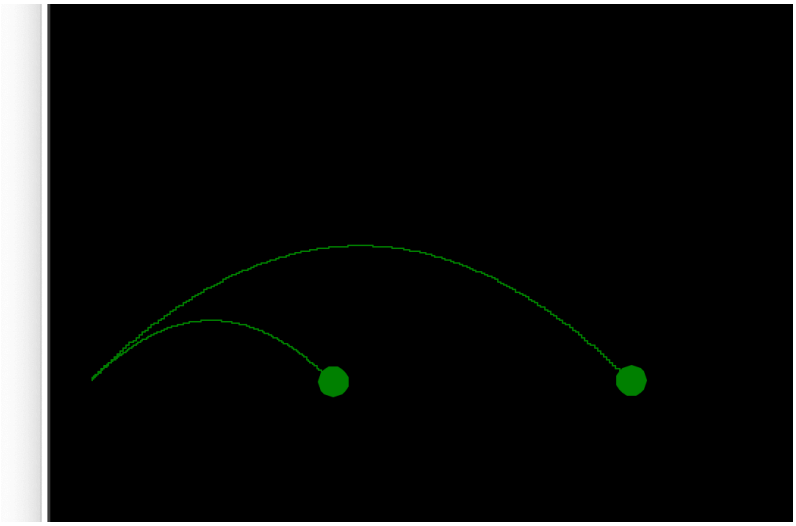
Here was the second attempt to solving this sub problem:

```
import turtle
##import turtle graphics library
from math import sin, cos, radians
##import relevant modules from math library
def run(velocity,angle):
##create function to run simulation with parameters of velocity and angle
    wn=turtle.Screen()
    wn.bgcolor("black")
    wn.title("bouncing ball")
##creates main window
    ball= turtle.Turtle()
    ball.shape('circle')
    ball.color('green')
##creates ball
    ball.penup()
    ball.speed(100)
    ball.goto(-330,-10)
    ball.pendown()
##ball is moved  to starting position without drawing on window
    ball.dy=0
    g=0.1
## gravity is set to 0.1
    time = 0


    while ball.ycor() >= -10:
##while loop to run until coordinate is back to original value of -10


        time = time + 0.05

## for each iteration, time is indented by 0.05


        changeInX = (velocity*cos(angle)) * time
        changeInY = ((velocity*sin(angle)) * time) + (0.5*-9.81*time*time)
##formulas for change in displacement from beggining are calculated on both the horizontal and vertical, for each iteration of time
        ball.sety(-10+changeInY)
        ball.setx(-330+changeInX)
##the displacement worked out above for each component is then added to the original coordinates and the ball is moved to new position


while True:
    velocity=int(input('enter initial velocity'))
    angle=int(input("enter angle of elevation"))
    angle=radians(angle)
    run(velocity,angle)
##until user ends programe, will continue to ask for values and these are used to be fed back as parameters of run function to run simulation again.
```

The changes I made from the first attempt was now taking into consideration the point of 0 displacement (the ground). I did this by instead using a while loop as this would be a condition of making sure it only runs until it reaches the 'floor'. Also the use of a while loop here allowed me to indent time by an extremely small amount and therefore have a more accurate model that resembles the real motion even more, since more values are used within the same time period. Increasing the frequency of data makes the model more accurate. However, with a for loop this is not possible as they only take integers not float, limiting the accuracy to every second.
The new version then calculates the change in displacement for both the horizontal and vertical components from the start (time=0) to that iteration of time in the loop. This is then added to the original starting points of each component and the ball is moved to this new position. This should be already much more accurate than the last due to the smaller time increments as well as now focusing the loop on time until it reaches the floor as opposed to going continuously until a certain time limit is reached. This is how it works in real life as time of projection will continue until it is stopped by the ground.
The results of running this are shown below:

Test 1: Constant angle, varied velocity:

```
enter initial velocity60
enter angle of elevation45
time taken 8.649999999999988
enter initial velocity40
enter angle of elevation45
time taken 5.799999999999987
enter initial velocity
```

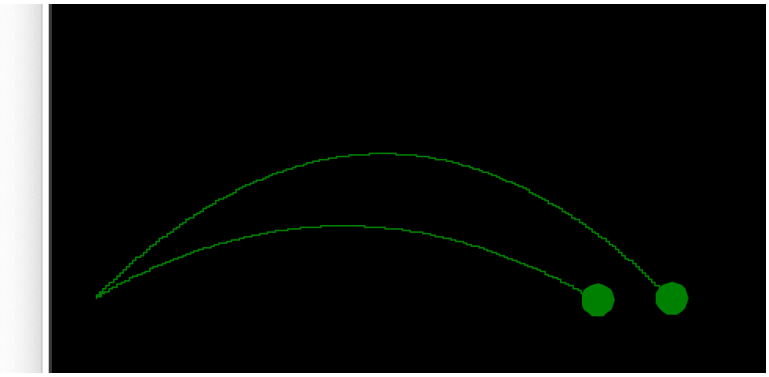


Velocity	Angle	Calculated time taken(s)	Algorithm calculated time taken (s)
60	45	8.65(2.d.p)	8.649999999999988
40	45	5.80(2.d.p)	5.799999999999987

Here we kept the angle the same but we changed the velocity, this mathematically should mean the path starts the same and follows the same line as the direction it is projected is the same however the lower initial velocity means it will reach a speed of 0 quicker (for same acceleration of gravity), which is shown by the smaller curve reaching a maximum earlier and overall not travelling as far. This combined with the results in the table showing the algorithm calculated extremely accurate answers, the test was successful.

Test 2: Constant velocity, varied angle:

```
enter initial velocity60
enter angle of elevation45
time taken 8.649999999999988
enter initial velocity60
enter angle of elevation30
time taken 6.149999999999986
enter initial velocity
```



Velocity	Angle	Calculated time taken(s)	Algorithm calculated time taken (s)
60	45	8.65(2.d.p)	8.649999999999988
60	30	6.15(2.d.p)	6.149999999999986

Here again, you can see from the results the numbers are very accurate to what they should be. Also looking at the actual simulation you can see for a constant velocity but smaller angle you get path which reaches a maximum at a lower displacement. This makes sense as the horizontal component is constant and the height is only affected by vertical component of velocity with for a smaller angle will be smaller initially so for constant acceleration (gravity) this will reach 0 velocity quicker so at lower displacement.

Overall both tests were passed so this algorithm meets requirements of the solution required as well as the accuracy required so this is complete.

Section 3.2: integrating simulator into main algorithm:
Now we've successfully created the simulator section of the final solution, the final step is to add this to the main solution in a way which allows the user to make full use of the algorithm in an easy to use way that meets the original requirements of the simulator section of the final solution.

```
from tkinter import *
from tkinter import ttk
import tkinter as tk
import sqlite3
from tkinter import font
import turtle
from math import sin, cos, radians, log
#libraries imported
def simulator(*args):
    root.geometry("685x600")
#set the dimensions of the window

    mainframe = ttk.Frame(root,padding= "100 200 100 200")
    subframe = ttk.Frame(mainframe,padding= "100 50 100 50",relief='sunken')
#padding adds extra space around the widget
#create the frame for the window
    mainframe.grid (column=0, row=0, sticky=(N, W, E, S))
    subframe.grid (column=2, row=15, sticky=(N, W, E, S))
#sticky means how the widget would line up within the grid cell
    mainframe.columnconfigure(0, weight=5)
    mainframe.rowconfigure(0, weight=5)
    subframe.columnconfigure(0, weight=5)
    subframe.rowconfigure(0, weight=5)
    ttk.Button(subframe, text="HOME", command=home).grid(column=2, row=11, sticky=S)
    cv = tk.Canvas(mainframe,width=200,height=200)
    t = turtle.RawTurtle(cv)
    velocity=60
    angle=radians(45)
##    run(velocity,angle)
##def run(velocity,angle):
    wn=turtle.Screen()
    wn.bgcolor("black")
    wn.title("bouncing ball")

    ball= turtle.Turtle()
    ball.shape('circle')
    ball.color('green')
    ball.penup()
    ball.speed(0)
    ball.goto(-330,-10)
    ball.pendown()
    ball.dy=0

    g=0.1
    time = 0


    while ball.ycor() >= -10:
        time = time + 0.05
        print(time)
```

To begin with I simply added the code for the simulation into the ‘simulator’ function which I had originally created in section 2 which is called when the button ‘simulator’ on the home page is pressed. So now, when the button is pressed you don’t have a blank window. Also a ‘home’ button has been added to return to the home page of the user interface.
Whilst this does work, the program immediately runs the simulation, not on user request like the specification requires. Also the user cannot control or vary any variables such as velocity or the angle meaning this does not yet sufficiently meet the requirements.

section 3.2.1:
Below is the first attempt at getting the simulator function to run on the users request. The principle idea to this attempt was to use parameters that will feed the value inputted by the user into the function each time and run it on the users request. In the example above not that there is only one parameter ‘velocity’, there is going to also be a second one for varying the angle too however for the purpose of testing and developing a working solution were only working with one to keep it simpler, and then add the other parameter once we find a working solution. The use of parameters here was an essential decision as it allows us to efficiently use and manipulate variables in separate functions.

```
#libraries imported
def get_velocity():
    value=velocity
    run_simulator(value)

def simulator(*args):
    root.geometry("685x600")
#set the dimensions of the window

    mainframe = ttk.Frame(root,padding= "100 200 100 200")
    subframe = ttk.Frame(mainframe,padding= "100 50 100 50",relief='sunken')
#padding adds extra space around the widget
#create the frame for the window
    mainframe.grid (column=0, row=0, sticky=(N, W, E, S))
    subframe.grid (column=0, row=15, sticky=(N, W, E, S))
#sticky means how the widget would line up within the grid cell
    mainframe.columnconfigure(0, weight=5)
    mainframe.rowconfigure(0, weight=5)
    subframe.columnconfigure(0, weight=5)
    subframe.rowconfigure(0, weight=5)
    global velocity
##declared as a global variable to be accessed outside of function
    velocity=int()
##making the content of the variable an integer data type
    ttk.Button(subframe, text="HOME", command=home).grid(column=0, row=11, sticky=S)
    ttk.Button(subframe, text="GO", command=get_velocity).grid(column=0, row=11, sticky=S)
    velocity_scale = tk.Scale(mainframe, orient=HORIZONTAL, length=200, from_=0.0, to=100.0,variable=velocity).grid(column=0, row=12, sticky=S)


def run_simulator(velocity):
    root=tk.Tk()
    cv = tk.Canvas(root,width=200,height=200)
    t = turtle.RawTurtle(cv)

    angle=radians(45)

    wn=turtle.Screen()

    wn.bgcolor('#FFD700')
    wn.title("projectile")

    ball= turtle.Turtle()
    ball.shape('circle')
    ball.color('#DC143C')
    ball.penup()
    ball.speed(0)
    ball.goto(-330,-10)
    ball.pendown()
    ball.dy=0
```

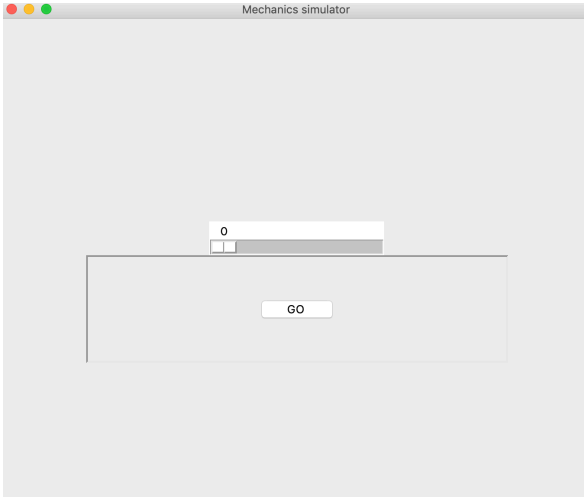
Its also a clear format to understand so makes the code more reusable too. To do this we needed to create 2 more functions, one which separately holds the algorithm for the simulator (‘run_simulator’), and one used to call this function with the value entered by the user being passed as the parameter (‘get_velocity’). Then we had to consider how we were going to get the user input:

```
velocity_scale = tk.Scale(mainframe, orient=HORIZONTAL, length=200, from_=0.0, to=100.0,variable=velocity).grid(column=0, row=12, sticky=S)
```

This line above shows how I did this. I decided to use the scan widget to receive the user input. The justification for this choice is that it allowed me to restrict the values the user ca input which was important as extremely large numbers for velocity will result in the simulation being too large and go off the canvas. Also thinking ahead to the angle init, this must be limited as it can only be between 0-180. Bearing this in mind other widgets such as a list box could have been use however ignorer to better meet the requirement of it being user friendly, a scale made more sense and there are many different value that can be inputted (for velocity anything between 0-100), this would result in an extremely long list and therefor very inconvenient for the user to use, whereas a scale widget can simply just be slid to the correct value very efficiently.

The way this widget works is first we use ‘tk.scale’ to create the widget and then I made it a horizontal slider, the I set the length to 200 to ensure it was big enough and each increment was big enough that finding a specific value won’t be hard. I then had to set the range of value it could take and where to the starting position should be (0). Finally I

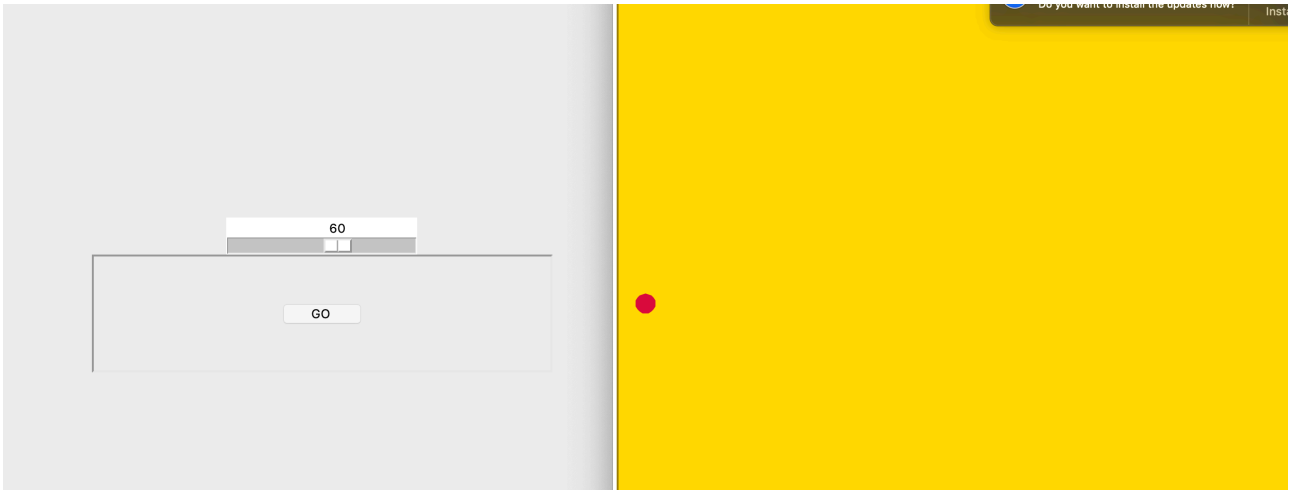
assigned the value in the scale at a given moment to be stored in the ‘velocity’ variable. The result of running this is shown below:



Here you can see the widget which is reasonable in size and also shows the value above it to aid the user. There Is not title yet however but these design elements will be added soon.

The way this now works is that the velocity variable was declared as a global variable to access it outside of the function. When the ‘Go’ button is pressed, the ‘get_velocity’ function is called which then runs the ‘run_simulator’ function but passes the current value stored in the variable velocity (the value on the slider in the image above) as the parameter. This value is the used in the function to run the simulation algorithm from earlier however with this particular value of velocity that the user inputted.

The results of running this are shown below:



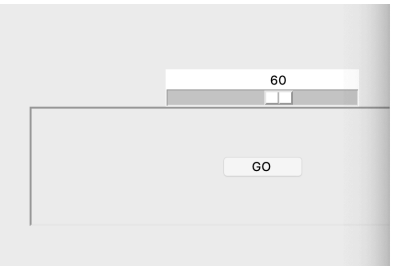
As you can see this test failed as the user input was 60 and if you look back to section **3.1 (attempt 2),** we ran a test of velocity at 60 and it was successful with the simulation shown, this does not match the above image and it appears nothing happens.

Looking at the result of running the code we can see the window for the simulation was created along with the ball and so the structure of the 2 new function being added does work as the ‘get_velocity’ function has called the ‘run_simulation’ function which produced the window above. Also no error message was printed so the issue is therefor in the loop shown again below:

```
while ball.ycor() >= -10:
    print(time)
    time=time+0.05
    print(time)
    changeInX = (velocity*cos(angle)) * time
    changeInY = ((velocity*sin(angle)) * time) + (0.5*-9.81*time*time)
    ball.sety(-10+changeInY)
    ball.setx(-330+changeInX)
```

This is the same code from earlier and hasn't been altered except I added print statements to help me find the error in the code.

Now if we run the same test again but with the print statements we see the following result:




```
>>>
============== RESTART: /Users/adam
>>> 0
0.05
```

Here we see only 2 times printed. The first indicates the first print statement which tells me the the while loop did run and wasn't bypassed and the second shows the value of time just before calculation were carried out. As this was correct and not still 0 it meant the logical error was in the velocity variable. When I then called the variable on its own it appeared to be 0 (shown on the left). This allowed me to conclude that the loop was not repeating due to velocity being 0 and essentially nullifying the equations as it produces 0 change in x and a negative change in y, making the while condition no longer true and the loop breaks.

```
>>> 0
0.05
velocity
0
>>>
```

Section 3.2.1 (Attempt 2):
Now I have established the error in my program, my next step to solve it is to use a different method, one that we've already used in **section 1** when connecting to the database and adding a user to it, the .get() method. As this has already worked before in my program it made sense to try it and also it is good practise to use this as it is specifically designed to retrieve the value held by a variable which is exactly what we are doing. Below is the code for this:

```
def velocity_angle():
    simulators(velocity.get(),angle.get())
```

Here we changed the function 'get_velocity' to 'velocity_angle' as now we've also added the parameter for altering the angle through user input, in addition the code analysed above for the scale widget has been duplicated but for the 'angle' variable. Here the code is kept efficient by now only using one line as we simply call the function and each parameter uses the .get() to retrieve the current values and they are passed straight into the 'run_simulator' function. The results of running this are shown below:

```
>>> Exception in Tkinter callback
Traceback (most recent call last):
  File "/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/tkinter/
__init__.py", line 1705, in __call__
    return self.func(*args)
  File "/Users/adamsatvilker/Documents/GUI8.py", line 10, in velocity_angle
    simulators(velocity.get(),angle.get())
AttributeError: 'NoneType' object has no attribute 'get'
```

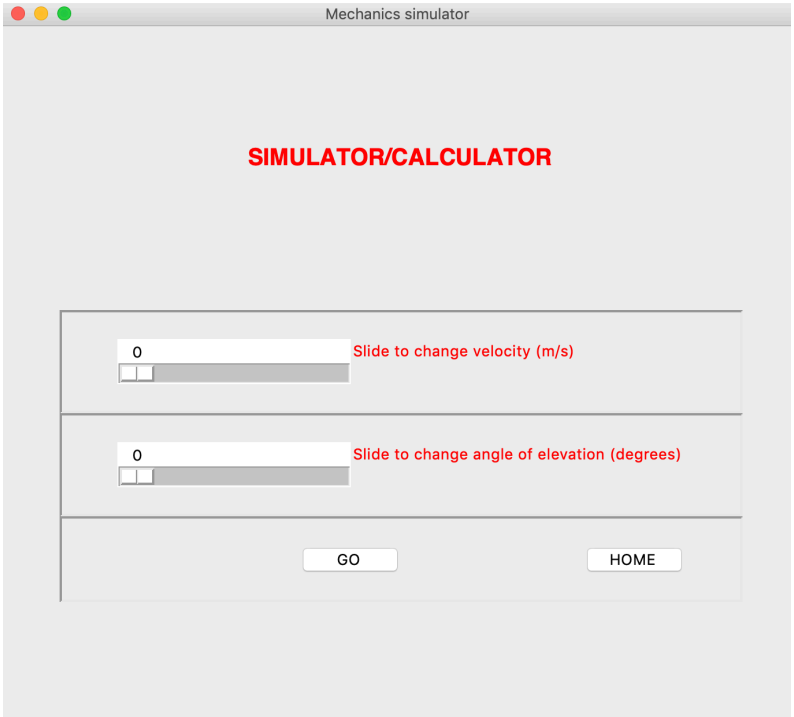
This syntax error occurs and the program breaks. From reading the error message we can see that the error is in the use of the .get() method and that it was being used wrong. First I compared this to how I previously used it in my program where it worked and there was no particular differences that would cause this error. When I researched into the error online I found in a forum that it was connected to the use of '.grid()' and 'tk.scale' on one line which we explored earlier in **section 3.2.1**. This was simply a formatting error as when you use the two methods on the same line, it automatically assigns 'None' to the variable connected to the scales ('velocity' and 'angle') and this means that the variables hold no value resulting in this error above as the .get() method couldn't retrieve a value that wasn't there. The simple solution was to separate this into 2 lines as shown below:

```
    global velocity
##global variable declared
    velocity = tk.Scale(velocity_frame, orient=HORIZONTAL, length=200, from_=0.0, to=100.0)
##scale widget created assigning its value to the variable velocity
    velocity.grid(column=0, row=0, sticky=N)
##used to position the widget on the GUI window
    global angle
    angle = tk.Scale(angle_frame, orient=HORIZONTAL, length=200, from_=0.0, to=180.0)
    angle.grid(column=0, row=0, sticky=N)
```

Along with this change, I also changed the design and added elements to the window to make it more user friendly which is shown below:

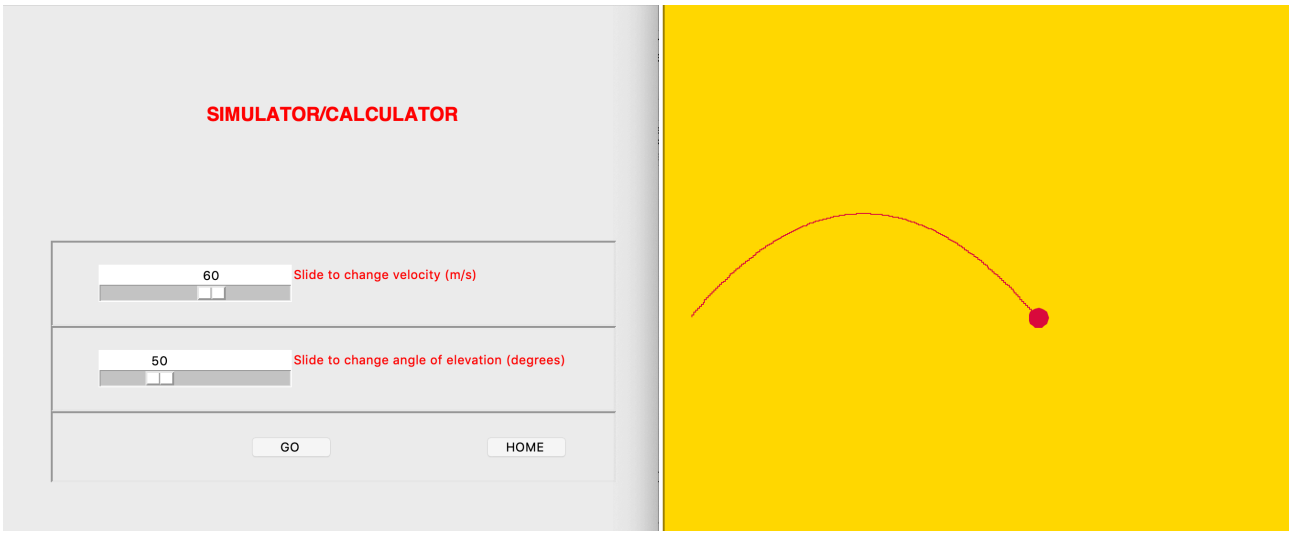
```
def velocity_angle():
    simulators(velocity.get(),angle.get())


def simulator(*args):
    root.geometry("685x600")
#set the dimensions of the window

    mainframe = ttk.Frame(root,padding= "50 100 50 100")
    subframe = ttk.Frame(mainframe,padding= "50 25 50 25",relief='sunken')
    velocity_frame= ttk.Frame(mainframe,padding= "50 25 50 25",relief='sunken')
    angle_frame= ttk.Frame(mainframe,padding= "50 25 50 25",relief='sunken')
#padding adds extra space around the widget
#create the frame for the window
    mainframe.grid (column=0, row=0, sticky=(N, W, E, S))
    subframe.grid (column=0, row=4, sticky=(N, W, E, S))
    velocity_frame.grid (column=0, row=2, sticky=(N, W, E, S))
    angle_frame.grid (column=0, row=3, sticky=(N, W, E, S))
#sticky means how the widget would line up within the grid cell
    mainframe.columnconfigure(0, weight=5)
    mainframe.rowconfigure(0, weight=5)
    subframe.columnconfigure(0, weight=5)
    subframe.rowconfigure(0, weight=5)
    font_change = font.Font(family='Helvetica', size=20, weight='bold')
##adding a new font style to use on labels below
    ttk.Label(mainframe, text="SIMULATOR/CALCULATOR",font=font_change,foreground='red').grid (column=0, row=0, sticky=(N))
    ttk.Label(velocity_frame, text="Slide to change velocity (m/s)", foreground='red').grid (column=3, row=0, sticky=(N))
    ttk.Label(angle_frame, text="Slide to change angle of elevation (degrees)", foreground='red').grid (column=3, row=0, sticky=(N))
##displays text on window
    ttk.Button(subframe, text="HOME", command=home).grid(column=0, row=11, sticky=S)
    global velocity
##global variable declared
    velocity = tk.Scale(velocity_frame, orient=HORIZONTAL, length=200, from_=0.0, to=100.0)
##scale widget created assigning its value to the variable velocity
    velocity.grid(column=0, row=0, sticky=N)
##used to position the widget on the GUI window
    global angle
    angle = tk.Scale(angle_frame, orient=HORIZONTAL, length=200, from_=0.0, to=180.0)
    angle.grid(column=0, row=0, sticky=N)
    ttk.Button(subframe, text="GO", command=velocity_angle).grid(column=0, row=11, sticky=S)
    ttk.Button(subframe, text="HOME", command=home).grid(column=1, row=11, sticky=S)
##buttons created and connected to other functions
```

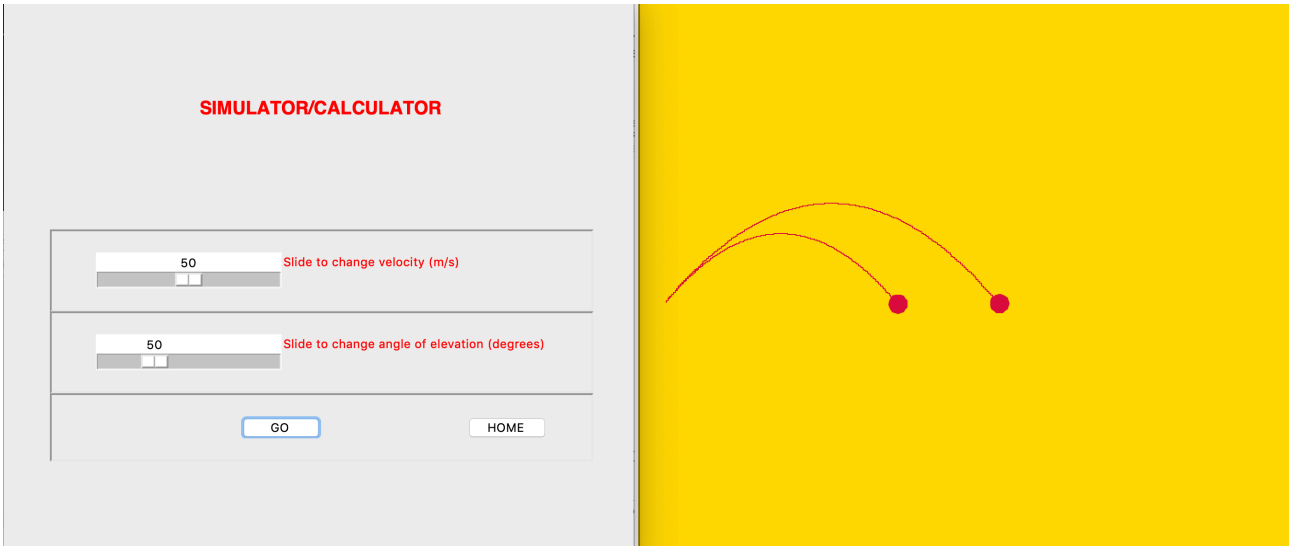
Running this now produces the window:



Here, the use of ‘relief = sunken’ allows the 3 sections to be divided making the design clearer and tidier. Also text label widgets have been used to help guide the user and make use of the program. The ‘home’ button adds convenience and a way to return to the home window, vital in making the interface easy to navigate.

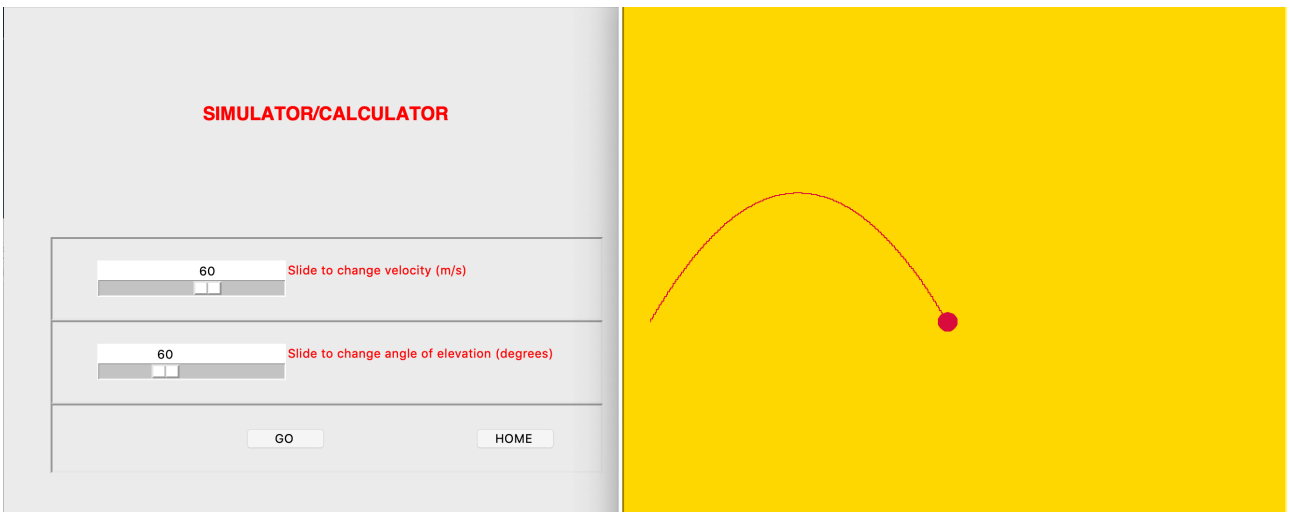
Test for pressing GO:
Test 1: Varying speed with constant angle:

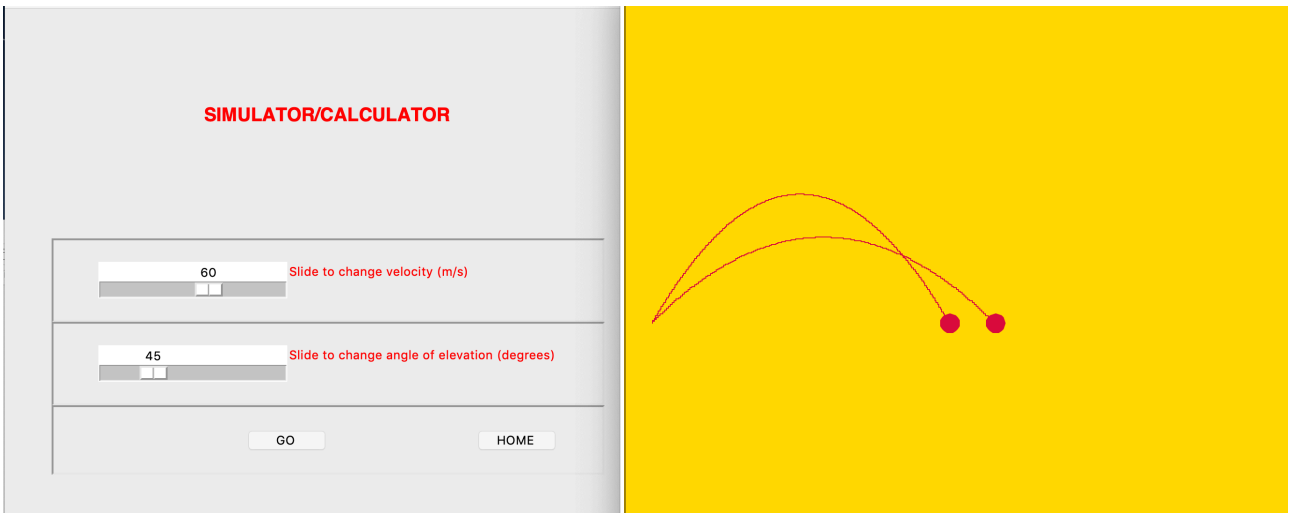




The results above are as expected a show the simulator to be working successfully on user request so far.

Test 2: Varying angle, constant velocity:





Test 2 continued:
Here we can see the test is successful and allows the user to vary the angle and see the correct results. Overall both tests were successful proving that the solution allows the user to change both velocity and angle and on request run the simulation, meeting the requirements of the program.

Section 3 review:

Test	What happens	Meet requirements?
Clicking Test	A new window is created	Yes
Selecting free fall, flat plane or projectile test		
For each of three test sections have questions displayed		
For each of three test sections have text entry boxes for users to input answer		
Click submit answers		
Be able to view results and total score		
Be able to go back or return home		
Clicking Graphs	A new window is created	Yes
Choose between graphs		
View example of each graph		
Clicking Simulator	A new window is created	Yes
Slider for altering velocity	A scale widget acts as slider, move slider to desired velocity and press ‘go’.	Yes
Slider for altering angle of elevation	A scale widget acts as slider, move slider to desired angle and press ‘go’.	Yes

Test	What happens	Meet requirements?
Click go and simulation runs	The values on sliders used to run simulation, new window pops up with simulation running	Yes
Can run simulation numerous times on user request	Every time user presses ‘go’ simulation runs	Yes
Can return home from simulator page	Click ‘home’ button to return to home page	Yes
Inputting text for username	Data is stored in variable however when clicking enter nothing happens.	Yes
Add username to database	The record is added for the username inputted when enter button clicked	Yes
Store data for each user and results		
Is user friendly (/10)		

Section 3 is now complete and the above table represents this as all the requirements involving the simulator have now been meet with Justification of how. Overall this section was very successful as no requirements were compromised or unable to be met, and whilst there were issue at different stages, a combination of debugging, inferring, deducing and online research allowed these problems to be resolved. We also looked to continuously keep the interface user friendly and considered this when designing how the user can input data. This seemed to start with, as the most tricky part of the solution and now it is successfully finished we can continue knowing the hardest and most fundamental section is complete. Also, as we continue we have time to allow for possible new ideas or improvements to come to mind and we can therefor improve it as it is a focal point of the solution which justifies why it has been done first, ahead of the other two sections.

Section 4: Test manager:

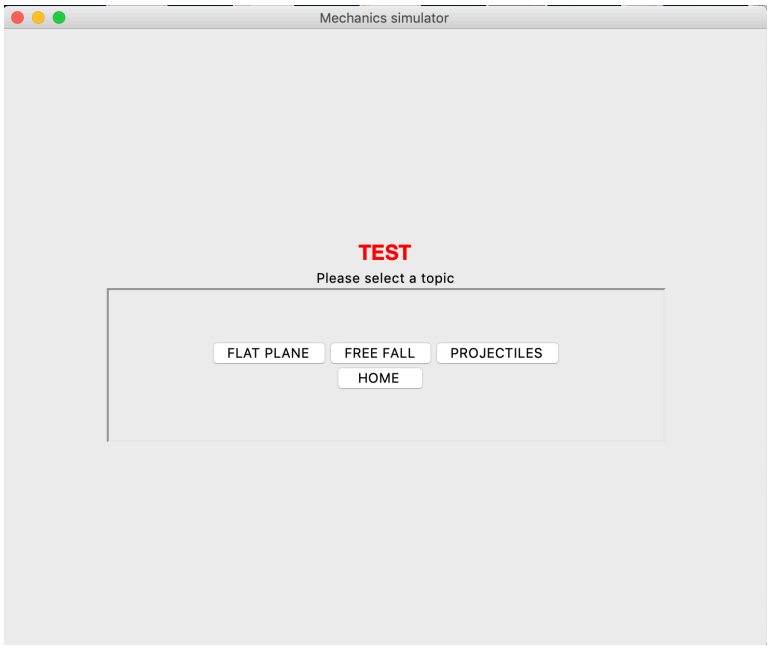
The next key stage of the development phase was to implement a test manager which asks the user questions provides a score for each test to allow the user to test their skills. This particular section will require use of the database created in **Section 1.** It will also be complex in nature as there will be three different sections to the test, testing three different topics however the code will be in theory the same for each and be consistent for all three to improve reusability and avoid over complicated code.

Section 4.1:

To begin with, I have used 3 functions separate to the main test function ‘Test_q’ which can be used to branch to the other three functions depending on the requested topic.

```
def Test_q(*args):
    root.geometry("685x600")
#set the dimensions of the window
    mainframe = ttk.Frame(root,padding= "100 200 100 200")
    subframe = ttk.Frame(mainframe,padding= "100 50 100 50",relief='sunken')
#padding adds extra space around the widget
#create the frame for the window
    mainframe.grid (column=0, row=0, sticky=(N, W, E, S))
    subframe.grid (column=2, row=15, sticky=(N, W, E, S))
#sticky means how the widget would line up within the grid cell
    mainframe.columnconfigure(0, weight=5)
    mainframe.rowconfigure(0, weight=5)
    subframe.columnconfigure(0, weight=5)
    subframe.rowconfigure(0, weight=5)

    font_change = font.Font(family='Helvetica', size=20, weight='bold')
    ttk.Label(mainframe, text=" TEST ",font=font_change, foreground='red').grid (column=2, row=1, sticky=(N))
    ttk.Label(mainframe, text="Please select a topic").grid (column=2, row=5, sticky=(N))
    ttk.Button(subframe, text="FLAT PLANE", command=Test_q_flat).grid(column=1, row=10, sticky=S)
    ttk.Button(subframe, text="FREE FALL", command=Test_q_free).grid(column=2, row=10, sticky=S)

    ttk.Button(subframe, text="PROJECTILES", command=Test_q_projectile).grid(column=3, row=10, sticky=S)
    ttk.Button(subframe, text="HOME", command=home).grid(column=2, row=11, sticky=S)

def Test_q_flat(*args):
    root.geometry("685x600")
#set the dimensions of the window
    mainframe = ttk.Frame(root,padding= "100 200 100 200")
#padding adds extra space around the widget
#create the frame for the window
    mainframe.grid (column=0, row=0, sticky=(N, W, E, S))
#sticky means how the widget would line up within the grid cell
    mainframe.columnconfigure(0, weight=5)
    mainframe.rowconfigure(0, weight=5)

def Test_q_free(*args):
    root.geometry("685x600")
#set the dimensions of the window
    mainframe = ttk.Frame(root,padding= "100 200 100 200")
#padding adds extra space around the widget
#create the frame for the window
    mainframe.grid (column=0, row=0, sticky=(N, W, E, S))
#sticky means how the widget would line up within the grid cell
    mainframe.columnconfigure(0, weight=5)
    mainframe.rowconfigure(0, weight=5)

def Test_q_projectile(*args):
    root.geometry("685x600")
#set the dimensions of the window
```

Looking at the first function ‘Test_q’, I have added a design which is consistent with the main home page in that it makes use of a mainframe and subframe. There are buttons created that link to the three branches of the test section (3 topics). The results are shown below:

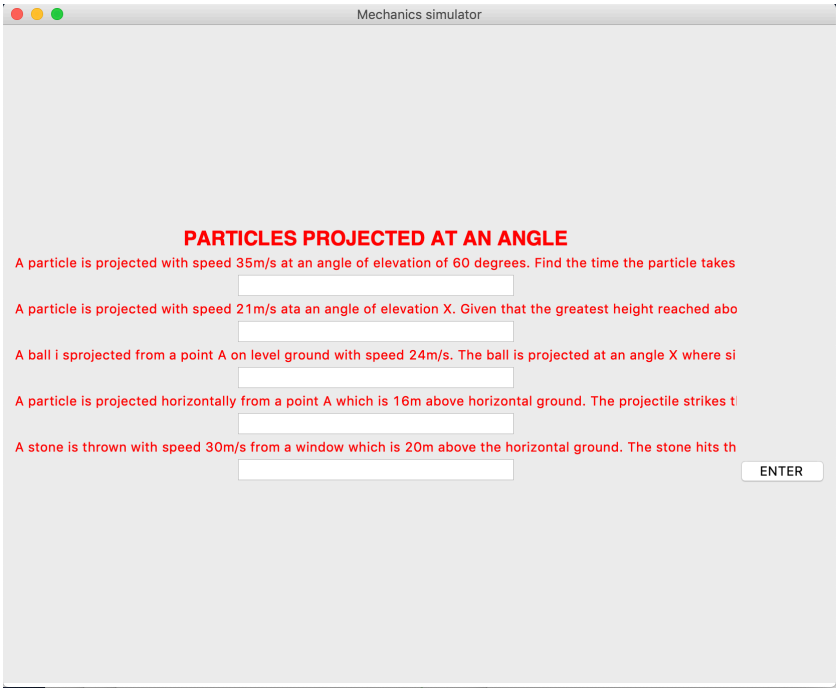


Section 4.2:
The next step is to create the window that appears when one of the three topics are selected. This will be generic for all three however each will be different in that it will be related to that specific topic (e.g. the questions displayed will be different). The requirements of this window are to provide a way of inputting answers to the corresponding question which will also be displayed. Thinking ahead, this will mean interactions with the database again so will require the imported library of 'sqlite3'. As they will be very similar windows following the same structure. I am coding one (in this case the 'projectiles') and once this is fully working we will duplicate and adjust it for the other two.

```
def Test_q_projectile(*args):
    root.geometry("685x600")
#set the dimensions of the window

    mainframe = ttk.Frame(root,padding= "100 200 100 200")
#padding adds extra space around the widget
#create the frame for the window
    mainframe.grid (column=0, row=0, sticky=(N, W, E, S))
#sticky means how the widget would line up within the grid cell
    mainframe.columnconfigure(0, weight=5)
    mainframe.rowconfigure(0, weight=5)
    font_change = font.Font(family='Helvetica', size=20, weight='bold')
    ttk.Label(mainframe, text="PARTICLES PROJECTED AT AN ANGLE",font=font_change,foreground='red').grid (column=0, row=0, sticky=(N))
    import sqlite3
##imports library for using sql
    value= "simulator10.db"
    conn = sqlite3.connect(value)
##connects to database
    c=conn.cursor()
##creates cursor
    for row in c.execute('SELECT question_num, question, answer FROM Questions WHERE question_type==3'):
##sql statement for querying database with a condition
        question=row[1]
        a=row[0]
        i=row[0]
##stores the contents of the item in the given position of the row in the above variables
        i=ttk.Label(mainframe, text=question,style='Mycolour.TLabel').grid (column=0, row=i*2, sticky=(N))
##code for creating label is stored in variable above
    global a_1
##declares variable as global to later access outside function
    a_1=StringVar()
##sets the data type for the variable
    a_1_entry = ttk.Entry(mainframe, width=30, textvariable=a_1)
    a_1_entry.grid(column=0, row=3, sticky=(N))
##creates a widget to enter answer and stores it in variable
    global a_2
    a_2=StringVar()
    a_2_entry = ttk.Entry(mainframe, width=30, textvariable=a_2)
    a_2_entry.grid(column=0, row=5, sticky=(N))
    global a_3
    a_3=StringVar()
    a_3_entry = ttk.Entry(mainframe, width=30, textvariable=a_3)
    a_3_entry.grid(column=0, row=7, sticky=(N))
    global a_4
    a_4=StringVar()
    a_4_entry = ttk.Entry(mainframe, width=30, textvariable=a_4)
    a_4_entry.grid(column=0, row=9, sticky=(N))
    global a_5
    a_5=StringVar()
    a_5_entry = ttk.Entry(mainframe, width=30, textvariable=a_5)
    a_5_entry.grid(column=0, row=11, sticky=(N))
    ttk.Button(mainframe, text="ENTER", command=results_projectiles).grid(column=2, row=11, sticky=S)
```

As shown above, I coded this in the function 'test_q_projectile' which is called when the button 'projectiles' is pressed. First we connected to the database and then we created a 'for loop', this containing an SQL statement. The SQL statement looks at the table 'questions' using the command 'FROM' and selects all the fields stated after the command 'SELECT' however it only does this for rows which have the field 'question_type' equal to '3' which is done using the command 'WHERE'. So as a result the SQL statement extracts these particular rows and the stated fields then the loop runs through each row Individually. Each row is essentially a list so we create the variable question to hold the contents at the position 1 in the row/list. We created the variables 'i' which using the same technique to hold the value of 'question_num' which would be the

first item in the list. We then reassign ‘i’ to a label which says : ‘text=question’ and this means it will display the contents of this variable question which we just assigned to hold the value of the second item (the question) for that row. We also changed the font using the style method previously discussed. And when using the grid formatting we use ‘i*2’. The contents of ‘i’ at this point is the question number (question_num) so will be 1-5 and the use of multiply this by 2 means it will display this label every 2 rows, leaving a row in-between for the user input which we are about to go on to. Now that we have the questions we need to create the user inputs for each of the 5 questions. First we declared the variable as global so we can access it outside the function when checking if it is correct. We use the variable name ‘a_1’ which means answer for question 1. And then the next 4 questions follow the same format to make it easy to understand. First we declared the data type as this will treat the user input as a string which we need for later when comparing to see if answer matches the one in the database. Then we create an entry widget where users can input their answer, has a width of 30 (‘width=30’) and then we store their input in the variable just created above (e.g. a_1) with the command (textvariable=a_1). Finally for each, we use .grid to position these inept boxes, notice the odd numbers for the rows as this will position them in-between the questions. Then, we created the button ‘enter’ which will take us to the results page to see our score and performance. The function for this is what we code next and it is shown below in **section 4.3.** The result of running the code above is shown below:



Here we can see that the manipulation of the database has been successful as the question have been displayed and the entry widgets have also successfully been created. However the obvious issue is the formatting as the questions are being cut off. As this is only a formatting issue I am going to continue to compel the results window after as that will tell me wether the entry boxes are successfully storing each answer and I’m able to verify if each one is correct which is the main requirement. Once these main requirements are met successfully then I will return to this issue to resolve it.

Section 4.3:

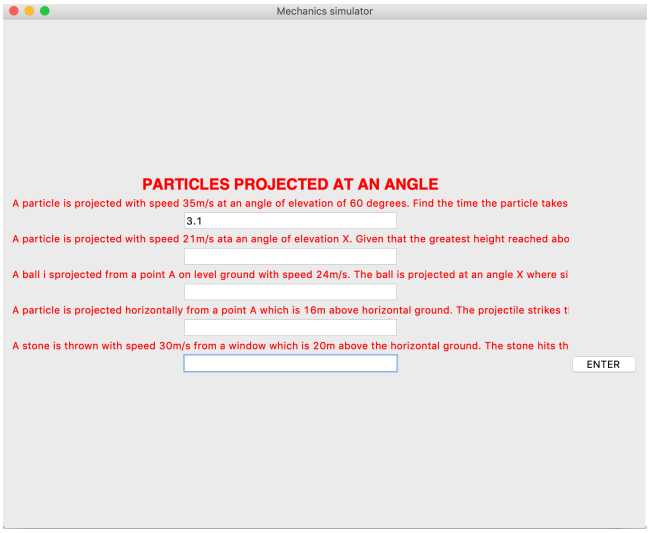
Now we move on to developing the window which follows the one from section 4.2. The aim of this section is to compare the users answer that was inputted to the answer stored in the database and if correct account for this in the score. This window will display the results for the test. The name of this function is ‘results_projectiles’ and again it will be generic for all three topics however each will be adapted to each topic so again we'll code it for one (in this case projectiles) and the duplicate it later for all 3 topics. First we

need to implement a design for the window and add the text labels then we verify each answer, displaying for each question if correct and then the final score. The code that does this is shown below:

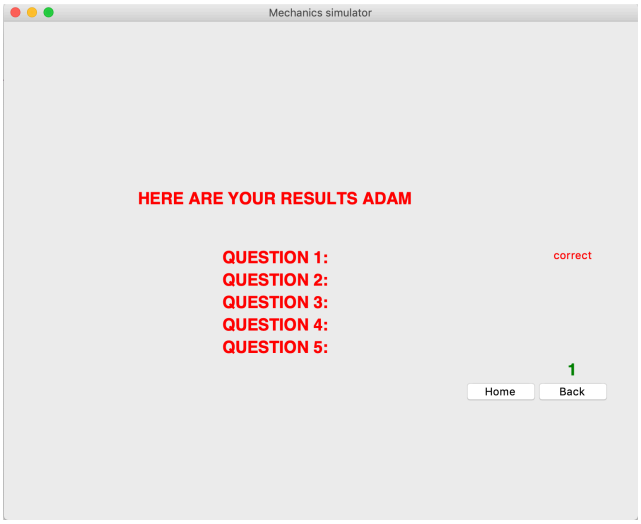
```
    count=0
##set count variable to 0
    name_upper=name.get().upper()
##retrieve the value of name and make it upper case
    font_change = font.Font(family='Helvetica', size=20, weight='bold')
##formatting for font change
    ttk.Label(mainframe, text="HERE ARE YOUR RESULTS "+name_u,font=font_change,foreground='red').grid (column=0, row=0, sticky=(N))
    ttk.Label(mainframe, text="QUESTION 1:",font=font_change,foreground='red').grid (column=0, row=1, sticky=(N))
    ttk.Label(mainframe, text="QUESTION 2:",font=font_change,foreground='red').grid (column=0, row=2, sticky=(N))
    ttk.Label(mainframe, text="QUESTION 3:",font=font_change,foreground='red').grid (column=0, row=3, sticky=(N))
    ttk.Label(mainframe, text="QUESTION 4:",font=font_change,foreground='red').grid (column=0, row=4, sticky=(N))
    ttk.Label(mainframe, text="QUESTION 5:",font=font_change,foreground='red').grid (column=0, row=5, sticky=(N))
    ttk.Button(mainframe, text="Back", command=Test_q_projectile).grid(column=2, row=7, sticky=S)
    ttk.Button(mainframe, text="Home", command=home).grid(column=1, row=7, sticky=S)

    for row in c.execute('SELECT question_num, question, answer FROM Questions WHERE question_type==3'):
## sql statement to query database
        if a_1.get() == row[2]:
##retrieves value in variable and compares if equal to value in the above position in the row
            c.execute('UPDATE results SET Q1=1 WHERE resultID=2')
##changes value in database
            ttk.Label(mainframe, text='correct',style='Mycolour.TLabel').grid (column=2, row=1, sticky=(N))
            count+=1
##adds 1 to count variable
        if a_2.get() == row[2]:
            c.execute('UPDATE results SET Q2=1 WHERE resultID=2')
            ttk.Label(mainframe, text='correct',style='Mycolour.TLabel').grid (column=2, row=2, sticky=(N))
            count+=1
        if a_3.get() == row[2]:
            c.execute('UPDATE results SET Q3=1 WHERE resultID=2')

            ttk.Label(mainframe, text='correct',style='Mycolour.TLabel').grid (column=2, row=3, sticky=(N))
            count+=1
        if a_4.get() == row[2]:
            c.execute('UPDATE results SET Q4=1 WHERE resultID=2')
            ttk.Label(mainframe, text='correct',style='Mycolour.TLabel').grid (column=2, row=4, sticky=(N))
            count+=1
        if a_5.get() == row[2]:
            c.execute('UPDATE results SET Q5=1 WHERE resultID=2')
            ttk.Label(mainframe, text='correct',style='Mycolour.TLabel').grid (column=2, row=5, sticky=(N))
            count+=1
    ttk.Label(mainframe, text=count,font=font_change,foreground='green').grid (column=2, row=6, sticky=(N))
##displays value currently in count

    conn.commit()
##saves changes to database
```

Here we first set count to 0, which is used to count the points throughout. The labels are standard labels with text to create the design of the window. Then you have the navigation buttons of ‘back’ and ‘home’. The the next part is where we query the database again, this line is exactly the same as the one we previously looked at. Then we have a series of ‘if statements’ which check each of the 5 answers inputted against the item in position 2 of that row (answer). It is checking every answer inputted against that one answer and if correct it will update the ‘results’ table to a ‘1’ for the field for that question (e.g.Q3) using the command ‘UPDATE’ and add 1 to count. Once repeated for every line selected it will break the loop and a text label is produced with the text being the value stored in the variable count which will be the total score. The results of running this are shown below:

The correct answer (3.1) is entered:



Then we press the 'enter' button which is calling the second function 'results_projectiles':



As shown on the right, there is a message displayed for the correct answer and only the correct answer. Then finally the total is produced in green. This is what was required to happen so the test was successful. Now we simply duplicate the code of this section and adapt it to hold the different questions for that topic. The topic free fall will have the 2 function names: 'test_q_free' and 'results_free'. Also when querying the database we will be using the condition: 'WHERE question_type==1' instead of 'WHERE question_type==3' which we saw earlier in this section. The same is goes for the free fall topic question who have the command 'WHERE question_type==2'. This focuses on manipulating the data in the database on that particular type of question (1-3 as there are 3 topics). The free fall topic will have the function names 'test_q_free' and 'results_free'.

Finally before moving on to the next section, we had a layout issue earlier in **section 4.2** which we have to fix in order to ensure it easy to use and convenient for user which is one of the requirements. The issue we had is the text was too long on some lines and therefor was being cut off meaning part of the questions were missing. To solve this issue I went back to the database from **section 1**, which holds the questions and on the lines which were too long I used '\n'. This, when read by the text label recognises it as an instruction to start a new line, an example from the database is:

```
Modelling the diver as a particle moving freely under gravity with initial speed 0, \n find the speed of the diver
```

Section 4 review:

Test	What happens	Meet requirements?
Clicking Test	A new window is created	Yes
Selecting free fall, flat plane or projectile test	The function for the test on the corresponding topic is called	Yes
For each of three test sections have questions displayed	When function is called, new window with question for that topic are displayed after being extracted from database using SQL	Yes
For each of three test sections have text entry boxes for users to input answer	Entry widgets are used to allow for user to enter data and store it in the corresponding variables	Yes
Click submit answers	This calls the next function for calculating and displaying results	
Be able to view results and total score	The .get() function is used to retrieve each answer which is compared against the values stored in the database for this questions, the variable count keeps a running total to keep track of total score which is displayed at the end as a label	Yes
Be able to go back or return home	a button widget is used twice, one to Call the ‘home’ function and the back button calls the function previous	Yes
Clicking Graphs	A new window is created	Yes
Choose between graphs		
View example of each graph		
Clicking Simulator	A new window is created	Yes
Slider for altering velocity	A scale widget acts as slider, move slider to desired velocity and press ‘go’.	Yes
Slider for altering angle of elevation	A scale widget acts as slider, move slider to desired angle and press ‘go’.	Yes
Click go and simulation runs	The values on sliders used to run simulation, new window pops up with simulation running	Yes
Can run simulation numerous times on user request	Every time user presses ‘go’ simulation runs	Yes
Can return home from simulator page	Click ‘home’ button to return to home page	Yes

Test	What happens	Meet requirements?
Inputting text for username	Data is stored in variable however when clicking enter nothing happens.	Yes
Add username to database	The record is added for the username inputted when enter button clicked	Yes
Store data for each user and results	The database has a 'results' table which stores the marks for that user on each question	Yes
Is user friendly (/10)		

Overall table above confirms that we have met all the requirements related to this 'test manager' section and so the section is now complete. This section was more straight forward than the last in that there wasn't any logical errors in my algorithms and the main skill that was required here was querying and manipulating the database. Now this section is complete the user has a way of testing their skills in each of the three topics and can see where they lack understanding when they view their results.

Section 5: Graph manager:
This is the third main section of the program, now that we've completed the test manager and simulator we have to add the final section which is the graph manager. The aim of this section is to provide a way lading student with the understanding of these key graphs (displacement/time and velocity/time). To do this the requirements state that the user should be able to view the graphs and be able to choose between the two so they can compare them. To do this I am going to use the imported library 'matplotlib' which is designed for creating graphs

Section 5.1:
To begin with I started with just trying to code the 'velocity/time' graph:

```
def run_Vt():
    Vt_graph(velocity1.get(),acceleration.get(),time1.get())
def Vt_graph(velocity,acceleration,time):
    import matplotlib.pyplot as plt
    x = []
    for i in range(0,time+1):
        x.append(i)
        i+=1

    # corresponding y axis values
    y = []
    for i in range(0,time+1):
##        print('at',i,'seconds')
        v=velocity+ acceleration*i

        y.append(v)
        i+=1

    # plotting the points
    plt.plot(x, y)

    # naming the x axis
    plt.xlabel('Time (s)')
    # naming the y axis
    plt.ylabel('velocity (m/s)')

    # giving a title to my graph
    plt.title('V/t graph')

    # function to show the plot
    plt.show()
```

Looking at the function ‘Vt_graph’, this is where I coded the actual graph. The parameters are for the values velocity, acceleration and time. These are the 3 variables needed in calculating the graphs and this will work in a similar way to the simulator in **section 3.** By this I mean that the user will input values for these variables then the function ‘run_Vt()’ will be called and this simply calls ‘Vt_graph()’. The need for this extra step is that it allows me to use .get() to retrieve the users values and pass them through as the parameters for the function which are then used to create the graphs. When creating the graph, time is on the x-axis and increments by 1 (1 second) so the first ‘for loop’ above represents the list of x-coordinates being created using .append() each increment up until the time that was the entered as there is no data after this so graph shouldn’t go further. One the x-axis is done, the next ‘for loop’ is for the y-coordinates. This follows a similar patter of appending to a list every increment of 1 up until the time that was entered as each 1 represents 1 second of time. The difference here however is to calculate the y-coordinate each time we needed a mathematical formulae which is for constant acceleration. This equation is V = U +at. This calculates current velocity (V) by adding acceleration multiplied by time (at) and initial velocity(U). Note that time here is going up by 1 each loop and this is what cause the velocity to change as acceleration and initial velocity are constant. Finally I used .plot() to take both lists and plot them on each axis. ‘xlabel’ and ‘ylabel’ allow me to add labels to each axis so user can understand what is being plotted as well as a title to clearly distinguish between this graph and the one that follows. .show() is what makes the graph display on the screen.
Next we created a similar algorithm with same structure but this time it is for the ‘displacement/time’ graph:

```
def run_St():
    St_graph(velocity1.get(),acceleration.get(),time1.get())
def St_graph(velocity,acceleration,time):
    import matplotlib.pyplot as plt
    x = []
    for i in range(0,time+1):
        x.append(i)
        i+=1

    # corresponding y axis values
    y = []
    for i in range(0,time+1):
##        print('at',i,'seconds')
        s=velocity*i + (0.5*acceleration*i*i)

        y.append(s)
        i+=1

    # plotting the points
    plt.plot(x, y)

    # naming the x axis
    plt.xlabel('Time (s)')
    # naming the y axis
    plt.ylabel('displacement (m)')

    # giving a title to my graph
    plt.title('S/t graph')
    plt.show()
```

As you can we have a function for creating the graph and a function specifically for retrieving user values the passing them as parameters when it call the graph creating function. Here the variables require are just the same so no change in parameters however now we're calculating displacement on y-axis, not velocity. To do this I have to make use of a different mathematical formula: s=Vt = 1/2at^2. This takes the initial velocity entered and and for every time increment of 1, calculates the new displacement. The labels have changed now to represent the displacement/time graph.

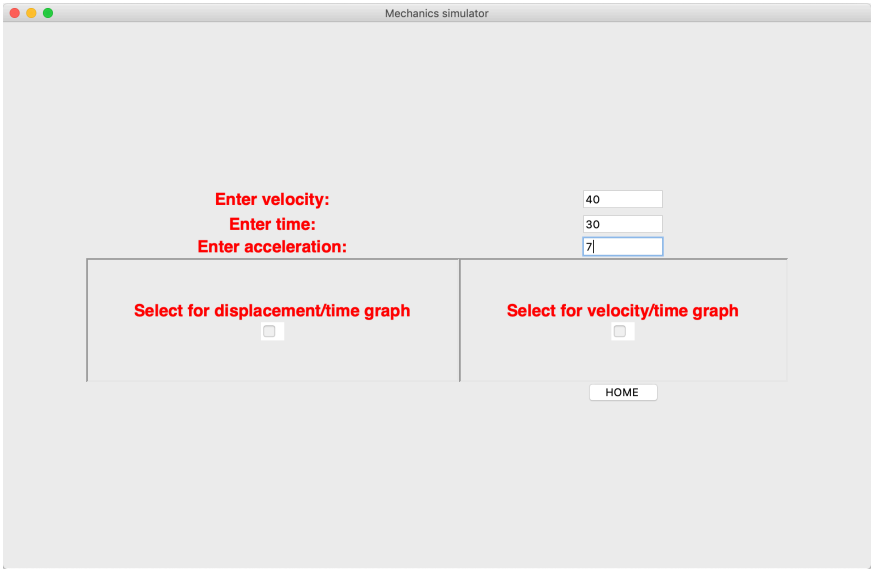
The code in the main window which is displayed when you first choose graphs:

```
def graphs(*args):
    root.geometry("685x600")
#set the dimensions of the window
    font_change = font.Font(family='Helvetica', size=20, weight='bold')
    mainframe = ttk.Frame(root,padding= "100 200 100 200")
    displacement_frame = ttk.Frame(mainframe,padding= "50 50 50 50",relief='sunken')
    velocity_frame = ttk.Frame(mainframe,padding= "50 50 50 50",relief='sunken')
#padding adds extra space around the widget
#create the frame for the window
    mainframe.grid (column=0, row=0, sticky=(N, W, E, S))
    displacement_frame.grid (column=0, row=3, sticky=(N, W, E, S))
    velocity_frame.grid (column=1, row=3, sticky=(N, W, E, S))
#sticky means how the widget would line up within the grid cell
    mainframe.columnconfigure(0, weight=5)
    mainframe.rowconfigure(0, weight=5)
    ttk.Label(mainframe, text=" Enter velocity: ",font=font_change, foreground='red').grid (column=0, row=0, sticky=(N))
    ttk.Label(mainframe, text=" Enter time: ",font=font_change, foreground='red').grid (column=0, row=1, sticky=(N))
    ttk.Label(mainframe, text=" Enter acceleration: ",font=font_change, foreground='red').grid (column=0, row=2, sticky=(N))
    global velocity1
    velocity1=IntVar()
    velocity1_entry = ttk.Entry(mainframe, width=10, textvariable=velocity1).grid(column=1, row=0, sticky=(N))
    global time1
    time1=IntVar()
    time1_entry = ttk.Entry(mainframe, width=10, textvariable=time1).grid(column=1, row=1, sticky=(N))
    global acceleration
    acceleration=IntVar()
    acceleration_entry = ttk.Entry(mainframe, width=10, textvariable=acceleration).grid(column=1, row=2, sticky=(N))

    ttk.Button(mainframe, text="HOME", command=home).grid(column=1, row=11, sticky=(N))
    ttk.Label(displacement_frame, text=" Select for displacement/time graph ",font=font_change, foreground='red').grid (column=1, row=0, sticky=(N))
    displacement_check= Checkbutton(displacement_frame, state=ACTIVE, command=run_St).grid (column=1, row=1, sticky=(N))
    ttk.Label(velocity_frame, text=" Select for velocity/time graph ",font=font_change, foreground='red').grid (column=1, row=0, sticky=(N))
    velocity_check= Checkbutton(velocity_frame, state=ACTIVE, command=run_Vt).grid (column=1, row=1, sticky=(N))
```

Here I decided to create 2 subframes which are held within the mainframe. 1 for velocity/ time and the other for displacement/time. The purpose of this is to separate the sections making it more organised and easier to use. The separated sections make it easier for user to quickly differentiate between the 2 and find what they want. I declared the 3 variables ‘velocity1’, ‘angle1’ and ‘acceleration’. (note that the 1 at the end of the variable names is to avoid clashes with the simulator modules that also use similar variables). They have to global as we saw above. We retrieve these variable outside of the function when passing parameters. The ‘Intvar()’ declares the variables as an integer datatype meaning it has an integer value and so when we pass it as a parameter and manipulate the values in the equation it is in the correct format. Also I introduced the use of the check button widget. This is ideal for the use of selecting the desired graph as it is a simple click of a box and a tick appears which tells the user they’ve selected it as apposed to a regular button. Also it is suited to when you have 2 distinct values (on or off) which is what we have here when selecting either one as each of the options are either selected or not selected so its also good practise to use this instead as the widget is designed for this situation.

```
    ttk.Button(mainframe, text="HOME", command=home).grid(column=1, row=11, sticky=(N))
    ttk.Label(displacement_frame, text=" Select for displacement/time graph ",font=font_change, foreground='red').grid (column=1, row=0, sticky=(N))
    displacement_check= Checkbutton(displacement_frame, state=ACTIVE, command=run_St).grid (column=1, row=1, sticky=(N))
    ttk.Label(velocity_frame, text=" Select for velocity/time graph ",font=font_change, foreground='red').grid (column=1, row=0, sticky=(N))
    velocity_check= Checkbutton(velocity_frame, state=ACTIVE, command=run_Vt).grid (column=1, row=1, sticky=(N))
```

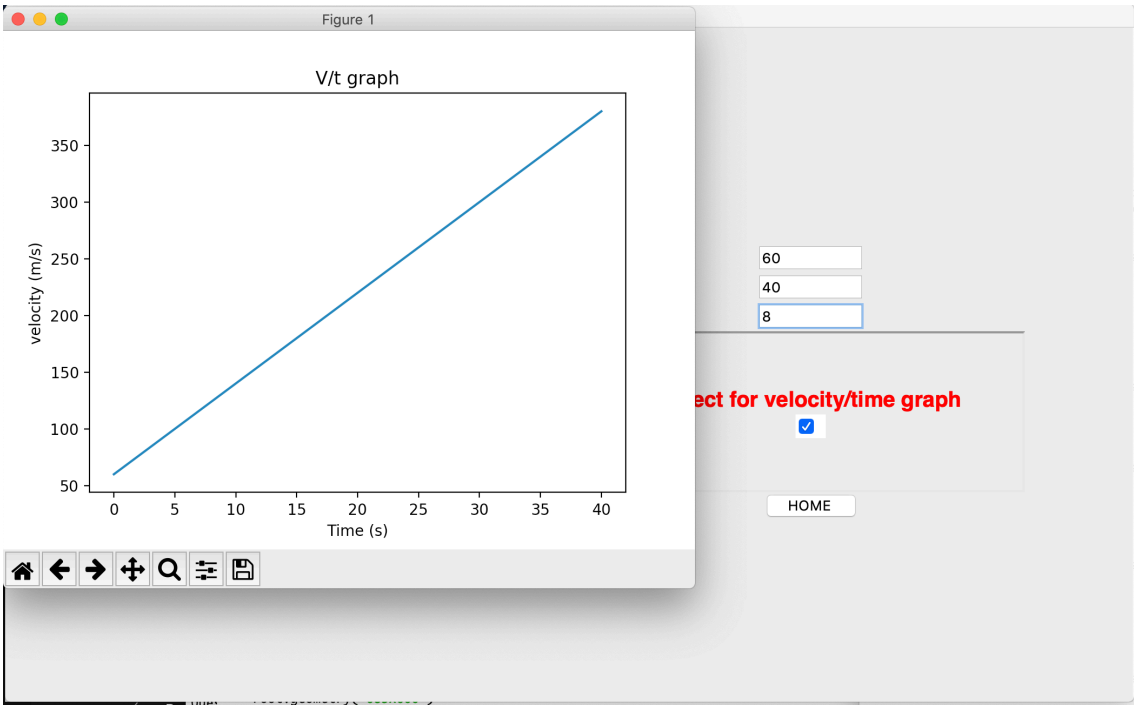
Above is an extraction of the code previously shown, where the check button is used. The use of ‘Checkbutton’ in front of the brackets tells the program it is a check button. So this creates the check button and then we configure it by first placing it in ‘velocity_frame’ and we set the ‘state’ to active which means the user will be able to click on it. Then we use ‘command’ followed by the desired function which is called when the check button is clicked on. In this case it is linked to the ‘run_St’ or ‘run_Vt’ functions which we looked at earlier in the section. Again we use .grid() to position the buttons within the frame. The results of running this are shown below:



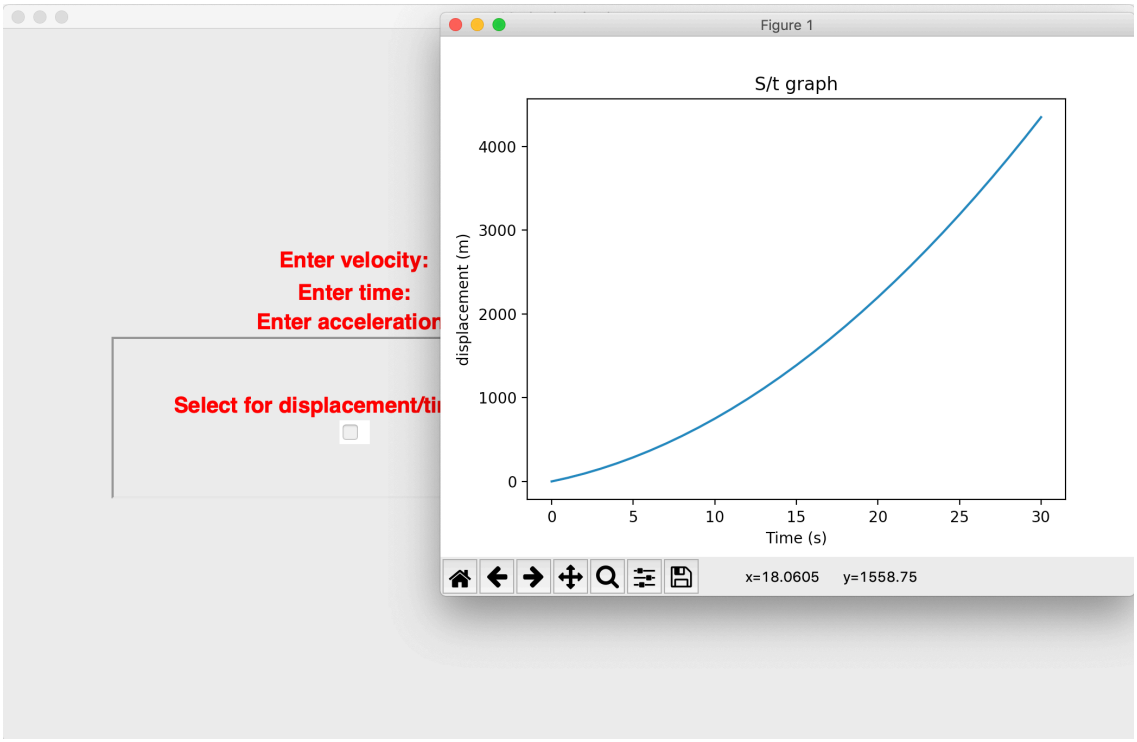
Here we can see what the main page for the graphs section look like. Here the segregation of the 2 option can be seen. Also we've inputted values for the three variables and now we are going to click the check buttons.

Test

When clicking the check button for velocity/time:



When clicking the check button for displacement/time:



Here we can see both graphs are successfully displayed with the axis correct and the shape also being correct so we can now say we have met the requirements of displaying the chosen graph to the user.

Review:

Test	What happens	Meet requirements?
Clicking Test	A new window is created	Yes
Selecting free fall, flat plane or projectile test	The function for the test on the corresponding topic is called	Yes
For each of three test sections have questions displayed	When function is called, new window with question for that topic are displayed after being extracted from database using SQL	Yes
For each of three test sections have text entry boxes for users to input answer	Entry widgets are used to allow for user to enter data and store it in the corresponding variables	Yes
Click submit answers	This calls the next function for calculating and displaying results	
Be able to view results and total score	The .get() function is used to retrieve each answer which is compared against the values stored in the database for this questions, the variable count keeps a running total to keep track of total score which is displayed at the end as a label	Yes

Test	What happens	Meet requirements?
Be able to go back or return home	a button widget is used twice, one to Call the ‘home’ function and the back button calls the function previous	Yes
Clicking Graphs	A new window is created	Yes
Choose between graphs	A check button widget is used to allow user to select either the v/t graph or s/t graph, calling the functions to run each	Yes
View example of each graph	The functions ‘v/t_graph’ and ’s/t_graph’ create these graphs using imported library matplotlib	Yes
Clicking Simulator	A new window is created	Yes
Slider for altering velocity	A scale widget acts as slider, move slider to desired velocity and press ‘go’.	Yes
Slider for altering angle of elevation	A scale widget acts as slider, move slider to desired angle and press ‘go’.	Yes
Click go and simulation runs	The values on sliders used to run simulation, new window pops up with simulation running	Yes
Can run simulation numerous times on user request	Every time user presses ‘go’ simulation runs	Yes
Can return home from simulator page	Click ‘home’ button to return to home page	Yes
Inputting text for username	Data is stored in variable however when clicking enter nothing happens.	Yes
Add username to database	The record is added for the username inputted when enter button clicked	Yes
Store data for each user and results	The database has a ‘results’ table which stores the marks for that user on each question	Yes
Is user friendly (/10)		

All sections have now been completed and so the development phase is now finished.

Final code:

```
from tkinter import *
from tkinter import ttk
import tkinter as tk
import sqlite3
from tkinter import font
import turtle
from math import radians, sin, cos
#libraries imported
def velocity_angle():
    simulators(velocity.get(),angle.get())


def simulator(*args):
    root.geometry("685x600")
#set the dimensions of the window

    mainframe = ttk.Frame(root,padding= "50 100 50 100")
    subframe = ttk.Frame(mainframe,padding= "50 25 50 25",relief='sunken')
    velocity_frame= ttk.Frame(mainframe,padding= "50 25 50 25",relief='sunken')
    angle_frame= ttk.Frame(mainframe,padding= "50 25 50 25",relief='sunken')
#padding adds extra space around the widget
#create the frame for the window
    mainframe.grid (column=0, row=0, sticky=(N, W, E, S))
    subframe.grid (column=0, row=4, sticky=(N, W, E, S))
    velocity_frame.grid (column=0, row=2, sticky=(N, W, E, S))
    angle_frame.grid (column=0, row=3, sticky=(N, W, E, S))
#sticky means how the widget would line up within the grid cell
    mainframe.columnconfigure(0, weight=5)
    mainframe.rowconfigure(0, weight=5)
    subframe.columnconfigure(0, weight=5)
    subframe.rowconfigure(0, weight=5)
    font_change = font.Font(family='Helvetica', size=20, weight='bold')
    ttk.Label(mainframe, text="SIMULATOR/
CALCULATOR",font=font_change,foreground='red').grid (column=0, row=0,
sticky=(N))
    ttk.Label(velocity_frame, text="Slide to change velocity (m/s)",
foreground='red').grid (column=3, row=0, sticky=(N))
    ttk.Label(angle_frame, text="Slide to change angle of elevation (degrees)",
foreground='red').grid (column=3, row=0, sticky=(N))
    ttk.Button(subframe, text="HOME", command=home).grid(column=0, row=11,
sticky=S)
    global velocity
    velocity = tk.Scale(velocity_frame, orient=HORIZONTAL, length=200, from_=0.0,
to=100.0)
    velocity.grid(column=0, row=0, sticky=N)
    global angle
```

```
    angle = tk.Scale(angle_frame, orient=HORIZONTAL, length=200, from_=0.0,
to=180.0)
    angle.grid(column=0, row=0, sticky=N)
    ttk.Button(subframe, text="GO", command=velocity_angle).grid(column=0,
row=11, sticky=S)
    ttk.Button(subframe, text="HOME", command=home).grid(column=1, row=11,
sticky=S)



def simulators(velocity,angle):

    cv = tk.Canvas(width=200,height=200)
    t = turtle.RawTurtle(cv)

    angle=radians(angle)

    wn=turtle.Screen()

    wn.bgcolor('#FFD700')
    wn.title("projectile")

    ball= turtle.Turtle()
    ball.shape('circle')
    ball.color('#DC143C')
    ball.penup()
    ball.speed(0)
    ball.goto(-330,-10)
    ball.pendown()
    ball.dy=0

    g=0.1
    time=0


    while ball.ycor() >= -10:
        time=time+0.05
        changeInX = (velocity*cos(angle)) * time
        changeInY = ((velocity*sin(angle)) * time) + (0.5*-9.81*time*time)
        ball.sety(-10+changeInY)
        ball.setx(-330+changeInX)
def run_Vt():
    Vt_graph(velocity1.get(),acceleration.get(),time1.get())
def Vt_graph(velocity,acceleration,time):
    import matplotlib.pyplot as plt
    x = []
    for i in range(0,time+1):
        x.append(i)
```

```
        i+=1

    # corresponding y axis values
    y = []
    for i in range(0,time+1):
##        print('at',i,'seconds')
        v=velocity+ acceleration*i

        y.append(v)
        i+=1

    # plotting the points
    plt.plot(x, y)

    # naming the x axis
    plt.xlabel('Time (s)')
    # naming the y axis
    plt.ylabel('velocity (m/s)')

    # giving a title to my graph
    plt.title('V/t graph')

    # function to show the plot
    plt.show()
def run_St():
    St_graph(velocity1.get(),acceleration.get(),time1.get())
def St_graph(velocity,acceleration,time):
    import matplotlib.pyplot as plt
    x = []
    for i in range(0,time+1):
        x.append(i)
        i+=1

    # corresponding y axis values
    y = []
    for i in range(0,time+1):
##        print('at',i,'seconds')
        s=velocity*i + (0.5*acceleration*i*i)

        y.append(s)
        i+=1

    # plotting the points
    plt.plot(x, y)

    # naming the x axis
    plt.xlabel('Time (s)')
    # naming the y axis
```

```
    plt.ylabel('displacement (m)')

    # giving a title to my graph
    plt.title('S/t graph')
    plt.show()
def graphs(*args):
    root.geometry("685x600")
#set the dimensions of the window
    font_change = font.Font(family='Helvetica', size=20, weight='bold')
    mainframe = ttk.Frame(root,padding= "100 200 100 200")
    displacement_frame = ttk.Frame(mainframe,padding= "50 50 50
50",relief='sunken')
    velocity_frame = ttk.Frame(mainframe,padding= "50 50 50 50",relief='sunken')
#padding adds extra space around the widget
#create the frame for the window
    mainframe.grid (column=0, row=0, sticky=(N, W, E, S))
    displacement_frame.grid (column=0, row=3, sticky=(N, W, E, S))
    velocity_frame.grid (column=1, row=3, sticky=(N, W, E, S))
#sticky means how the widget would line up within the grid cell
    mainframe.columnconfigure(0, weight=5)
    mainframe.rowconfigure(0, weight=5)
    ttk.Label(mainframe, text=" Enter velocity: ",font=font_change,
foreground='red').grid (column=0, row=0, sticky=(N))
    ttk.Label(mainframe, text=" Enter time: ",font=font_change, foreground='red').grid
(column=0, row=1, sticky=(N))
    ttk.Label(mainframe, text=" Enter acceleration: ",font=font_change,
foreground='red').grid (column=0, row=2, sticky=(N))
    global velocity1
    velocity1=IntVar()
    velocity1_entry = ttk.Entry(mainframe, width=10,
textvariable=velocity1).grid(column=1, row=0, sticky=(N))
    global time1
    time1=IntVar()
    time1_entry = ttk.Entry(mainframe, width=10, textvariable=time1).grid(column=1,
row=1, sticky=(N))
    global acceleration
    acceleration=IntVar()
    acceleration_entry = ttk.Entry(mainframe, width=10,
textvariable=acceleration).grid(column=1, row=2, sticky=(N))

    ttk.Button(mainframe, text="HOME", command=home).grid(column=1, row=11,
sticky=(N))
    ttk.Label(displacement_frame, text=" Select for displacement/time graph
",font=font_change, foreground='red').grid (column=1, row=0, sticky=(N))
    displacement_check= Checkbutton(displacement_frame, state=ACTIVE,
command=run_St).grid (column=1, row=1, sticky=(N))
    ttk.Label(velocity_frame, text=" Select for velocity/time graph ",font=font_change,
foreground='red').grid (column=1, row=0, sticky=(N))
```

```
    velocity_check= Checkbutton(velocity_frame, state=ACTIVE,
command=run_Vt).grid (column=1, row=1, sticky=(N))
def Test_q(*args):
    root.geometry("685x600")
#set the dimensions of the window

    mainframe = ttk.Frame(root,padding= "100 200 100 200")
    subframe = ttk.Frame(mainframe,padding= "100 50 100 50",relief='sunken')
#padding adds extra space around the widget
#create the frame for the window
    mainframe.grid (column=0, row=0, sticky=(N, W, E, S))
    subframe.grid (column=2, row=15, sticky=(N, W, E, S))

#sticky means how the widget would line up within the grid cell
    mainframe.columnconfigure(0, weight=5)
    mainframe.rowconfigure(0, weight=5)
    subframe.columnconfigure(0, weight=5)
    subframe.rowconfigure(0, weight=5)

    font_change = font.Font(family='Helvetica', size=20, weight='bold')
    ttk.Label(mainframe, text=" TEST ",font=font_change, foreground='red').grid
(column=2, row=1, sticky=(N))
    ttk.Label(mainframe, text="Please select a topic").grid (column=2, row=5,
sticky=(N))
    ttk.Button(subframe, text="FLAT PLANE", command=Test_q_flat).grid(column=1,
row=10, sticky=S)
    ttk.Button(subframe, text="FREE FALL", command=Test_q_free).grid(column=2,
row=10, sticky=S)

    ttk.Button(subframe, text="PROJECTILES",
command=Test_q_projectile).grid(column=3, row=10, sticky=S)
    ttk.Button(subframe, text="HOME", command=home).grid(column=2, row=11,
sticky=S)


#########################################################################
##############################################
def results():
    value= "simulator15.db"
    conn = sqlite3.connect(value)
    c=conn.cursor()

    root.geometry("685x600")
#set the dimensions of the window

    mainframe = ttk.Frame(root,padding= "100 200 100 200")
#padding adds extra space around the widget
#create the frame for the window
```

```
    mainframe.grid (column=0, row=0, sticky=(N, W, E, S))
#sticky means how the widget would line up within the grid cell
    mainframe.columnconfigure(0, weight=1)
    mainframe.rowconfigure(0, weight=1)
##    ttk.Button(mainframe, text="HOME", command=home).grid(column=2, row=11,
sticky=S)
    count=0
    name_u=name.get().upper()
    font_change = font.Font(family='Helvetica', size=20, weight='bold')
    ttk.Label(mainframe, text="HERE ARE YOUR RESULTS
"+name_u,font=font_change,foreground='red').grid (column=0, row=0, sticky=(N))
    ttk.Label(mainframe, text="QUESTION
1:",font=font_change,foreground='red').grid (column=0, row=1, sticky=(N))
    ttk.Label(mainframe, text="QUESTION
2:",font=font_change,foreground='red').grid (column=0, row=2, sticky=(N))
    ttk.Label(mainframe, text="QUESTION
3:",font=font_change,foreground='red').grid (column=0, row=3, sticky=(N))
    ttk.Label(mainframe, text="QUESTION
4:",font=font_change,foreground='red').grid (column=0, row=4, sticky=(N))
    ttk.Label(mainframe, text="QUESTION
5:",font=font_change,foreground='red').grid (column=0, row=5, sticky=(N))
    ttk.Button(mainframe, text="Back", command=Test_q_flat).grid(column=2, row=7,
sticky=S)
    ttk.Button(mainframe, text="Home", command=home).grid(column=1, row=7,
sticky=S)
    for row in c.execute('SELECT question_num, question, answer FROM Questions
WHERE question_type==1'):
        if a_1.get() == row[2]:
            c.execute('UPDATE results SET Q1=1 WHERE resultID=2')
            ttk.Label(mainframe, text='correct',style='Mycolour.TLabel').grid (column=2,
row=1, sticky=(N))
            count+=1
        if a_2.get() == row[2]:
            c.execute('UPDATE results SET Q2=1 WHERE resultID=2')
            ttk.Label(mainframe, text='correct',style='Mycolour.TLabel').grid (column=2,
row=2, sticky=(N))
            count+=1
        if a_3.get() == row[2]:
            c.execute('UPDATE results SET Q3=1 WHERE resultID=2')

            ttk.Label(mainframe, text='correct',style='Mycolour.TLabel').grid (column=2,
row=3, sticky=(N))
            count+=1
        if a_4.get() == row[2]:
            c.execute('UPDATE results SET Q4=1 WHERE resultID=2')
            ttk.Label(mainframe, text='correct',style='Mycolour.TLabel').grid (column=2,
row=4, sticky=(N))
            count+=1
```

```
        if a_5.get() == row[2]:
            c.execute('UPDATE results SET Q5=1 WHERE resultID=2')
            ttk.Label(mainframe, text='correct',style='Mycolour.TLabel').grid (column=2,
row=5, sticky=(N))
            count+=1

    ttk.Label(mainframe, text=count,font=font_change,foreground='green').grid
(column=2, row=6, sticky=(N))
    print('hello')
    for row in c.execute('SELECT * FROM results'):
        print(row)
    print('hello')
    print('hello')

def Test_q_flat(*args):
    root.geometry("685x600")
#set the dimensions of the window

    mainframe = ttk.Frame(root,padding= "100 200 100 200")
#padding adds extra space around the widget
#create the frame for the window
    mainframe.grid (column=0, row=0, sticky=(N, W, E, S))
#sticky means how the widget would line up within the grid cell
    mainframe.columnconfigure(0, weight=1)
    mainframe.rowconfigure(0, weight=1)
##    ttk.Button(mainframe, text="HOME", command=home).grid(column=2, row=11,
sticky=S)
    font_change = font.Font(family='Helvetica', size=20, weight='bold')
    ttk.Label(mainframe, text="PARTICLES ON A FLAT
PLANE",font=font_change,foreground='red').grid (column=0, row=0, sticky=(N))
    import sqlite3
    value= "simulator15.db"
    conn = sqlite3.connect(value)
    c=conn.cursor()

    for row in c.execute('SELECT question_num, question, answer FROM Questions
WHERE question_type==1'):
        question=row[1]
        a=row[0]
        i=row[0]
        i=ttk.Label(mainframe, text=question,style='Mycolour.TLabel').grid (column=0,
row=i*2, sticky=(N))
    global a_1
    a_1=StringVar()
    a_1_entry = ttk.Entry(mainframe, width=30, textvariable=a_1)
    a_1_entry.grid(column=0, row=3, sticky=(N))
    global a_2
    a_2=StringVar()
```

```
    a_2_entry = ttk.Entry(mainframe, width=30, textvariable=a_2)
    a_2_entry.grid(column=0, row=5, sticky=(N))
    global a_3
    a_3=StringVar()
    a_3_entry = ttk.Entry(mainframe, width=30, textvariable=a_3)
    a_3_entry.grid(column=0, row=7, sticky=(N))
    global a_4
    a_4=StringVar()
    a_4_entry = ttk.Entry(mainframe, width=30, textvariable=a_4)
    a_4_entry.grid(column=0, row=9, sticky=(N))
    global a_5
    a_5=StringVar()
    a_5_entry = ttk.Entry(mainframe, width=30, textvariable=a_5)
    a_5_entry.grid(column=0, row=11, sticky=(N))

    ttk.Button(mainframe, text="ENTER", command=results).grid(column=2, row=11,
sticky=S)
##########################################################################
#########################################################


def results_free():
    value= "simulator15.db"
    conn = sqlite3.connect(value)
    c=conn.cursor()

    root.geometry("685x600")
#set the dimensions of the window

    mainframe = ttk.Frame(root,padding= "100 200 100 200")
#padding adds extra space around the widget
#create the frame for the window
    mainframe.grid (column=0, row=0, sticky=(N, W, E, S))
#sticky means how the widget would line up within the grid cell
    mainframe.columnconfigure(0, weight=1)
    mainframe.rowconfigure(0, weight=1)
##    ttk.Button(mainframe, text="HOME", command=home).grid(column=2, row=11,
sticky=S)
    count=0
    name_u=name.get().upper()
    font_change = font.Font(family='Helvetica', size=20, weight='bold')
    ttk.Label(mainframe, text="HERE ARE YOUR RESULTS
"+name_u,font=font_change,foreground='red').grid (column=0, row=0, sticky=(N))
    ttk.Label(mainframe, text="QUESTION
1:",font=font_change,foreground='red').grid (column=0, row=1, sticky=(N))
    ttk.Label(mainframe, text="QUESTION
2:",font=font_change,foreground='red').grid (column=0, row=2, sticky=(N))
```

```
    ttk.Label(mainframe, text="QUESTION
3:",font=font_change,foreground='red').grid (column=0, row=3, sticky=(N))
    ttk.Label(mainframe, text="QUESTION
4:",font=font_change,foreground='red').grid (column=0, row=4, sticky=(N))
    ttk.Label(mainframe, text="QUESTION
5:",font=font_change,foreground='red').grid (column=0, row=5, sticky=(N))
    ttk.Button(mainframe, text="Back", command=Test_q_free).grid(column=2,
row=7, sticky=S)
    ttk.Button(mainframe, text="Home", command=home).grid(column=1, row=7,
sticky=S)

    for row in c.execute('SELECT question_num, question, answer FROM Questions
WHERE question_type==2'):
        if a_1.get() == row[2]:
            c.execute('UPDATE results SET Q1=1 WHERE resultID=2')
            ttk.Label(mainframe, text='correct',style='Mycolour.TLabel').grid (column=2,
row=1, sticky=(N))
            count+=1
        if a_2.get() == row[2]:
            c.execute('UPDATE results SET Q2=1 WHERE resultID=2')
            ttk.Label(mainframe, text='correct',style='Mycolour.TLabel').grid (column=2,
row=2, sticky=(N))
            count+=1
        if a_3.get() == row[2]:
            c.execute('UPDATE results SET Q3=1 WHERE resultID=2')

            ttk.Label(mainframe, text='correct',style='Mycolour.TLabel').grid (column=2,
row=3, sticky=(N))
            count+=1
        if a_4.get() == row[2]:
            c.execute('UPDATE results SET Q4=1 WHERE resultID=2')
            ttk.Label(mainframe, text='correct',style='Mycolour.TLabel').grid (column=2,
row=4, sticky=(N))
            count+=1
        if a_5.get() == row[2]:
            c.execute('UPDATE results SET Q5=1 WHERE resultID=2')
            ttk.Label(mainframe, text='correct',style='Mycolour.TLabel').grid (column=2,
row=5, sticky=(N))
            count+=1
    ttk.Label(mainframe, text=count,font=font_change,foreground='green').grid
(column=2, row=6, sticky=(N))
    print('hello')
    for row in c.execute('SELECT * FROM results'):
        print(row)
    print('hello')


def Test_q_free(*args):
```

```
    root.geometry("685x600")
#set the dimensions of the window

    mainframe = ttk.Frame(root,padding= "100 200 100 200")
#padding adds extra space around the widget
#create the frame for the window
    mainframe.grid (column=0, row=0, sticky=(N, W, E, S))
#sticky means how the widget would line up within the grid cell
    mainframe.columnconfigure(0, weight=5)
    mainframe.rowconfigure(0, weight=5)
    ttk.Button(mainframe, text="HOME", command=home).grid(column=2, row=11,
sticky=S)
    root.geometry("685x600")
#set the dimensions of the window

    mainframe = ttk.Frame(root,padding= "100 200 100 200")
#padding adds extra space around the widget
#create the frame for the window
    mainframe.grid (column=0, row=0, sticky=(N, W, E, S))
#sticky means how the widget would line up within the grid cell
    mainframe.columnconfigure(0, weight=1)
    mainframe.rowconfigure(0, weight=1)
    font_change = font.Font(family='Helvetica', size=20, weight='bold')
    ttk.Label(mainframe, text="PARTICLES IN FREE
FALL",font=font_change,foreground='red').grid (column=0, row=0, sticky=(N))
    import sqlite3
    value= "simulator15.db"
    conn = sqlite3.connect(value)
    c=conn.cursor()

    for row in c.execute('SELECT question_num, question, answer FROM Questions
WHERE question_type==2'):
        question=row[1]
        a=row[0]
        i=row[0]
        i=ttk.Label(mainframe, text=question,style='Mycolour.TLabel').grid (column=0,
row=i*2, sticky=(N))
    global a_1
    a_1=StringVar()
    a_1_entry = ttk.Entry(mainframe, width=30, textvariable=a_1)
    a_1_entry.grid(column=0, row=3, sticky=(N))
    global a_2
    a_2=StringVar()
    a_2_entry = ttk.Entry(mainframe, width=30, textvariable=a_2)
    a_2_entry.grid(column=0, row=5, sticky=(N))
    global a_3
    a_3=StringVar()
```

```
    a_3_entry = ttk.Entry(mainframe, width=30, textvariable=a_3)
    a_3_entry.grid(column=0, row=7, sticky=(N))
    global a_4
    a_4=StringVar()
    a_4_entry = ttk.Entry(mainframe, width=30, textvariable=a_4)
    a_4_entry.grid(column=0, row=9, sticky=(N))
    global a_5
    a_5=StringVar()
    a_5_entry = ttk.Entry(mainframe, width=30, textvariable=a_5)
    a_5_entry.grid(column=0, row=11, sticky=(N))

    ttk.Button(mainframe, text="ENTER", command=results_free).grid(column=2,
row=11, sticky=S)

#########################################################################
#########################################################################




def results_projectiles():
    value= "simulator15.db"
    conn = sqlite3.connect(value)
    c=conn.cursor()

    root.geometry("685x600")
#set the dimensions of the window

    mainframe = ttk.Frame(root,padding= "100 200 100 200")
#padding adds extra space around the widget
#create the frame for the window
    mainframe.grid (column=0, row=0, sticky=(N, W, E, S))
#sticky means how the widget would line up within the grid cell
    mainframe.columnconfigure(0, weight=1)
    mainframe.rowconfigure(0, weight=1)
##    ttk.Button(mainframe, text="HOME", command=home).grid(column=2, row=11,
sticky=S)
    count=0
##set count variable to 0
    name_upper=name.get().upper()
##retrieve the value of name and make it upper case
    font_change = font.Font(family='Helvetica', size=20, weight='bold')
##formatting for font change
    ttk.Label(mainframe, text="HERE ARE YOUR RESULTS
"+name_upper,font=font_change,foreground='red').grid (column=0, row=0,
sticky=(N))
    ttk.Label(mainframe, text="QUESTION
1:",font=font_change,foreground='red').grid (column=0, row=1, sticky=(N))
```

```
   ttk.Label(mainframe, text="QUESTION
2:",font=font_change,foreground='red').grid (column=0, row=2, sticky=(N))
   ttk.Label(mainframe, text="QUESTION
3:",font=font_change,foreground='red').grid (column=0, row=3, sticky=(N))
   ttk.Label(mainframe, text="QUESTION
4:",font=font_change,foreground='red').grid (column=0, row=4, sticky=(N))
   ttk.Label(mainframe, text="QUESTION
5:",font=font_change,foreground='red').grid (column=0, row=5, sticky=(N))
   ttk.Button(mainframe, text="Back", command=Test_q_projectile).grid(column=2,
row=7, sticky=S)
   ttk.Button(mainframe, text="Home", command=home).grid(column=1, row=7,
sticky=S)

   for row in c.execute('SELECT question_num, question, answer FROM Questions
WHERE question_type==3'):
## sql statement to query database
      if a_1.get() == row[2]:
##retrieves value in variable and compares if equal to value in the above position in
the row
         c.execute('UPDATE results SET Q1=1 WHERE resultID=2')
##changes value in database
         ttk.Label(mainframe, text='correct',style='Mycolour.TLabel').grid (column=2,
row=1, sticky=(N))
         count+=1
##adds 1 to count variable
      if a_2.get() == row[2]:
         c.execute('UPDATE results SET Q2=1 WHERE resultID=2')
         ttk.Label(mainframe, text='correct',style='Mycolour.TLabel').grid (column=2,
row=2, sticky=(N))
         count+=1
      if a_3.get() == row[2]:
         c.execute('UPDATE results SET Q3=1 WHERE resultID=2')

         ttk.Label(mainframe, text='correct',style='Mycolour.TLabel').grid (column=2,
row=3, sticky=(N))
         count+=1
      if a_4.get() == row[2]:
         c.execute('UPDATE results SET Q4=1 WHERE resultID=2')
         ttk.Label(mainframe, text='correct',style='Mycolour.TLabel').grid (column=2,
row=4, sticky=(N))
         count+=1
      if a_5.get() == row[2]:
         c.execute('UPDATE results SET Q5=1 WHERE resultID=2')
         ttk.Label(mainframe, text='correct',style='Mycolour.TLabel').grid (column=2,
row=5, sticky=(N))
         count+=1
   ttk.Label(mainframe, text=count,font=font_change,foreground='green').grid
(column=2, row=6, sticky=(N))
```

```
##displays value currently in count

    conn.commit()
##saves changes to database

def Test_q_projectile(*args):
    root.geometry("685x600")
#set the dimensions of the window

    mainframe = ttk.Frame(root,padding= "10 200 10 200")
#padding adds extra space around the widget
#create the frame for the window
    mainframe.grid (column=0, row=0, sticky=(N, W, E, S))
#sticky means how the widget would line up within the grid cell
    mainframe.columnconfigure(0, weight=5)
    mainframe.rowconfigure(0, weight=5)
    font_change = font.Font(family='Helvetica', size=20, weight='bold')
    ttk.Label(mainframe, text="PARTICLES PROJECTED AT AN
ANGLE",font=font_change,foreground='red').grid (column=0, row=0, sticky=(N))
    import sqlite3
##imports library for using sql
    value= "simulator15.db"
    conn = sqlite3.connect(value)
##connects to database
    c=conn.cursor()
##creates cursor
    for row in c.execute('SELECT question_num, question, answer FROM Questions
WHERE question_type==3'):
##sql statement for querying database with a condition
        question=row[1]
        a=row[0]
        i=row[0]
##stores the contents of the item in the given position of the row in the above
variables
        i=ttk.Label(mainframe, text=question,style='Mycolour.TLabel').grid (column=0,
row=i*2, sticky=(N))
##code for creating label is stored in variable above
    global a_1
##declares variable as global to later access outside function
    a_1=StringVar()
##sets the data type for the variable
    a_1_entry = ttk.Entry(mainframe, textvariable=a_1)
    a_1_entry.grid(column=0, row=3, sticky=(N))
##creates a widget to enter answer and stores it in variable
    global a_2
    a_2=StringVar()
    a_2_entry = ttk.Entry(mainframe, textvariable=a_2)
    a_2_entry.grid(column=0, row=5, sticky=(N))
```

```
    global a_3
    a_3=StringVar()
    a_3_entry = ttk.Entry(mainframe, textvariable=a_3)
    a_3_entry.grid(column=0, row=7, sticky=(N))
    global a_4
    a_4=StringVar()
    a_4_entry = ttk.Entry(mainframe, textvariable=a_4)
    a_4_entry.grid(column=0, row=9, sticky=(N))
    global a_5
    a_5=StringVar()
    a_5_entry = ttk.Entry(mainframe, textvariable=a_5)
    a_5_entry.grid(column=0, row=11, sticky=(N))
    ttk.Button(mainframe, text="ENTER",
command=results_projectiles).grid(column=2, row=11, sticky=S)




##########################################################################
########################################################################

def connect(*args):
    value= "simulator15.db"
    conn = sqlite3.connect(value)
    c=conn.cursor()
    user=name.get()
    record = [(2,str(user),0,0,0,0,0,0)]
    c.executemany('INSERT INTO results VALUES (?,?,?,?,?,?,?,?)', record)
##    additional_window = tk.Toplevel()   #**********HERE*******
##    additional_window.title("welcome")
#function to connect databse to the programme
    for row in c.execute('SELECT * FROM results'):
        print(row)
    conn.commit()




def home(*args):


    #set the dimensions of the window
    s = ttk.Style()
    s.configure('Mycolour.TFrame',background='red')
    s.configure('Mycolour.TButton',background='red',foreground='red')
```

```
    s.configure('Mycolour.TLabel',background='red',foreground='red')

    mainframe = ttk.Frame(root,padding= "100 200 100
200",relief='sunken',style='Mycolour.TFrame')
    subframe = ttk.Frame(mainframe,padding= "100 50 100
50",relief='sunken',style='Mycolour.TFrame')
    ##mainframe.configure(background='red')
    #padding adds extra space around the widget
    #create the frame for the window
    mainframe.grid (column=0, row=0, sticky=(N, W, E, S))
    subframe.grid (column=2, row=15, sticky=(N, W, E, S))
    #sticky means how the widget would line up within the grid cell
    mainframe.columnconfigure(0, weight=1)
    mainframe.rowconfigure(0, weight=1)
    subframe.columnconfigure(0, weight=5)
    subframe.rowconfigure(0, weight=5)

    #column and row configure tells Tk that if the main window is resized, the frame
should expand to take up extra space
    global name
    name=StringVar()
    font_change = font.Font(family='Helvetica', size=20, weight='bold')
    ttk.Label(mainframe, text="MECHANICS IN
MATHS",font=font_change,foreground='red').grid (column=2, row=0, sticky=(N))
    ttk.Label(mainframe, text="Please enter a
username",style='Mycolour.TLabel').grid (column=2, row=5, sticky=(N))
    name_entry = ttk.Entry(mainframe, width=30, textvariable=name)
    name_entry.grid(column=2, row=10, sticky=(N))
    ttk.Button(mainframe, text="ENTER", command=connect,
style='Mycolour.TButton').grid(column=2, row=10, sticky=E)
    ttk.Button(subframe, text="TEST", command=Test_q,
style='Mycolour.TButton').grid(column=1, row=10, sticky=S)
    ttk.Button(subframe, text="GRAPHS",
command=graphs,style='Mycolour.TButton').grid(column=2, row=10, sticky=S)

    ttk.Button(subframe, text="SIMULATOR",
command=simulator,style='Mycolour.TButton').grid(column=3, row=10, sticky=S)

    ##frame = ttk.Frame(subframe,borderwidth=10,
relief='sunken').grid(column=2,row=15, sticky=(S))
root = tk.Tk()
root.title("Mechanics simulator")

root.geometry("685x600")



home()
```

Evaluation:
Now the solution is completed well begin the post development testing phase where we will test for robustness as well as functionality of the solution as a whole, then this will be cross referenced against the success criteria to evaluate if the criteria were met fully, partially or not at all. Finally we'll also consider the future maintenance or improvements that could be added.

Post development testing:
To carry this out I will be using black box testing and I will being testing for robustness, evaluating all possible pathways. To verify the tests I will be checking to see if the expected outcome is met. For each section of the iterative testing there will be a separate test and for each successful test there will be a point awarded and if a certain percentage is met by the end, then it will be deemed sufficient.

Section 1: Home screen

Test	Data Input	Expected outcome	Outcome as expected?	Points
A name is entered and the enter button is clicked	'John'	The name 'john' is added to the database	Yes.	1
The 'test' button is pressed	n/a	The screen for the test section of the program is displayed and the user can now access these functions	Yes.	1
The 'graphs' button is pressed	n/a	The screen for the graphs section of the program is displayed and the user can now access these functions	Yes.	1
The 'simulator' button is pressed	n/a	The screen for the simulator section of the program is displayed and the user can now access these functions	yes.	1

Section 2: Test screen

Test	Data Input	Expected outcome	Outcome as expected?	Points
The flat plane button is pressed	n/a			

Test	Data Input	Expected outcome	Outcome as expected?	Points
User can input answers under question or can be left blank	Q1) 23.4 Q2) 35 Q3) abc Q4) 4 Q5)	The first question is correct, the second is a correct answer in the database but for a different question so should be incorrect. the second is wrong but also a different data type to test if it will crash. Question 4 is simply wrong and the last question is blank so should be automatically incorrect	Yes. the results displayed matched those expected. (1 0 0 0 0)	1
The enter button is pressed	n/a	The results are displayed. The answers inputted are checked against the ones stored in the database for that particular question. Points are awarded where correct.	yes. As the above test showed.	1
the back button is pressed	n/a	Returns user to previous page. The input boxes should be empty and scores reset so user can try again	yes.	1
The home button is pressed	n/a	the home screen is now displayed	Yes.	1
The free fall button is pressed	n/a			
User can input answers under question or can be left blank	Q1)8 Q2)35 Q3)abc Q4)4 Q5)	The first question is correct, the second is a correct answer in the database but for a different question so should be incorrect. the second is wrong but also a different data type to test if it will crash. Question 4 is simply wrong and the last question is blank so should be automatically incorrect	Yes	1

Test	Data Input	Expected outcome	Outcome as expected?	Points
The enter button is pressed	n/a			
the back button is pressed	n/a			
The home button is pressed	n/a			
The projectiles button is pressed				
User can input answers under question or can be left blank	Q1) Q2) Q3) Q4) Q5)			
The enter button is pressed				
the back button is pressed				
The home button is pressed				

Section 3: Graphs screen

Test	Data Input	Expected outcome	Outcome as expected?	Points
Integers are entered for three values and 'velocity/ time' check box is clicked	Velocity:50 Time:30 Acceleration:5	A new screen should pop up with the graph correctly displayed. Each point on graph should be calculated using the values entered. Should be a straight line graph.	Yes.	1
Integers are entered for three values and 'displacement/ time' check box is clicked	Velocity:50 Time:30 Acceleration:5	A new screen should pop up with the graph correctly displayed. Each point on graph should be calculated using the values entered. Should be a straight curved graph.	Yes.	1

Test	Data Input	Expected outcome	Outcome as expected?	Points
A letter is inputted instead of an integer	Velocity:A Time:30 Acceleration:5	An invalid data type was entered so should not open new graph window	Yes.	1
The home button is pressed	n/a			

Section 4: Simulator screen

Test	Data Input	Expected outcome	Outcome as expected?	Points
The velocity and angle are set to the lowest possible	Velocity:0 Angle:0	As the algorithm follows the laws of mechanics the ball should remain stationary.	Yes.	1
The velocity and angle are set to the maximum value	Velocity:100 Angle:180	Here the ball should move the exact opposite way to which its originally going however does not come off the floor of the plane so shouldn't move either as it doesn't leave the plane.	Yes.	1
the velocity is set to a random value in between maximum and minimum	Velocity:60 Angle:45	Should expect the simulation to be a curve witch is symmetrical, when ball comes back down and reach the floor, should stop immediately	yes.	1
Home button is pressed				

Usability test:
The following questionnaire was given to a user after using the solution to pass this test, the solution must score at least 70%. A rating out of 10 is given for each and an average out of 10 is calculated where the percentage will be taken from.

Question	Score /10
How simple was the colour scheme and how much did this help to make the solution simpler?	6/10
How simple and easy to interpret were the different pages, in terms of the layout of widgets?	8/10
How useful and easy to use are the widgets in the solution?	9/10
How noticeable are the headings of each page?	7/10
How easy was it to navigate from one page to another or back to the home page?	9/10
TOTAL (AVERAGE SCORE):	7.8/10 (**78%**)

As the table above shows, the user on average has rated the usability of the solution a 7.8/10 whlichmeets the requirement that was set before of 70%, theretore the solution has passed the usability test and this success criteria has officially been met.

To conclude the post development testing, the success percentage was 100% which is sufficient in that there are no obvious errors in the solution that would prevent the user from using it properly. As a result, the development has been successfully completed.

Evidence for testing:

Success criteria evaluation:

Can connect to a database and store a users results and can be manipulated by the program to display results:
This requirement was essential as it was a way of personalising the solution each time it was used and for a learning too this is very important. As a result, the user must be able to add a name which will then be stored in a database along with their scores from the test. As it is a learning to, reflecting on progress is extremely important and so the program is then required to show this result and give them a total. Looking at the evidence above, when a name is inputted it is clearly added to a database and later on the results page you can see what there score is but it also has a name at the top to signify who the result belongs to and this score is also shown to be stored in the database as where there were ‘0’ in the dates for that user, if correct, there is now a ‘1’. This criteria was partially met as although it does store a user then display and store results however the solution is not capable of showing more than one result (for when user has taken the test multiple times) and also once the result is displayed, it cannot be displayed again. It is still stored in the database but the solution has no feature that would allow it to be shown again so for future improvements this is something that can be added as to the user the current solution isn’t very useful although it does provide the basic information and therefor is partially met.

The solution provides a testing feature for the three topics where the user can attempt questions:
As this is a learning tool, a main part of that is being able to test yourself or you cant view your progress. In the evidence you can see that there are three options in the GUI to chose from in after you choose test and each one will contain different questions corresponding to the topic. The user is able to input their answers and check them. This criteria was fully met as all three topics are covered and have test questions. They also all have an input box allowing the user to actually answer the questions and then check the answer. They can attempt the questions again by simply pressing 'back'. This allows the user to see their progress but also try again and see how their progress develops meaning the criteria is thoroughly met.

The two key graphs in the specification can be plotted by the program when user enters data:
A key area of the syllabus is the distance/time and velocity/time graphs and being able to interpret them is essential. The solution I've created only partially meets the requirements. The solution currently allows you to enter values and select the graph you desire. Then it uses the matplot library to instantly plot the graph. The user can zoom in and out of the graph and also save the graphs for later reference which is useful however it hasn't completely met the criteria since there isn't any guidance or information on how to interpret the graph it simply does it for you therefore this is useful to a limit. For this reason the requirement is only partially met as the user may struggle to use it to improve understanding of the graphs unless they already have a foundation of knowledge and can use it according to what they are specifically looking for, not requiring any basic guidance.

The solution has a simulator which can model projectile motion and use the SUVAT equations to calculate this each time so the model is accurate and proportional to the results expected:
This was the most intricate section and is the focal point of the solution. It aims to tack the issue of visualising a very hard concept of projectile motion. Being able to do this can help to understand and therefor answer the questions on it. The solution currently allows you to input values and using the relevant equations, it calculates the new geographical position and moves the 'particle' to that point. This increments every second and so all the small movements add together to create a simulation of what would happen. This can be seen in the evidence and in the testing that the simulator does run correctly. This criteria has been fully met as it does have a working simulation and uses the correct equations.

Allowed to change a value at any time in the simulator and receive a new output:
The user must be able to test many different scenarios so constantly re entering new values and running again. This requirement was fully met as shown above in the testing as when the program ran and values were re-entered multiple times where you can see multiple lines drawn that represent each time the input was changed and inputted. The fact that each line drawn was different shows that the output was a new one for that particular new input and not just the last one repeated so this criteria was fully met

Usability testing:

The solution is essentially aimed at older kids who are generally very good at using technology, nevertheless it can also be used by those aiming to teach the subject who are much older and being a tool aimed at teaching, a simple and easy to use solution would help in keeping the focus on learning than trying to navigate and utilise the programme properly. As mentioned from the design phase, the layout is very simple. The widgets are large and easy to notice, the use of bold letters helps to create a contrast and make certain words stand out to the user and make it easier to navigate. For example, titles are in bold so it is immediately obvious what the page is about and where they are in the solution so they can navigate accordingly. Also the use of red outlining is particularly useful as it supported out certain sections of the page which can be seen on multiple pages. This segregation makes the layout more organised and so easier for the user to read. Besides layout, the use of certain wingers such as the scales widget make the solution particularly user friendly. The scale widget is used for the simulator and it simplifies the users input as the simply slide between the set values and if they want to slight increase or decrease they can just slide left or right. This is appeased to having to enter values themselves each time and also any numbers bigger than 100 for velocity would have seen the simulation not fit on the page so instead of telling the user not to enter values higher, the scale widget made the problem

simpler by making it impossible to enter anything more so the user doesn't have to worry about maximum values.

Overall all of these feature combined to give the solution an overall rating of 7.8/10 by the user and this was sufficient meaning the usability of the solution is sufficient and the user had no problem and needed little help in navigating and using the solution which means it was successful.

However one thing I would consider in later development would be the addition of keyboard commands that would allow for more than one way of input. The variety makes it more user friendly as the user isn't restricted to one form of input. Plus, if the mouse or pad were to not be working there would be an alternative for of input (e.g. pressing enter button) whereas if that happened now the solution will be redundant as there would be no way of using it or navigating the solution.

Limitations:
Due to the time constraints I wasn't able to include more stakeholders and more user tests so as a result the feedback was limited. This limited the improvements I could make. Also it would have been more useful to have added a more complex database and added more information that was displayed to the user about results however du e to time constraints again, this was not possible. Another limitation was in the calculator, Ideally there should have been more functions to it like being able to enter values and the solution find the correct equation and solve it however this was very complex and would've taken a lot more time therefore within the time span I had I chose to use this time to add a variety of other functions instead like the graphing and test questions.

Maintenance:
To allow for easier maintenance in the future, the code is fully commented to help whoever is looking at it understand it and follow it. This would save time and make it easier to find where to implement the changes as there is a clear understanding of what is happening at each point. I also attempted to keep it very modular and use smaller functions where possible to allow for a more decomposed solution. This is useful s where there isa lot of different things happening in one function it can get confusing and hard to follow, as a result it would make it harder to troubleshoot any issues and locate them or find where to implement a solution. However the use of very small functions keeps it very organise and so simple to locate any problems as it is much clearer.

On the other hand, my solution was not written in object orientated form, as a result my code isn't as efficient and is potentially longer than it needs to be which creates a problem when maintaining as changes have to be made multiple times rather than once and it can be time consuming in finding where all these places are. For example, each page follows the same basic layout and instead of using the idea of polymorphism to create a super class for all these pages, I had to repeat the code. Now when making a change to this layout it must be done for all pages however had we coded it using polymorphism it would only have to be changed in one place.

Final comments:

Overall I am very please with how this project has gone. There were limitations to what I could do due to time and also ability, particularly involving using the equations and all the calculations. That being said, the project did end up meeting the criteria that I set out to achieve and it did so in a way which was very user friendly whilst also being completed in good time. There were a few issues in the development phase when coding where I got stuck and wasted a bit of time in debugging however these were all rectified and overall the timing of the whole project wasn't disturbed greatly. So, to conclude I believe this project was a such however with extra time there are improvements to be made that could certainly benefit the user.