STUDENT ORGANISER

Chester Lloyd - 4755

St George's Academy
Computer Science

Table of Contents

STUDENT ORGANISER

3.1.1 - Problem Identification:

Homework can be forgotten or not recorded which causes an inconvenience to the teacher and student. This results in the student being behind in their work and a delay in the teacher's marking. Homework may be overlooked if there are multiple tasks in a small space on a page or if the student simply forgets to record it.

Schools spend a lot of money when printing planners every year. Each time is a one-off print job that will only last the school for a single year before another set is required. A digital solution would be preferable if used correctly. However, issues could arise with a software solution, especially a mobile application. Some students could go off task when recording homework on their mobile devices and partake in inappropriate activities. A desktop application would solve this issue but not all classrooms or students will have a computer available. As well as this, it may not be as convenient to use a computer to view homework because a more portable method may be desirable.

Throughout the development of this solution, I will be using multiple elements of computational thinking.

I will use **logical thinking** when designing the program structure. It is vital for the planning process that I thoroughly detail the logical flow of the program and map every **branch**, **decision** and stage onto a flowchart. As a result, I can clearly see how the program will work and a success criteria can then be drafted in order to test the solution at a number of stages.

To ensure that this solution will be complete and functionally sound, I will deploy the **thinking ahead** computational technique when approaching this task. To start, I will plan the features that should be included. I will then create any logical process or algorithm templates. Whilst doing so, any inputs or outputs that occur during these algorithms should be determined. I will record any information for widgets or other solutions that I could refer to. Once, I have all of the data necessary to complete this task, I will construct the success criteria and begin the development process.

As well as just planning the program, I can use Python's libraries to import additional resources that would be very useful. Libraries contain code that can be called within a program to offer additional functionality. For example, I could use the package, 'tkinter' to create a graphical user interface so that the program is easier to use. I could use a package for connecting to and interacting with a database, such as 'sqlite3.' These modules are very useful and I can continuously use their functions to reduce the amount of code needed and improve the efficiency of the overall solution.

It could be possible that some parts of this program could be created using the same code. Connecting to a database and adding in tasks or simply reading them and sorting are two examples of how I could use this element of **computational thinking** within my solution. By using code in this manner, I can create code that is more efficient and will not repeat any similar or identical code.

Feature	How it is solvable by computational methods	Why it is amenable to computational approach
Store tasks	A database could be used to store the data for tasks. A front-end GUI would present this data and allow data to be inserted and modified.	Using a database for this will keep an organised structure of tasks. Each record can be accessed individually to output them to the user. This feature is amenable by a computational approach as a digital planner has many benefits over the traditional solution. They are safer as data can be backed up by simply copying the database file. In addition to this, planners are prone to damage. If a page got wet, any ink or writing on that page would be ruined and therefore the data is lost.
Sort tasks	Again, a database could store each task with a unique identifier. This can be used when sorting as each record ID unique.	A database requires that any dat stored in a table mut have an ID in order for the data contained to be sorted or searched. It is possible to sort tasks in a normal planner however it would be very time consuming to wite out these tasks again.
Access learning resources	Add in pages to the program with the resource's information. Pages can be easy to access and there would be, in theory, no limit to the number of pages created.	Each page added will not incur any additional charges as a paper based planner would. This will reduce costs and data pages can still be added at any time, modified or updated. The school could create as many resources that they wanted to add to this planner as pages can be easily created and links would be established in the menu bar to these pages. The main benefit is that any resource could be modified at any time without the need to re-print any pages. It would be very difficult to add pages to each planner once they have been printed and bineded together.

3.1.2 - Stakeholders:

People who would be interested in this solution would be any students who find the use of digital organization useful and aid their learning, or students that are revising for upcoming exams and want to clearly see what tasks to focus upon. I have opened this up for any students however; it would be more suitable for Key Stage Four, Key Stage Five or university students. This is because these students would have better access to computers in which this program can be used. This solution would be appropriate for a student’s needs as it will offer a number of sorting algorithms so that they can quickly view the tasks that require attention first. It will help them focus on the most important tasks available by changing their colours based on their due dates and potentially removing the overdue tasks that are no longer required. The program will aid their organisation for school as tasks will be stored in a logical, structured layout where modifications can be made if any data requires altering.

A planner is a tool that teachers use when communicating with student’s parents and when checking that the student is using it appropriately. Therefore, teachers would be another stakeholder in this project. By including a notes section into the program, the teacher could leave comments that could be accessible by the student and the parents.

Parents of the students would have an interest in this program too. They can read comments from tutors and teachers and ensure that they are using the organizer appropriately, recording homework and checking that they are up to date on all their tasks. As a future development, I could allow the parents to log in and view the student’s data including their timetable, calendar, tasks and any notes written.

Another stakeholder in this project would be the school. The cost of printing a custom set of planners every year is very high. By using software as a replacement, this can become much cheaper and more environmentally friendly. It will also be more convenient as a few dates or pages can be added in that can be seen by every user instantly rather than printing new planners. As well as this, I plan to allow the school to add in data pages where school resources can be found, for example: a school map, contact information, code of conduct, etc. For a future development of this project, the school could monitor the student’s use of the planner by using a log in. In addition, I could include a school calendar where all school events would be displayed. This would be a very useful feature as the student would be able to see any relevant events such as sports fixtures and term dates. The student’s timetable could be achieved in the same manner.

Stakeholder	Their needs	How they will make use of this solution	Why it is appropriate to their needs
Student	Record their homework, coursework and exams. Access educational resources including information pages.	They can insert tasks into the program which can be sorted. A resources section can be accessed by the student to improve key skills.	The main function of a planner is to record tasks; this will be the primary action that a student would perform with the traditional planner. Because of this, it is vital that the student should be able to record any tasks that can be given at school. As well as this, tasks can be sorted so that the user can manage their time better by sorting these tasks by due date or by other task data to prioritise more important tasks. Another need is that they could access additional support material if necessary to improve their learning. This could include a map, translations, a periodic table and other similar material.

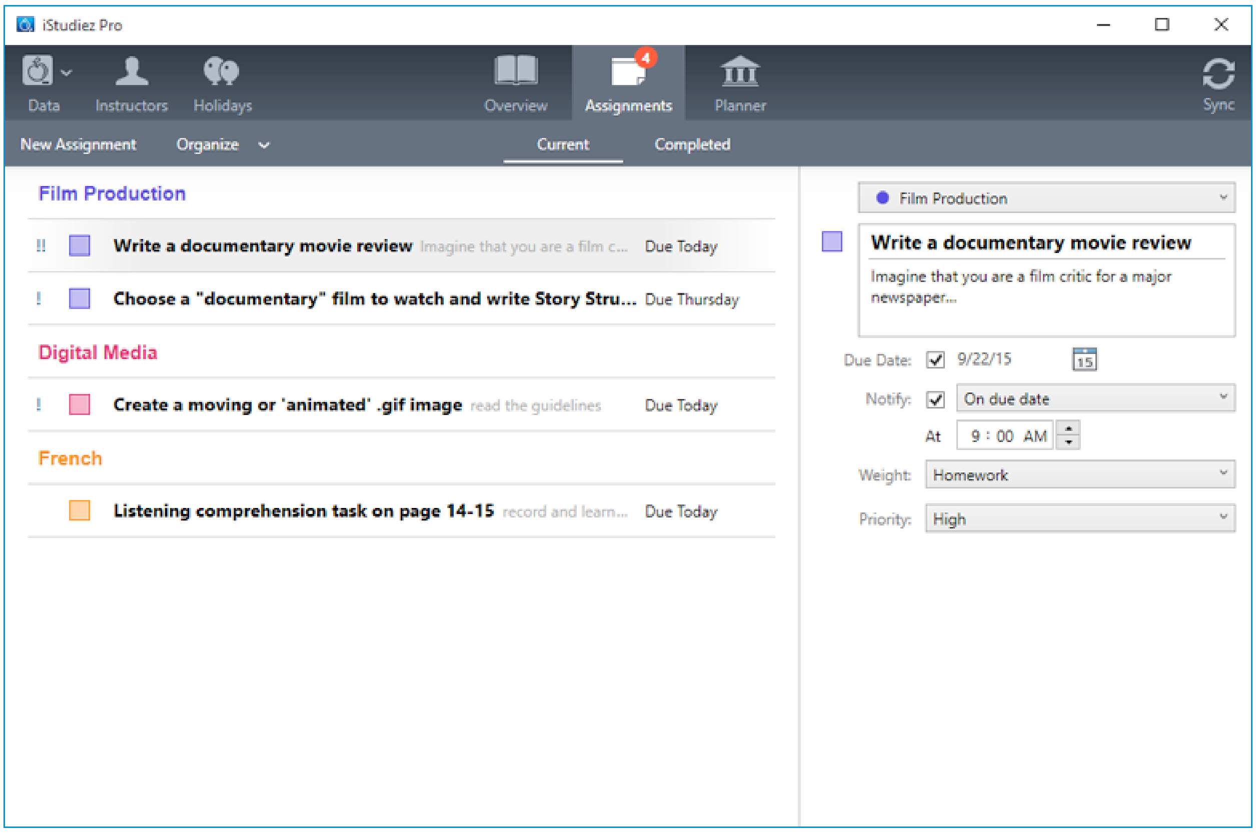
Stakeholder	Their needs	How they will make use of this solution	Why it is appropriate to their needs
Parent	Parents should be able to review the progress of their child or children by reviewing their planners and checking if their tasks have been completed. The schools contact details would be stored within a planner which a parent may need to use to contact the school. Planners can be used as a communication method between the teachers and the student's parents. View the schools term dates.	Parents can monitor the student's progress by checking tasks have been marked as complete and that tasks are not overdue. Contact details provided by the school would be useful for any enquiries that parents may have regarding current events or the school. By adding a notes section, teachers or form tutors can add notes for the parents to read and possibly contact the teachers if necessary. Parents will need to know when each term starts and finishes for their own planning. If these are made available, then the parents can use these to make arrangements without affecting the studen'ts attendance.	It is important for parents to keep track of their child's progress and by allowing a parent to access the student's planner, they can easily view how their school activities are going. If they identify an area of concern within a subject or across multiple from inspection, they could consult the school or raise these concerns at a parents evening to resolve any isues. Parents must be able to contact the school when required. By adding in a page to view these details will help the parent as all forms of contact will be available. Parents do not usually see the student's teachers regularly and therefore an alternative method for communication is used. A notes section within a planner is a common method for teachers and parents to communicate. Term dates will be indispensable for parents as planning for their holidays or other out of school activities is dependant on the school's term times.

Stakeholder	Their needs	How they will make use of this solution	Why it is appropriate to their needs
School	Personalisation of the product	Schools will add their branding to their planners.	Applying their logo, colour scheme or motto on the product will help present the overall identity of the school. Although branding does not need to be too prominent, it can improve with aesthetics and complement other school materials and the uniform. Schools are passionate about their identity and public image so they could add in their styles and logo to advertise this.

3.1.3 - Research the Problem:

A company called iStudiez had created a solution to this problem. "The app allows you to easily manage all your homework and assignments. Assignments view is a comprehensive tool to help you organize your academic work." *iStudiez Pro for Windows - Best App for Students*. Available at: http://istudentpro.com/ [Accessed 14 Sep. 2016]. This is a quote from their website stating the advantages of their software, specifically the one page that handles the assignments.

The 'assignments' page in this program is very similar to the program that I'd like to make as the purpose of my solution. Below is an image from their website showing this assignments pages.



iStudiez Pro

Data · Instructors · Holidays · Overview · Assignments · Planner · Sync

New Assignment · Organize · Current · Completed

Film Production

!! Write a documentary movie review — Imagine that you are a film c... — Due Today

! Choose a "documentary" film to watch and write Story Stru... — Due Thursday

Digital Media

! Create a moving or 'animated' .gif image — read the guidelines — Due Today

French

Listening comprehension task on page 14-15 — record and learn... — Due Today

Film Production

Write a documentary movie review

Imagine that you are a film critic for a major newspaper...

Due Date: ✓ 9/22/15

Notify: ✓ On due date

At 9 : 00 AM

Weight: Homework

Priority: High

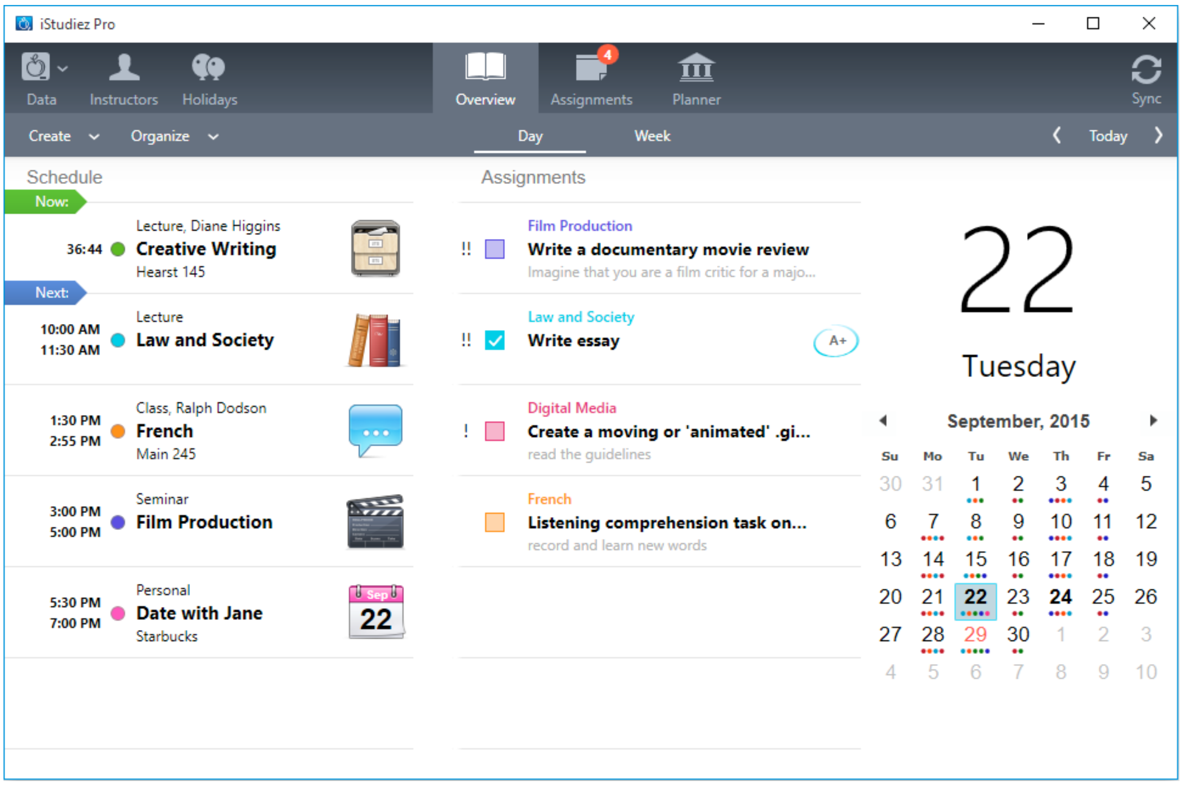
Figure 1.1. *Assignments Page* **At:** http://istudentpro.com/windows.php

There are many features on this page that will be very helpful to students as well as functional components for the working of the program. They have every task listed with titles, dues dates, subject and importance. This is a suitable set of data to have present for each task as it is only the essential facts that the user can see at a glance and know instantly, the priority of tasks and what they are. There is also an option to organize the data, the tasks can be sorted by date, course and priority. I will add in similar sorting options for my tasks pages based on the data that is recorded.

The way in which data is modified in the program is with a right-hand pane. There is a lot of data collected here which include: subject, task name, description, due date, time, weight and priority. For the purpose of my program, I may not request all the data as they have here.

This program has the ability to notify the student at a set time. This would be a useful feature if their program were to be running constantly, even as a background process. Notifications could be in the form of a pop up box to alert the user of an upcoming event.

The ‘overview’ page shows three panes. Each complementing each other to provide a general agenda showing all relevant information for the day.



Schedule

Now:

36:44 Lecture, Diane Higgins **Creative Writing** Hearst 145

Next:

10:00 AM 11:30 AM Lecture **Law and Society**

1:30 PM 2:55 PM Class, Ralph Dodson **French** Main 245

3:00 PM 5:00 PM Seminar **Film Production**

5:30 PM 7:00 PM Personal **Date with Jane** Starbucks

Assignments

!! Film Production **Write a documentary movie review** Imagine that you are a film critic for a majo...

!! Law and Society **Write essay** A+

! Digital Media **Create a moving or 'animated' .gi...** read the guidelines

French **Listening comprehension task on...** record and learn new words

22

Tuesday

September, 2015

Su	Mo	Tu	We	Th	Fr	Sa
30	31	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	1	2	3
4	5	6	7	8	9	10

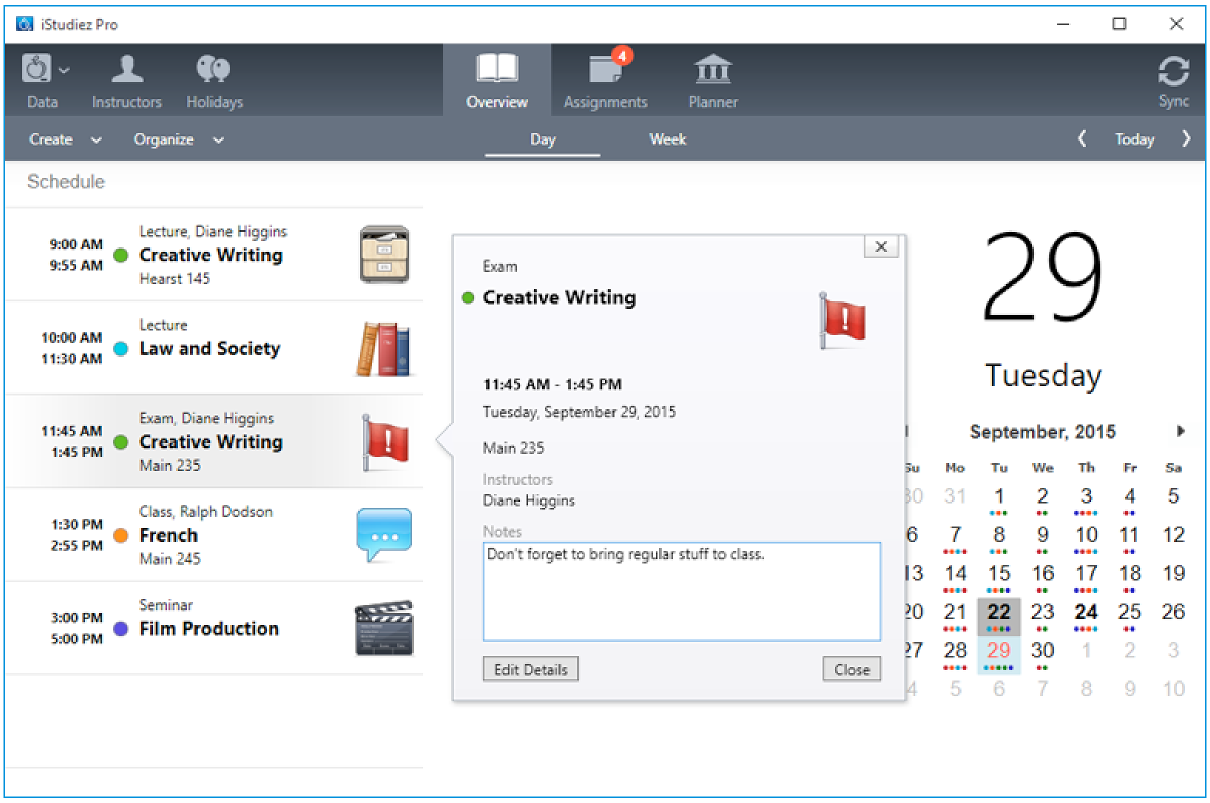
Figure 1.2. *Overview Page* **At:** http://istudentpro.com/windows.php

The first is a schedule with each subject listed in time order where the lesson due soonest is displayed first. This is different method to display the user’s timetable but only the upcoming lessons for the current or upcoming days. A very useful asset to the page focusing upon the student’s agenda.

The next pane consists of upcoming assignments. These tasks are again, listed in time order and show five pieces of data per task: task, subject, description, priority and grade achieved. Tasks here can be mark as completed too with the use of a simple check box.

The final pane shows a calendar. The calendar displays the current month only with the ability to switch between months. The current date is highlighted with a blue filled box within the calendar. There is also the large text number and the weekday positioned just above this. Each day where the student has a subject will have the subject’s coloured circle below the day’s number in the calendar. If the student colour codes their lessons, they could easily see the upcoming lessons at any date.

Tasks on the overview page can be easily modified. Upon selecting a task, a window will appear with all of the tasks details. By default, the notes box is selected for editing. The user has a choice to edit the details for the task or to close the window.

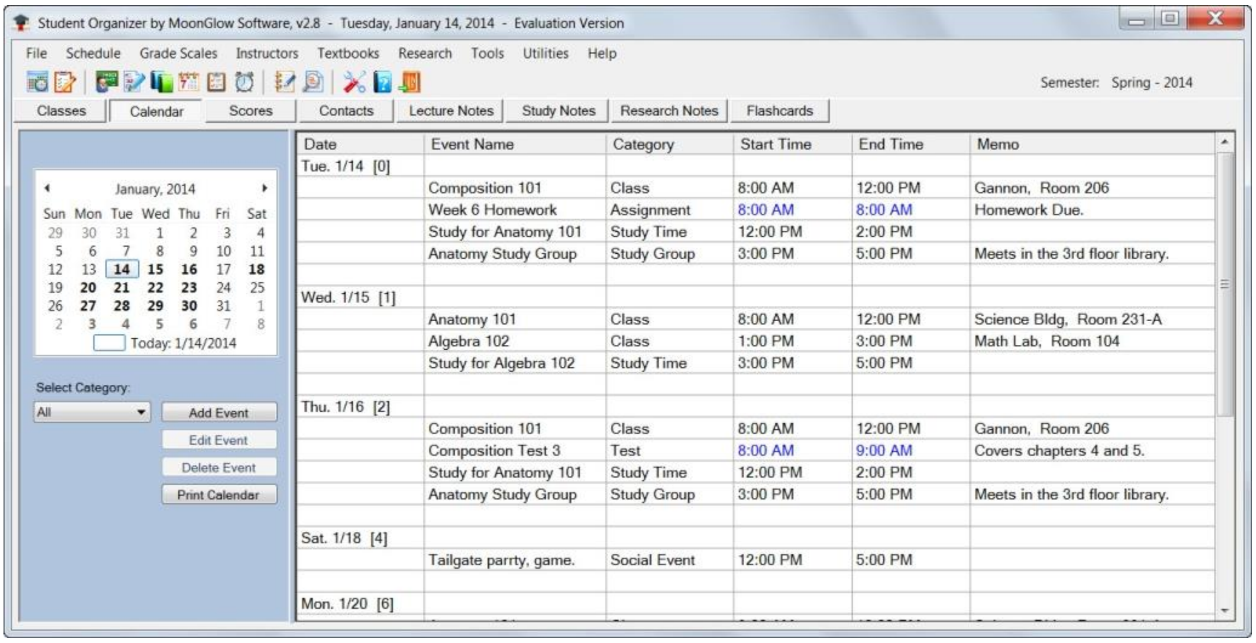


9:00 AM – 9:55 AM — Lecture, Diane Higgins — **Creative Writing** — Hearst 145
10:00 AM – 11:30 AM — Lecture — **Law and Society**
11:45 AM – 1:45 PM — Exam, Diane Higgins — **Creative Writing** — Main 235
1:30 PM – 2:55 PM — Class, Ralph Dodson — **French** — Main 245
3:00 PM – 5:00 PM — Seminar — **Film Production**

Exam
Creative Writing
11:45 AM - 1:45 PM
Tuesday, September 29, 2015
Main 235
Instructors
Diane Higgins
Notes
Don't forget to bring regular stuff to class.

Figure 1.3. *Selecting tasks in Overview* **At:** http://istudentpro.com/windows.php

Another example of an existing solution to this problem was created by a company called MoonGlow Software. Their software has many features and pages, more than the software above. Below is a screen shot of the page that is like what I plan to create.


Student Organizer by MoonGlow Software, v2.8 - Tuesday, January 14, 2014 - Evaluation Version
File Schedule Grade Scales Instructors Textbooks Research Tools Utilities Help
Semester: Spring - 2014
Classes Calendar Scores Contacts Lecture Notes Study Notes Research Notes Flashcards
January, 2014
Today: 1/14/2014
Select Category:
All
Add Event
Edit Event
Delete Event
Print Calendar

Date	Event Name	Category	Start Time	End Time	Memo
Tue. 1/14 [0]					
	Composition 101	Class	8:00 AM	12:00 PM	Gannon, Room 206
	Week 6 Homework	Assignment	8:00 AM	8:00 AM	Homework Due.
	Study for Anatomy 101	Study Time	12:00 PM	2:00 PM	
	Anatomy Study Group	Study Group	3:00 PM	5:00 PM	Meets in the 3rd floor library.
Wed. 1/15 [1]					
	Anatomy 101	Class	8:00 AM	12:00 PM	Science Bldg, Room 231-A
	Algebra 102	Class	1:00 PM	3:00 PM	Math Lab, Room 104
	Study for Algebra 102	Study Time	3:00 PM	5:00 PM	
Thu. 1/16 [2]					
	Composition 101	Class	8:00 AM	12:00 PM	Gannon, Room 206
	Composition Test 3	Test	8:00 AM	9:00 AM	Covers chapters 4 and 5.
	Study for Anatomy 101	Study Time	12:00 PM	2:00 PM	
	Anatomy Study Group	Study Group	3:00 PM	5:00 PM	Meets in the 3rd floor library.
Sat. 1/18 [4]					
	Tailgate parrty, game.	Social Event	12:00 PM	5:00 PM	
Mon. 1/20 [6]					

Figure 2.1. *Calendar Page* **At:** http://www.moonglowsoftware.com/student-organizer.html

On the calendar page, there is a table of upcoming events. These include lessons, assignments and study time. Each event includes a name, category, start time, end time and memo. These would be useful as this is a calendar, therefore the start and end times would be necessary to sort these events into chronological order. As I am not planning to create a calendar in with my solution, I will only need one time, a due date. The memo column would be useful however, so that the user can have a small title and follow with a longer description. Classes, assignment due dates and study times are added automatically added to this calendar. Reminders can be set for any events either recurring or non-recurring. These will be in the form of a pop up window that will display any reminders of any event that the user has set.

All notes are kept organized and searchable. This will allow the user to quickly find any notes of a task. There is also an integrated recorder that will allow the student to record their lessons and take written notes at the same time. Therefore, any information that the student may have missed or would need repeating can be easily found by playing back the recording.

Additional features such as quizzes and flashcards that can be created to test the students. As well as this, conversion and programmable formula calculators are included to aid the user in their studies.

Essential Features:

Lists or tables of tasks. This is very important as the user should be able to see essential information about a task and have the ability to sort this list. With this, the user has the option to easily sort their tasks which isn't easily, nor efficiently, possible to do without the use of computational methods. As well as sorting, the tasks can change in appearance, so that they could change colours based on importance or when it is due or even when expired, if not deleted. This shows the user which tasks to focus on.

The ability to edit a task, which is easily done with both computational and non-computational methods. However, using software to achieve this will result in cleaner, tidier sets of tasks. If the date of the task needs changing, for example; due to an extended deadline, then the program can move this task and put it in the correct place, if the list is sorted in date order.

There should be an efficient page system that allows the user to scroll through tasks. If there are too many to fit on a single page, there should be navigation buttons with clear instructions and indication that there will be further pages. This is important as the user should be able to create and delete many tasks without any issues with viewing them all. There should be a reasonable number of tasks displayed on a single page to reduce the need to change pages.

View important data relevant to the school, student or courses taken. This is another fundamental property of a school planner that should be incorporated into this program. I plan to add an option into the menu bar that can be used to access this data. The information displayed should be the details about the school.

Limitations:

This program can only be used on a computer. Although the operating system is very flexible, Linux, Windows or Mac, using a computer means there is little portability when checking or adding tasks. This is not suitable in schools or classrooms without computers, making this solution unsustainable and limited in use. It is also not an efficient solution for on the go use either. Another issue is the loading time of the computer as the computer must be loaded before my program can be viewed by the user to review their tasks. Checking a planner or a written task sheet requires little time as its just reading from a page.

Limitation	Justification
The computer running this program must have Python 3.5 installed.	This is required to run the program as it is written in the Python programming language. It must also be version 3.5 as this contains the correct modules that I have imported for use in my program, such as tkFont and sqlite3 where some modules do not exist on previous versions. In addition to this, the syntax that I have written this program in is specifically for 3.5.
All task data is stored in a database file. This database is stored locally on the machine running the software.	All data is written to the program file's directory. This requires full access in order to communicate correctly with the database and execute the main Python program. It must be able to read the database to load the tasks into the program, it must be able to write here as it should be able to save user's tasks and finally it must be able to execute files from this location in order for this program to load. A limitation here is that there will be no cloud storage available to access the data on multiple machines.
The software will only run on a computer.	I have written this in Python using Tkinter to handle my GUI. This is only supported on a computer and therefore it would not be easily transferred to a mobile device. Instead, a new application would need to be created for specific mobile operating systems.
Will not be able to store pictures with tasks.	Some tasks may require an illustration or a sketch with annotations, especially for practical based activities. This program will not be able to store pictures with the task data. Even if I were to add this in for further development, I would need to create a method to input drawings, however, this would require the user creating a sketch with a mouse or trackpad which wouldn't work very well at all. Because of these reasons, a normal planner would be able to accommodate for this but the software solution would not.

Parents would not be able to sign the planner.	Usually, parents would sign their child’s planner on a weekly basis upon checking their progress. Their teachers or form tutors would check that their planners have been signed and then sign it themselves. This is a working system where both, teachers, and parents, know that the other party has reviewed that week in the planner. This helps to quickly resolve any issues if the parent or teacher were to identify any areas of concern where tasks were not completed. In addition, it allows for notes to be written and read by either party to act as a form of communication. This would be unable to work as effectively with software as there wouldn’t be an effective input method to input a signature and is therefore a limitation to this program.

3.1.4 - Specify the Proposed Solution:

Requirements:

These are the initial requirements for the software that I plan to include for my program.

Requirement	Justification
The program will present tasks to the user in a page system.	The user must be able to view all of their tasks that they have recorded. By displaying the tasks, the user can review their upcoming events as they can with a conventional planner. A page system is required so that the screen is not overcrowded with information and that the user will not lose focus. It will also present the data more clearly and improve the ease of use for the software. By splitting tasks into multiple pages, the student can quickly see the tasks that are due in soonest without the need to browse through multiple pages in a standard planner.
The program must allow the user to easily insert a task.	One of the main features of a planner is to be able to record tasks. Therefore, this is an integral part of the software and must be included. I will design a page that is dedicated to accepting user input to the program. By separating this function into a new page, the user will only focus on one part of the program, no other elements will be available at that point.
The program must allow the user to delete a task.	Once a task has been completed, or the user no longer wants to keep a task, this should be able to be removed. It is important that tasks can be removed so that they will no longer appear first in the list, blocking other, more important, tasks form view.
The program must allow the user to edit a task.	There are many reasons why a student may need to modify a task, for example, there is an extension and the due date needs to be adjusted or a spelling mistake may have been made whilst typing in the subject. To accommodate for these cases, every tasks must be able to be modified.

Measurable Success Criteria:

A computer program should be used to organise school activities to save time and money when manually writing them out. This time could be better spent learning.

Measurable Success Criteria	Justification
The program should offer a sorting system.	This will allow the user to view their tasks in an order that they select. These sorting methods include sorting by, task name, subject, due date and the type of task. By including this feature, students could easily identify what they need to focus upon in terms of their upcoming tasks. A sorting method will present the tasks in alternative orders based on the data for all tasks involved.
Overdue tasks must be written in red text.	I have chosen to use red because this is a colour often associated with negativity. By only altering the text colour, the overdue task will become more prominent on the page and stand out to the user to alert them that this task is currently overdue.
Automatically delete old tasks.	An automatically deleting function would remove old tasks that are a set number of days past the due date. This would be a useful feature as older tasks are removed automatically so the user can keep track of current tasks to be completed. By leaving tasks that have been completed but not removed will build up overtime and would remove the benefit of a software solution because outdated information will be displayed first.
An option to load a page specifically to add a task.	I have chosen to create a separate page for when the user adds a task to the planner. This is because it will avoid confusion as there will only be one activity to focus on; adding a new task. This program should be as easy to use as possible to ensure that it is efficient in its role of a student planner for all years.

Tasks must be able to be modified.	In a conventional planner, tasks can be altered but not easily or sometimes not clearly as crossing out previously written words can become harder to read. To take full advantage of a computational solution, it would be suitable to allow the information entered by the user to be easily adjusted for a more cleanly presented page. As stated previously, there are many reasons to adjust any part of a task and therefore I will allow the user to alter any piece of data for any task in the program.
Tasks must be able to be deleted.	Tasks that are no longer required should be able to be removed to reduce redundant information. The program will list tasks based on their due date where the task due soonest will appear first. By leaving old, completed tasks in the program would reduce the benefit of this sorting algorithm as the more important tasks haven been moved further back in the pages.
There should be school information available to access (including: a school map, term dates, contact details and website links).	The program should act as a student's planner; therefore, there should be multiple aspects to this program. The main purpose is to store and sort tasks. In addition to this, I will be adding in a databank where the user can access important information about their school. The reason for including this data is to improve the functionality of the program whilst reducing need for external material that can be found in the paper based planners.
The software must be presented clearly so that it is easy to use.	In order to success as a software solution, the software must be presented clearly so that it is easy to use. It should be presented in a logical manner where widgets are placed with care and appropriately. The design should be aesthetically pleasing. All of these design factors will allow all year groups to make use of this software reducing need for other software solutions for the school. A clean design would entice students to make use of this utility, rather than deter them if it becomes overcomplicated and difficult to understand.

3.2 - DESIGN THE SOLUTION

3.2.1 - Decompose the Problem:

Incorporation of Agile Development:

I will adopt the Agile methodology when creating my program. I will use this group of methods because it allows me to easily adapt to changes in requirements. This is a vital part in the developing stage as I will have regular meetings with my client. These meetings will involve me asking questions about current prototypes or the progress on the solution so far. The client may suggest that they would like to see a feature added or even removed. They may suggest that come features should be altered in some way and I will need to adapt to these suggestions. An Agile approach will suit this project perfectly because of this.

This is an iterative process. A term that hereby means that there will be a repeating cycle of development involved. The program will not be complete within the first sprint. There will be constant change as I develop the solution further. Every iteration will produce a version of the software, where each will build on the previous with an increasing set of requirements. I can review the solution at the end of each iteration and evaluate to what extend the current version meet the measurable success criteria. This will take into account the client feedback once I allow them to test it. Any requirements that have not been fulfilled in their entirety or any requirements that have been suggested can then be added in the subsequent build.

The result of this will be a program well adapted to the client's feedback and therefore a vastly improved solution for use by students. This is because students would be testing these prototypes at the end of each iteration to provide their thoughts. By working with students, I can adapt this very well for my intended end users.

Structure of the Solution:

I will break this program down into functions. Each function will be a page or the set of instructions called due to the interactions of widgets on the GUI by the user. I have chosen to do this because it will allow me to focus on designing and coding one page at a time. Once a page has been completed, I can collect client feedback to improve the prototype before moving to the next component. Whilst doing so, I will be given a larger set of requirements to follow to ensure that the program meets the user's needs. By regularly meeting with the client, the solution will be exactly what the user wants and no time should be wasted developing unnecessary features that the client did not ask for. This reduces risk as I will be less likely to run into timing issues where I can't include a feature for a client because of this.

I will then split the main program into the following components:

- Set up theme
- Load tasks from the database
- Display the tasks
- Add tasks
- Delete tasks
- Information pages

Set up Theme:

I plan to split the setup of the theme for each page into three; creating the title bar, creating the menu bar and displaying the tabs. This will allow me to easily manage each part of the main program's theme without it affecting the running of the program or causing objects to block the view of others. By splitting up these three into separate functions, they can be easily edited and the changes will work for the whole program, therefore the theme will be the same for every page. Also, the order in which they load will mean that they shouldn't interfere with how the others will appear.

Load Tasks from the Database:

This component of the program will involve multiple stages. The first, I will open a connection with a database. Depending on the tab that the user has currently selected (Overview, Homework, Coursework or Exams) I will select every task that is in that task type create a two-dimensional array containing all the data for each record. This will allow me to use the data that the user has selected and easily use it anywhere within the program.

The next part is to split this data up so that it can be displayed to the user. I will be splitting up this data so that there are four tasks to a page and a page navigation system will be in place where the user could scroll through the pages of tasks.

Calculate the number of tasks in the database and create an array of them

The way in which this algorithm should work is that, I will create a variable to store the total number of tasks selected. At the start of the algorithm, I will set this to '0' as there will be no tasks at this point. Then, for every task in the array of tasks, it will increment the number of tasks by one. This will then have the value of the number of tasks in the database.

Calculate the total number of pages for each tab (based on 4 tasks per page)

The next stage of this algorithm is to calculate how many pages each tab would use. Basing the number of tasks per page to a maximum of four, I can calculate: how many pages there will be in total, how many full pages there will be (sets of four tasks) and how many remaining slots will be on the final page (if there are less than four tasks on a page). These three variables will be very important when organizing the tasks into pages.

Create the sets of task data

Once the data has been split and calculated, it now needs to be put back together into pages. I will call these 'sets' as there will be sets of tasks where a full set would contain four tasks. To track the current position of the user through theses sets of tasks, I will use an array called 'progress.' Where this variable will be declared before it is to be used, ensuring that the progress array will be empty from the start.

For the amount of sets, a minimum and maximum number of tasks is then calculated. This will be achieved by using the number of each set and multiplying it by 4 and for the maximum number, adding 3 to minimum value. These two values are then added to the progress array. This array will contain the start and end values for every task in every set.

For example:

If there were 10 tasks in the database, then this array would be created, '0,3,4,7,8,10.'
Tasks 0 to 3 make the first set (page 1)
Tasks 4 to 7 make the second set (page 2)
Tasks 8 to 10 make the final set (page 3)

Now this array will be used for showing the relevant tasks for the pages. The numbers within the array can be considered to be paired. When inserting tasks into a page, the use of this array makes it possible.

Display the Tasks:

To insert the appropriate tasks into a page, I can use the variables previously calculated with my algorithm. First, I will need to create a variable called 'set', which will be the set of data to insert.

In the tasks array, I can use their location number to retrieve the correct set. I will achieve this by using the progress array for the tasks. Select the tasks in the array 'tasks' which have the positions of the page number in the in the progress array and the tasks which have the position of the page number plus one in the progress array. Select any values between these two positions in the tasks array too.

For example:

```
tasks = ['Complete 1.1.3 Notes', 'Finish EPQ Feedback', 'Exercise 3A', 'Exercise 9B', 'ET6 Plan', Read P88']
progress = [0,3,4,5]
page = 0
set = []
```

When loading page one, the values would be selected as follows:

```
sets = tasks[progress[page]:progress[page+1]+1]
sets = tasks[progress[0]:progress[1]+1]
sets = tasks[0:3+1]
sets = tasks[0:4]
sets = 'Complete 1.1.3 Notes', 'Finish EPQ Feedback', 'Exercise 3A', 'Exercise 9B'
```

When loading page two, the values would be selected as follows:

```
sets = tasks[progress[page]:progress[page+1]+1]
sets = tasks[progress[2]:progress[3]+1]
sets = tasks[4:5+1]
sets = tasks[4:6]
sets = 'ET6 Plan', Read P88'
```

Add Tasks:

I have split this into its own section as this is a main component in this solution and should be able to be used anywhere in the program. This section should work by creating a screen where the user can input all the details for a task. These details include: the task's name, the subject of the task, the due date and the type of task.

There should be an option to delete the data that the user has entered so that the task will not be added and the option to save the data. Saving the data will insert the information entered into its own record within the tasks table of the database. If this task is saved, the tasks page will load and the task array refreshed so that the new tasks should then be visible.

Delete Tasks:

I will open a connection to the database and use the delete SQL statement to remove the task by its ID. I can get the ID from the tasks array assigned to the task. I will use the ID in this case and no other field as the ID is the unique variable in this table and using any other field could delete other tasks too.

Information Pages:

As this project is essentially a digital school planner, it is important that I include the other aspects that come with the conventional planner, not just the recording of tasks.

I will include the following three information pages:

- School map
- Contact details
- Useful websites

I chose to include a school map as this can help students who are new to the school. It will allow them to easily find where their lessons will be or where other facilities within the school can be found. This page will be made up of a single picture that will contain the map.

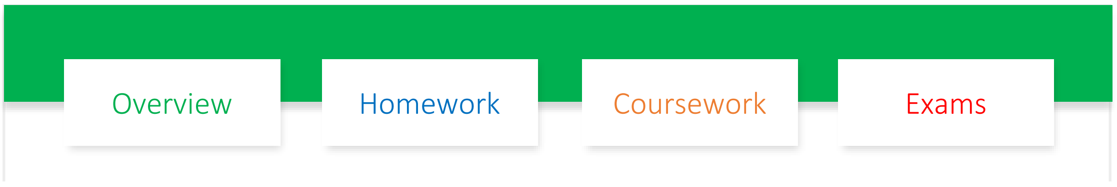
The school's contact details will also be displayed on another page. I will make all relevant information for contacting the school available by inserting text onto the page.

Any useful websites that students would find useful will be listed on a page with hyperlinks that will allow the user to open these easily.

3.2.2 - Describe the Solution:

Overall Plan:

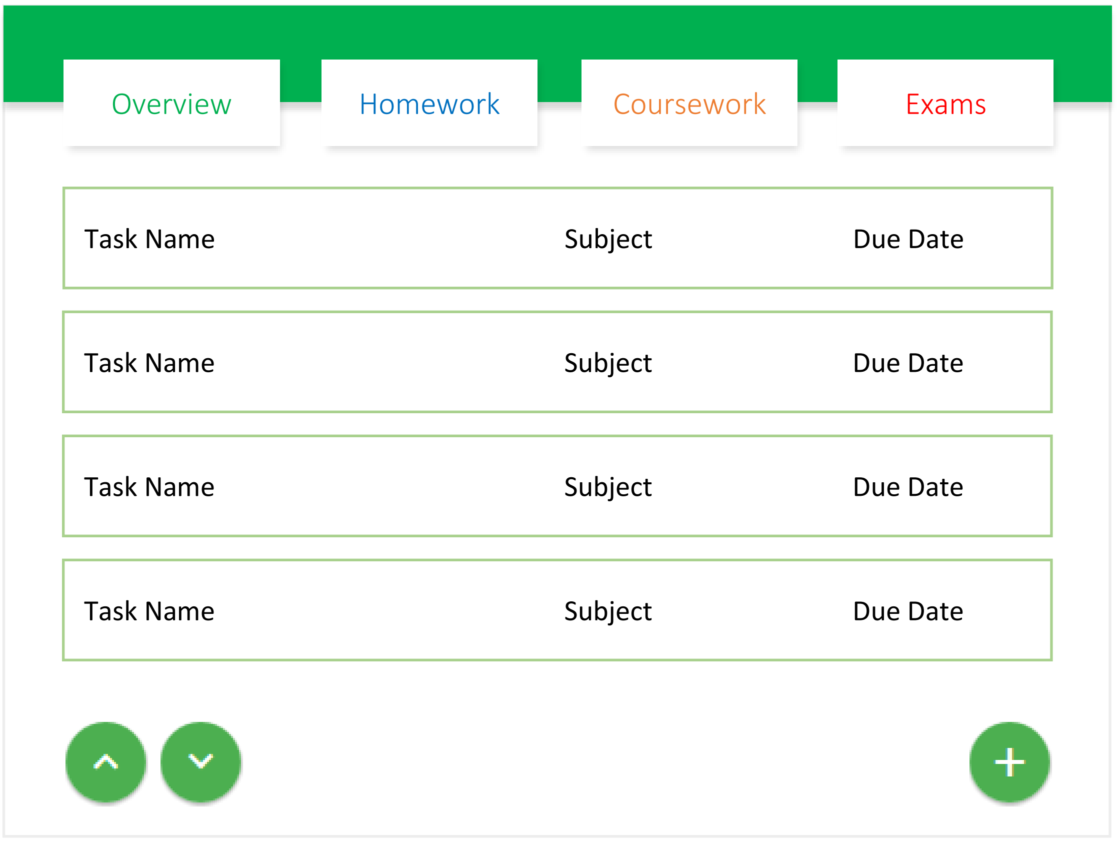
Header:



I will be using a consistent theme throughout the program, therefore I will use a header across every page in the program. As this program will allow the user to store multiple types of task, I will be using a system to view the tasks for this type. Tasks will be assigned to a task type; homework, coursework or an exam. I created four links based on these three task types. When clicked, the relevant tasks should be shown on a page. The fourth option I included was the overview link. I had decided that the student might want to view all their tasks regardless on the type.

I will also colour code these links to show contrast between different task types and pages. I plan to follow the colour scheme throughout the program.

Home Page:

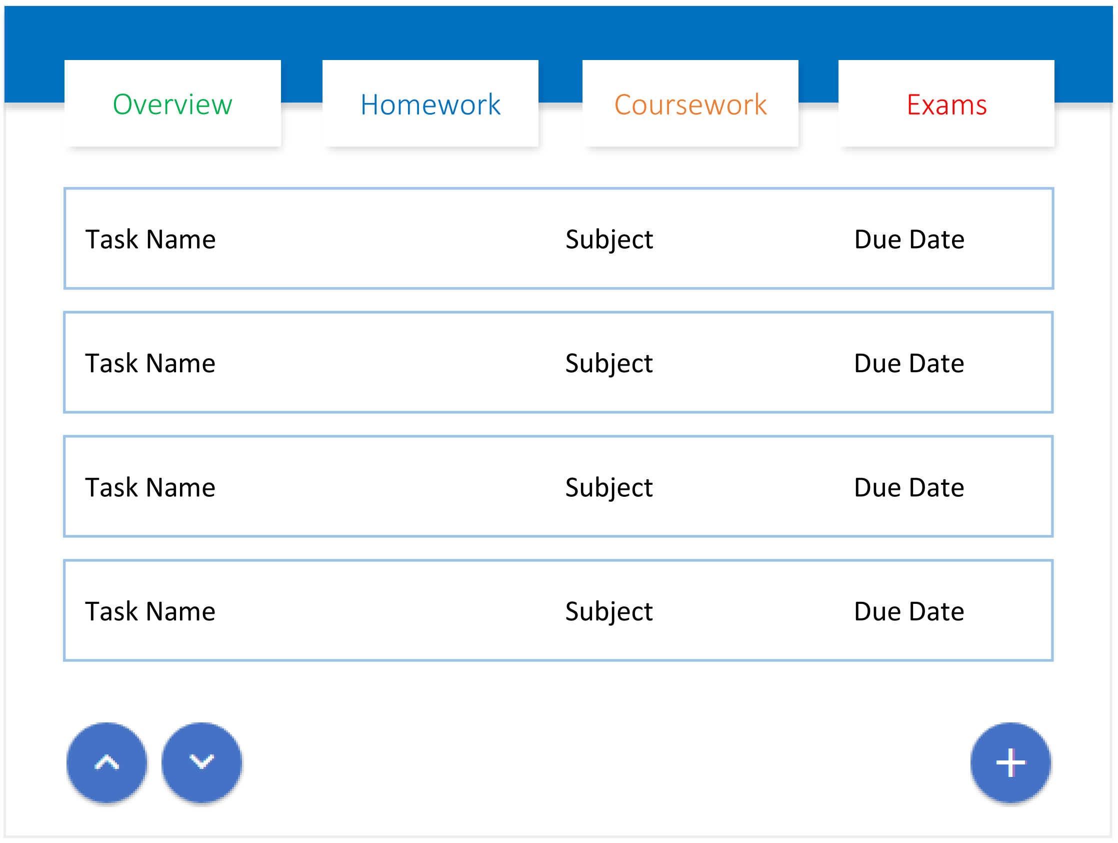


This is how I plan for the tasks to be displayed to the user. Each task will be contained within its own individual coloured rectangle. This will clearly split the data up on the page. I have only chosen to reveal the task name, the subject and the due date to the user. This is because it will be the only relevant data required for the student when viewing the list.

I have followed the green colour scheme with this page as this represents the overview tab. The buttons and task rectangles follow this with differing shades of green. I had decided that the user should first be shown the overview tab as the home page because this contains every single task in the program. This will also show the task that's due soonest out of every category.

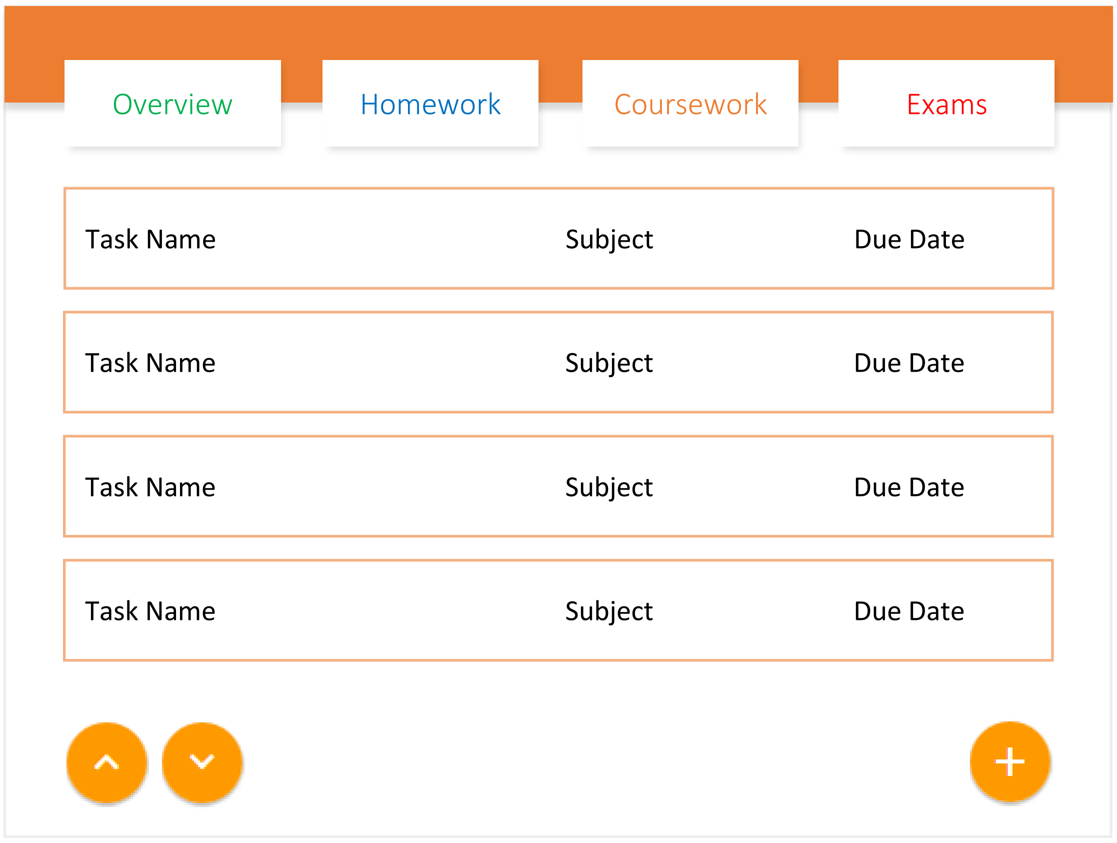
Each page will have three buttons, the first will return to the previous page of tasks, the second will load the next page of tasks and the third can be used to add tasks to the program. I had decided to use two buttons to control the scrolling as it will be easy for the user to navigate through the pages as they are clearly marked with arrows. The add task button will proceed to a new page which will show all the inputs to create a task.

Homework Tab:



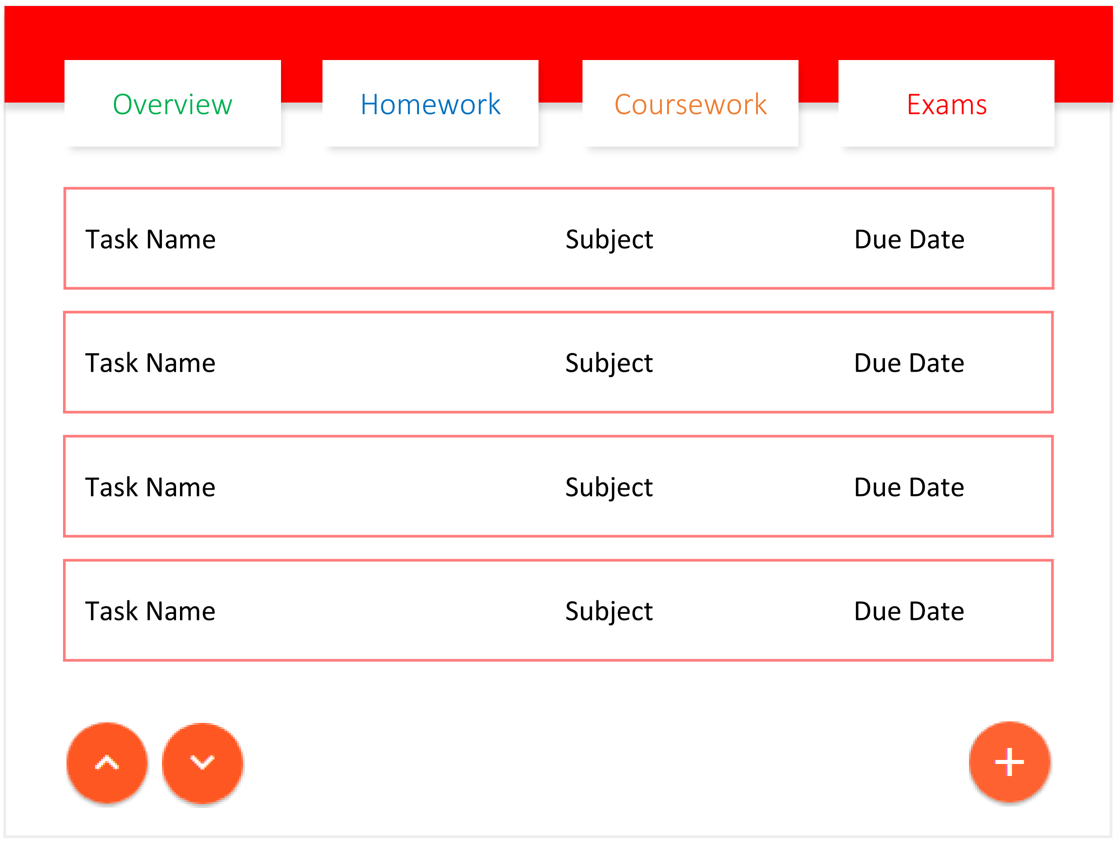
This tab will use the same template as the overview tab. The only differences in this page is that the colour scheme is now blue and the only tasks displayed are tasks with the task type as 'homework.'

Coursework Tab:



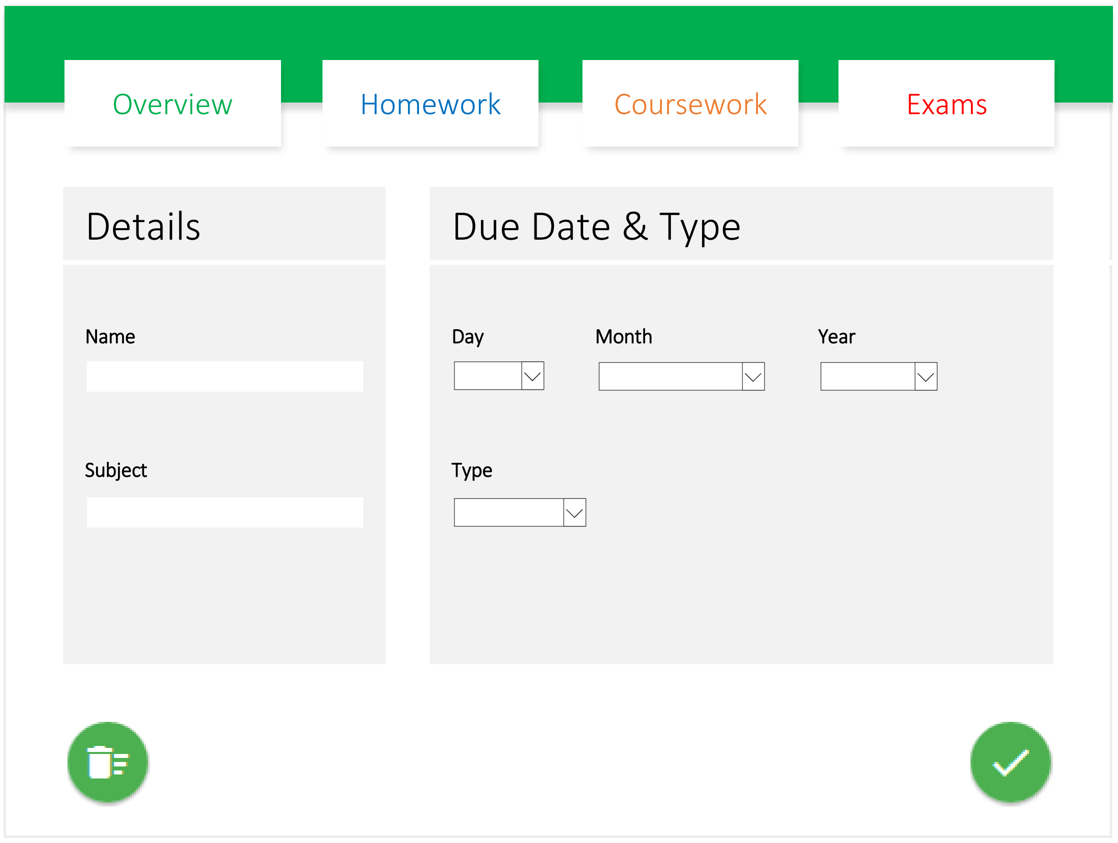
This tab will use the same template as the other tabs. The only differences in this page is that the colour scheme is now orange and the only tasks displayed are tasks with the task type as 'coursework.'

Exams Tab:



This tab will use the same template as the other tabs. The only differences in this page is that the colour scheme is now red and the only tasks displayed are tasks with the task type as 'exams.'

New Tab: Overview

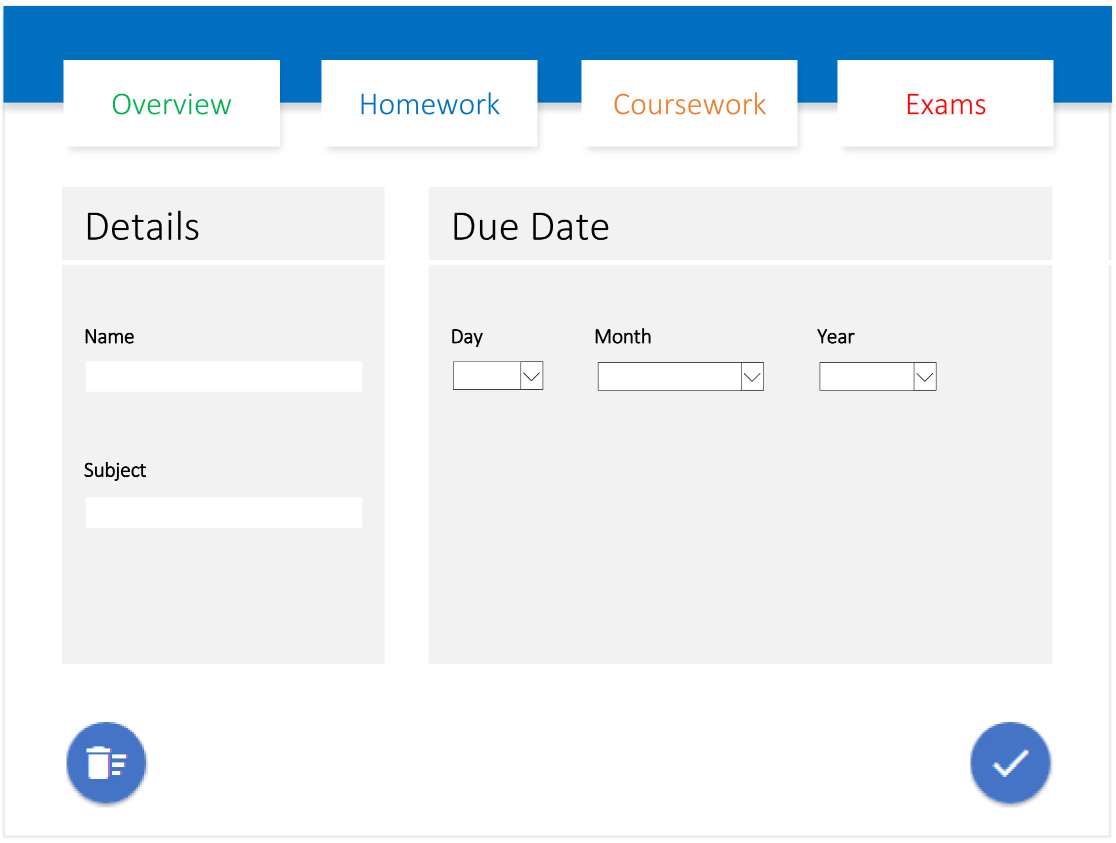


When the user clicks the lower right hand button containing the add icon, this page is shown. I have split this up into two boxes. In the box on the left, I have added two entry boxes, one that is labelled ‘Name’ and the other is labelled ‘Subject.’ Here, the user should enter the name of the task that they want to create and the subject that this task is for.

On the right, I have chosen to create a larger rectangle. Within this rectangle, I plan to add in three four drop down menus: where the first three are used to set the due date of the task and the final one is to set the task type. Using three dropdown menus for the date picker would allow a user to input any date easily. The final box will contain three values, homework, coursework and exams. The task type set here will be used when loading the pages of tasks. This is an important piece of data as the tabs depend on this to show tasks specific to the tab.

The page navigation buttons in the lower left will be replaced with a single delete button. I plan for this to delete any changes that the user had made in the current page and then return the user to the previous page where the tasks are loaded in again. In the lower right, the add task button will be replaced with a confirm button. Once the user has added in the details of the task that they would like to add, they can click this button to save the task into the program. Like the delete button, the program should load the user’s previous page and the new task should be listed within the tasks page.

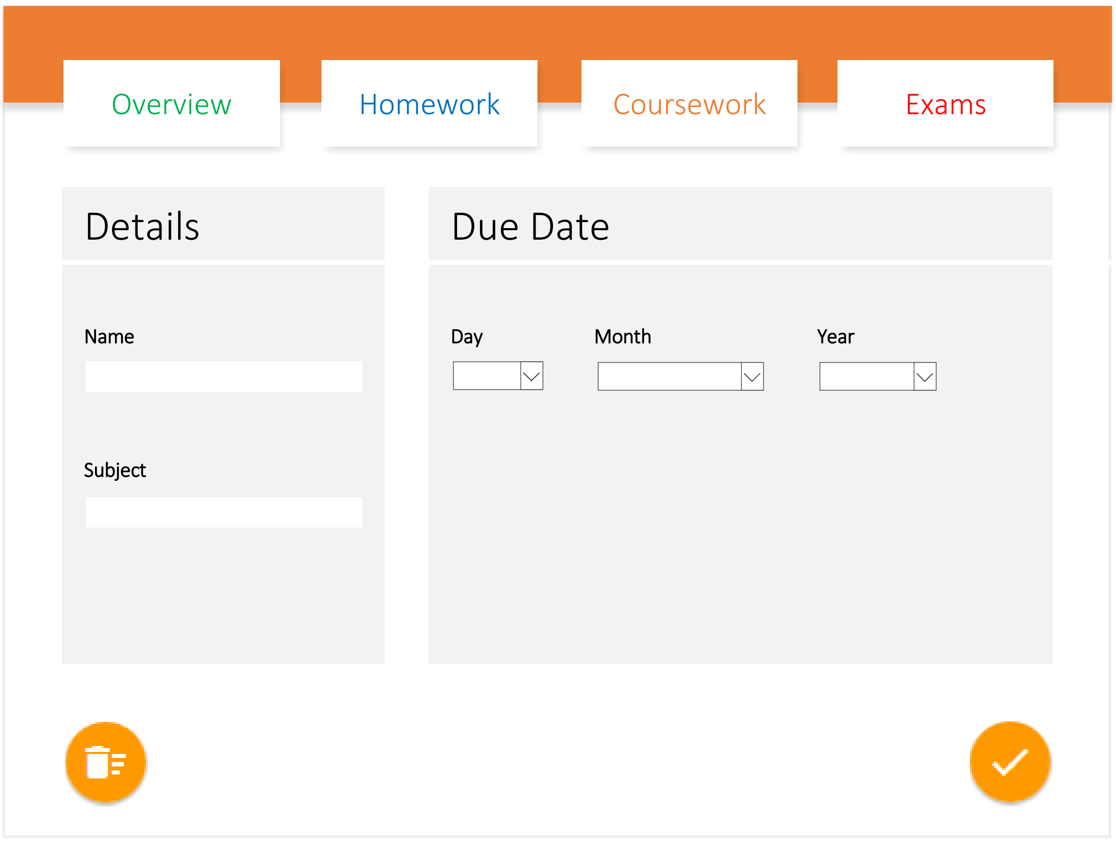
New Tab: Homework



This page will load when the user clicks the add task button when in the homework tab. The theme will continue, as blue buttons will be used. They will all operate in the same way. The homework tab will be loaded once the user has clicked the delete or add button and their functions have completed.

I have not added in a task type option for this page, as it should be assumed that the task to be added would be a homework. I will use this idea for my other two tabs in the program.

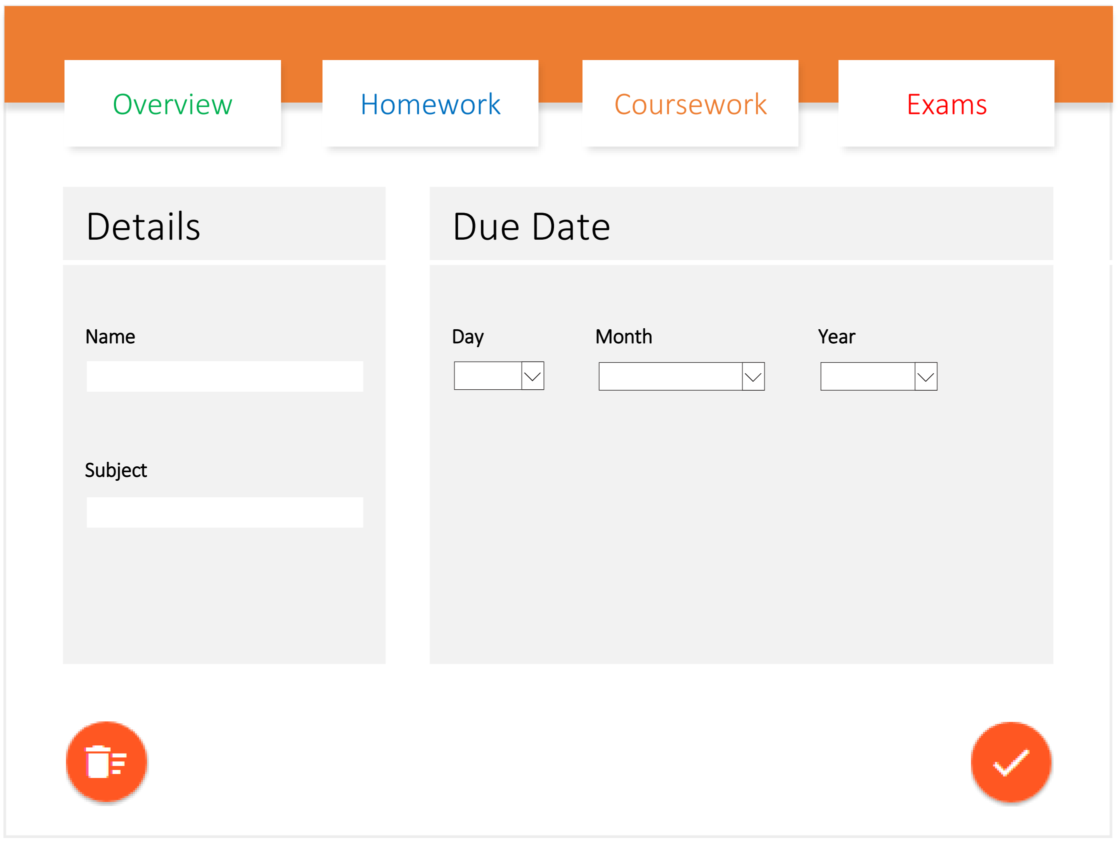
New Tab: Coursework



This page will load when the user clicks the add task button when in the coursework tab. The theme will continue, as orange buttons will be used. They will all operate in the same way. The coursework tab will be loaded once the user has clicked the delete or add button and their functions have completed.

I have not added in a task type option for this page, as it should be assumed that the task to be added would be a coursework. I will use this idea for my other two tabs in the program.

New Tab: Coursework



This page will load when the user clicks the add task button when in the exams tab. The theme will continue, as red buttons will be used. They will all operate in the same way. The exams tab will be loaded once the user has clicked the delete or add button and their functions have completed.

I have not added in a task type option for this page, as it should be assumed that the task to be added would be an exam. I will use this idea for my other two tabs in the program.

Load Tasks:

When the program loads, it must show the user every task in the database. This is because I will use the overview tab as the ‘home page’ of the program. When this page loads, or through any of the other three tabs, I will need a function that will set up the tasks in order to be presented upon request.

The algorithm here will work by fetching every task from the database for the selected tab. Once complete, the data will be added to a two-dimensional array of tasks. Then I will apply logic to create three variables. These will be required as there can only be up to four tasks displayed at any one time and they should all be shown in order where a page navigation should give the user control of the current set.

To start, the number of tasks is calculated. This will be required when splitting the tasks into their sets of four. To find the number of sets (groups of four adjacent tasks) will also find the number of pages required.

- If there are no tasks, then there are zero sets so therefore there will also be zero pages.
- If the number of tasks is a multiple of four, the number of sets is this multiple.
- If there is a remainder when dividing by four, sets is equal to the amount of times four will go into the number of tasks and then increase this value by one.

Once the number of sets has been calculated, the index of each task in the array should be grouped in pairs so that first and the last task’s index is saved. That will allow me to choose every task between these two values when displaying each page.

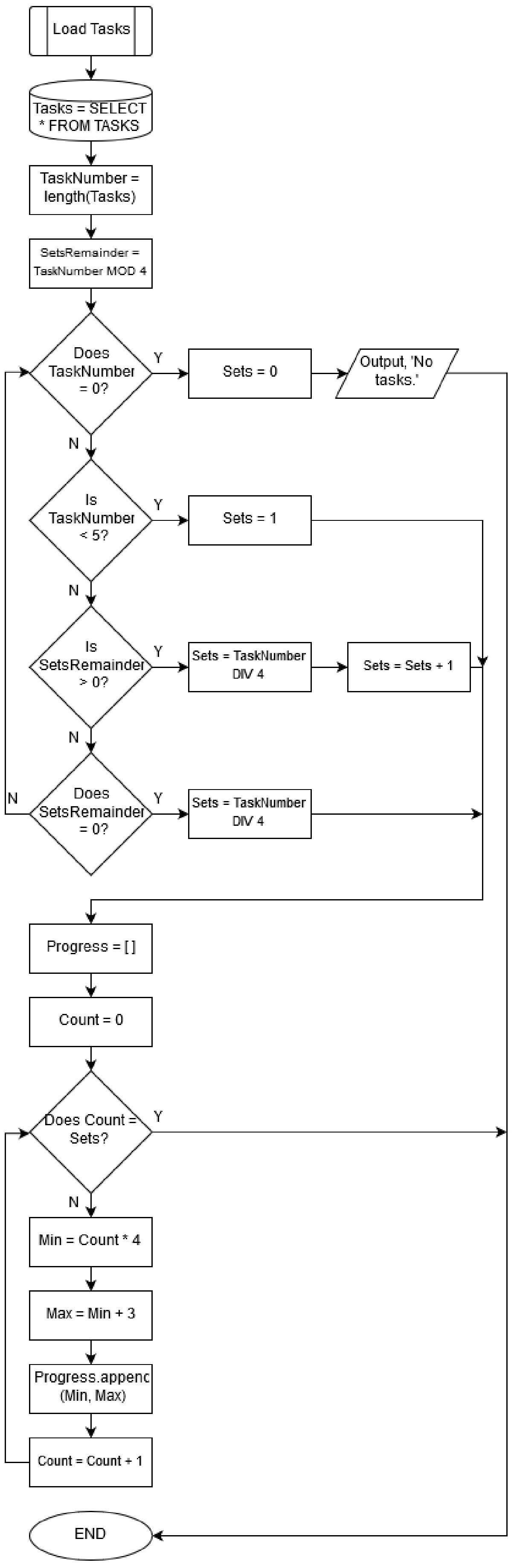
Pseudo Code:

```
START
Tasks = SELECT * FROM TASKS
TaskNumber = 0
FOR record IN Tasks:
       TaskNumber += 1
END FOR
SetsRemainder = TaskNumber % 4
IF TaskNumber == 0:
       Sets = 0
ELSE IF TaskNumber < 5:
       Sets = 1
ELSE IF SetsRemainder > 0:
       Sets = TaskNumber // 4
       Sets += 1
ELSE IF SetsRemainder == 0:
       Sets = taskNumber // 4
END IF
Progress = []
FOR i IN RANGE(LENGTH(Sets)):
       MinTask = (i) * 4
       MaxTask = MinTask + 3
       Progress.append(MinTask)
       Progress.append(MaxTask)
END FOR
END
```

Data Table:

Variable	Data Type	Rationale
$Tasks$	Array	Every record stored in the database table, 'TASKS' will be stored in this array. It must be an array as this can be used to select specific pieces of data when using it in the program.
$TaskNumber$	Integer	This will be used to store the number of tasks that are in the array, Tasks.
$record$	Integer	A temporary variable that will be used in an iteration loop to determine the number of tasks stored in the database.
$SetsRemainder$	Integer	Stores the number of tasks that are left over when the total number of tasks is divided by 4. This will be used to determine how many tasks will be left over as there are 4 to a page.
$Sets$	Integer	Stores the number of sets of tasks. A set is considered to be 4 tasks. Any remaining tasks will also increase the 'Sets' variable by 1.
$Progress$	Array	This will be used as a page pointer. An array is important as it can be appended to and future aspects of the program will rely on selecting certain positions in this array to find a value of tasks to display.
i	Integer	A temporary variable that will be used in an iteration loop to populate the 'Progress' array.
$MinTask$	Integer	Stores the variable for the variable, 'i' multiplied by 4. This relates to the position in the 'Tasks' array where the task at this location will be the first in the page.
$MaxTask$	Integer	Stores the variable for the variable, 'MinTask' added to 3. This relates to the position in the 'Tasks' array where the task at this location will be the last in the page.

Flowchart:


Load Tasks
Tasks = SELECT * FROM TASKS
TaskNumber = length(Tasks)
SetsRemainder = TaskNumber MOD 4
Does TaskNumber = 0?
Y
Sets = 0
Output, 'No tasks.'
N
Is TaskNumber < 5?
Y
Sets = 1
N
Is SetsRemainder > 0?
Y
Sets = TaskNumber DIV 4
Sets = Sets + 1
N
Does SetsRemainder = 0?
Y
Sets = TaskNumber DIV 4
N
Progress = [ ]
Count = 0
Does Count = Sets?
Y
N
Min = Count * 4
Max = Min + 3
Progress.append (Min, Max)
Count = Count + 1
END

Add Tasks:

This element of the solution will be accessed by a button on any of the four tasks pages or through the use of the menu bar. When this button is pressed, the current page is cleared and is replaced with the add task page. This page will contain entry boxes and drop down menus for the user to input all the relevant information.

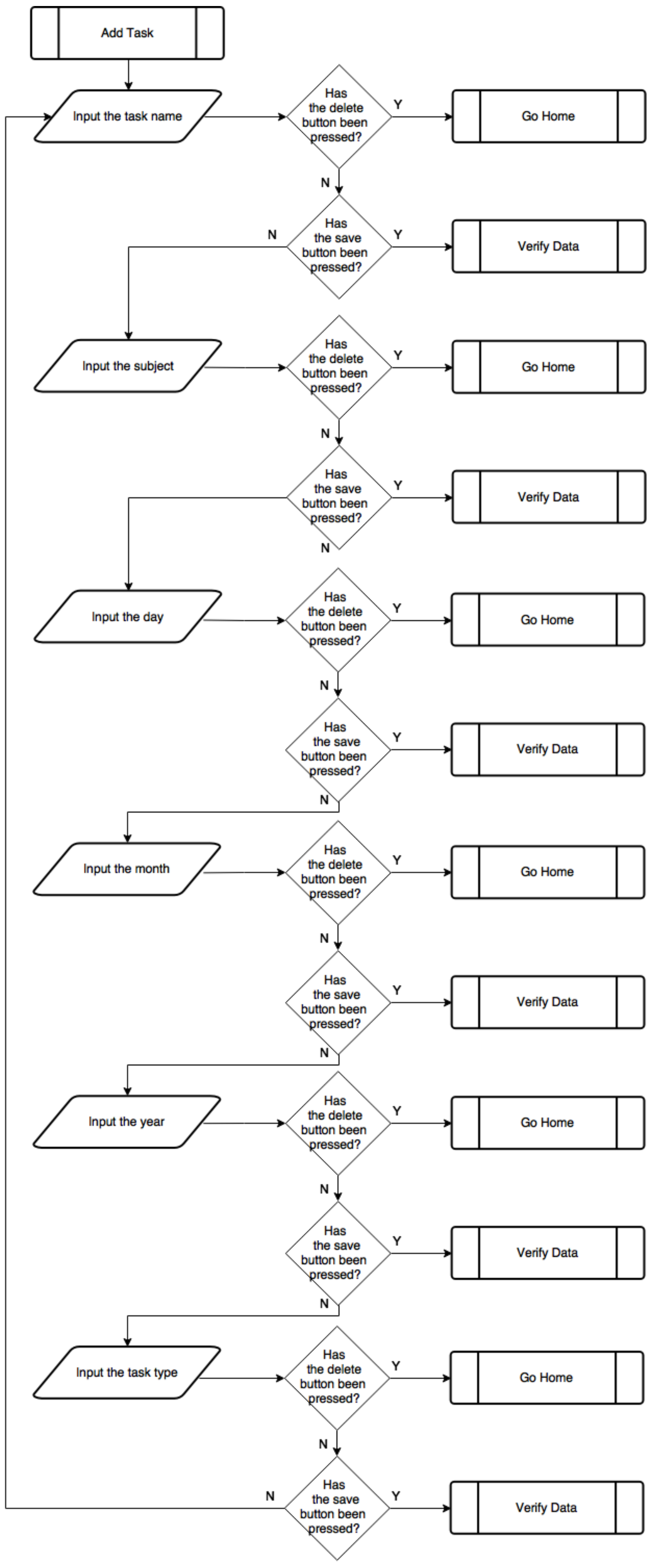
Pseudo Code:

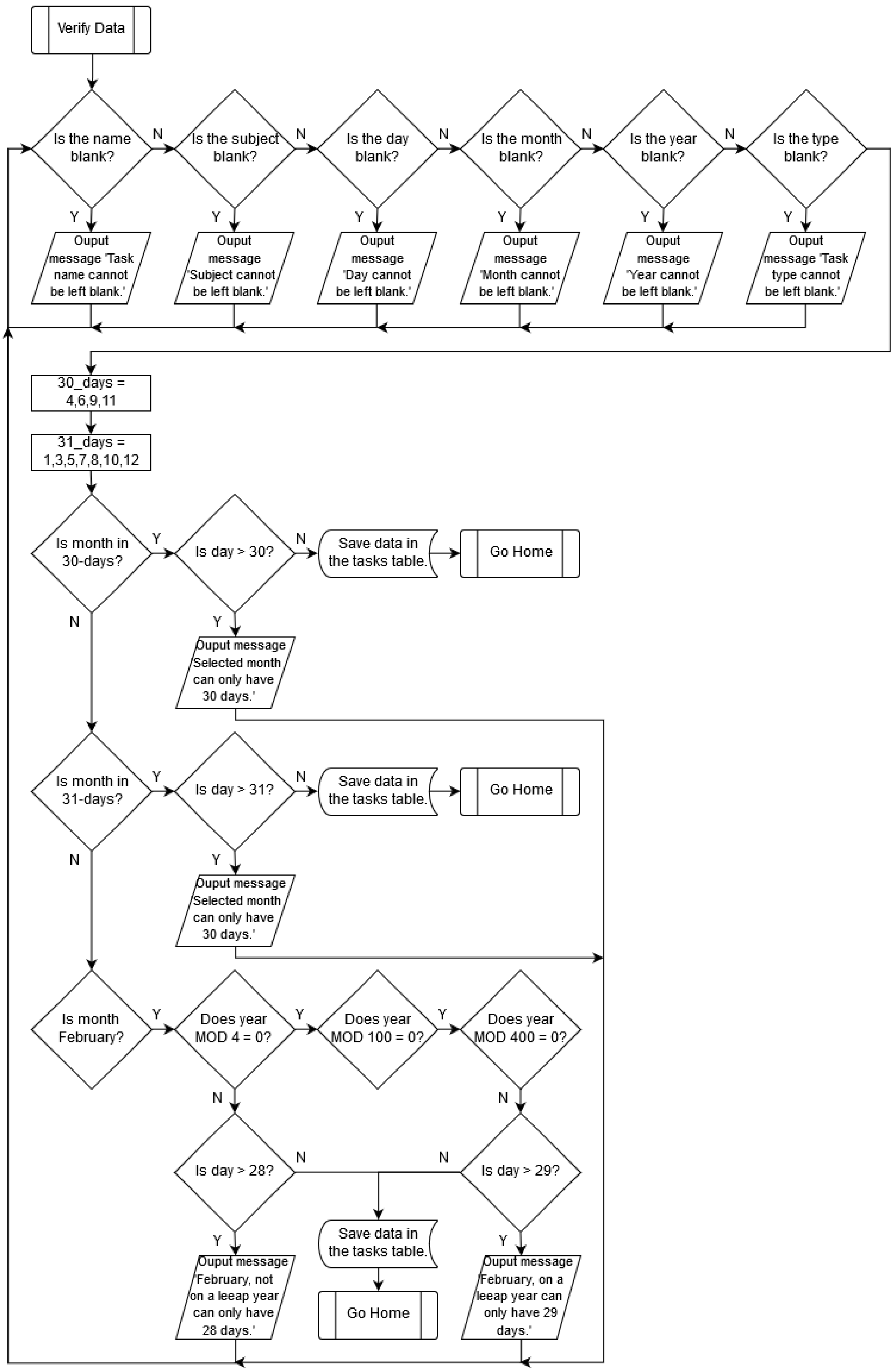
```
START
Name = USERINPUT
Subject = USERINPUT
Day = USERINPUT
Month = USERINPUT
Year = USERINPUT
Type = USERINPUT
IF Name = “” THEN
        OUTPUT “Task name cannot be left blank.”
        RETURN
ELSE IF Subject = “” THEN
        OUTPUT “Subject cannot be left blank.”
        RETURN
ELSE IF Day = “” THEN
        OUTPUT “Day cannot be left blank.”
        RETURN
ELSE IF Month = “” THEN
        OUTPUT “Month cannot be left blank.”
        RETURN
ELSE IF Year = “” THEN
        OUTPUT “Year cannot be left blank.”
        RETURN
ELSE IF Type = “” THEN
        OUTPUT “Task type cannot be left blank.”
        RETURN
END IF
30_days = [4,6,9,11]
31_days = [1,3,5,7,8,10,12]
IF Month IN 30_days AND Day > 30:
        OUTPUT Month + “can only have 30 days.”
        RETURN
ELSE IF Month IN 31_days AND Day > 31:
        OUTPUT Month + “can only have 31 days.”
        RETURN
ELSE IF Month = 2:
        LeapYear = False
        IF Year % 4 = 0:
                LeapYear = True
                IF Year % 100 = 0:
                        LeapYear = False
                ELSE IF Year % 400 = 0:
                        LeapYear = True
                END IF
        END IF
        IF LeapYear = False AND Day > 28:
                OUTPUT “Not on a leap year, February can only have 28 days.”
                RETURN
        ELSE IF LeapYear = True AND Day > 29:
                OUTPUT “On a leap year, February can only have 29 days.”
                RETURN
        END IF
END IF
INSERT INTO TASKS (Name, Subject, Due Date, Type)
END
```

Data Table:

Variable	Data Type	Rationale
$Name$	String	Stores the name of the task entered by the user.
$Subject$	String	Stores the subject of the task entered by the user.
Day	Integer	Stores the day for the due date of the task entered by the user.
$Month$	Integer	Stores the month for the due date of the task entered by the user.
$Year$	Integer	Stores the year for the due date of the task entered by the user.
$Type$	String	Stores the type of task entered by the user.
30_days	Array	An array that stores the integer value of every month that has a maximum of 30 days.
31_days	Array	An array that stores the integer value of every month that has a maximum of 31 days.
$LeapYear$	Boolean	A variable that is true when the date selected falls on a leap year and false if it does not. Used to verify the day selected when the month selected is February.

Flowchart:


Add Task
Input the task name
Has the delete button been pressed?
Y
Go Home
N
Has the save button been pressed?
Y
Verify Data
N
Input the subject
Has the delete button been pressed?
Y
Go Home
N
Has the save button been pressed?
Y
Verify Data
N
Input the day
Has the delete button been pressed?
Y
Go Home
N
Has the save button been pressed?
Y
Verify Data
N
Input the month
Has the delete button been pressed?
Y
Go Home
N
Has the save button been pressed?
Y
Verify Data
N
Input the year
Has the delete button been pressed?
Y
Go Home
N
Has the save button been pressed?
Y
Verify Data
N
Input the task type
Has the delete button been pressed?
Y
Go Home
N
Has the save button been pressed?
Y
Verify Data
N


Verify Data
Is the name blank?
Is the subject blank?
Is the day blank?
Is the month blank?
Is the year blank?
Is the type blank?
Y
N
Ouput message 'Task name cannot be left blank.'
Ouput message 'Subject cannot be left blank.'
Ouput message 'Day cannot be left blank.'
Ouput message 'Month cannot be left blank.'
Ouput message 'Year cannot be left blank.'
Ouput message 'Task type cannot be left blank.'
30_days = 4,6,9,11
31_days = 1,3,5,7,8,10,12
Is month in 30-days?
Is day > 30?
Save data in the tasks table.
Go Home
Ouput message 'Selected month can only have 30 days.'
Is month in 31-days?
Is day > 31?
Save data in the tasks table.
Go Home
Ouput message 'Selected month can only have 30 days.'
Is month February?
Does year MOD 4 = 0?
Does year MOD 100 = 0?
Does year MOD 400 = 0?
Is day > 28?
Is day > 29?
Save data in the tasks table.
Go Home
Ouput message 'February, not on a leeap year can only have 28 days.'
Ouput message 'February, on a leeap year can only have 29 days.'

Database:

I will be using a database for storing all details about a task. I have chosen to store the data in a database as I think it will be the best form of data storage and the most suitable for this project. Alternatives that I could use include storing the data in raw text files or in JSON format. The disadvantage of these are that data cannot be ordered, modified or removed that easily. A database can easily sort data, output ordered data, insert records and delete or modify.

Within my database, I will be using a single table to store the tasks. The table structure will be as follows:

Column	Data Type	Rationale
ID	INT PRIMARY KEY	Every record stored in a table must have a unique identifier in order to be able to edit the data. This unique ID will allow me to access a specific record and perform crucial operations such as delete or modify.
$TASK$	TEXT	The task name will be recorded within in this field. The data type will be text so that the user can enter any character as their task name.
$SUBJECT$	TEXT	The task's subject will be saved in this field. Again, the type will be text as the user should be able to insert any character for their subject.
$DATE$	DATE	The date field will be used to save the due date of the task. The date type will require data to be entered in the format YYYY/MM/DD. By using the date type, I will be able to sort the records by their dates too.
TAB	TEXT	As there are four main tabs, these will select the data with their tab. This filed will store one of three values that will correspond to one of the three tabs.

Usability Features:

I will be using various forms of usability features in this program.

The design will be clean, clutter free with a simple layout. The purpose of this is to reduce confusion and increase productivity through the simplicity of displaying the relevant data clearly to the student. Buttons will al follow the same theme. I will use Google's Material Design floating action button combined with their icon set to produce each button within this solution.

I will select appropriate input methods for the GUI. For example, I will use dropdown menus for date selection rather than an entry box for ease of use. I will not use any scrolling pages where widgets are partial or not at all in view. By doing so, the user can easily see what is on screen and the data that they require form the program.

All input methods should use validation where any invalid inputs will result in a message box explaining the reason for the error. This will inform the user on why their input was invalid, allowing the user to understand the problem and correct it easily.

Usability Feature	Justification
Clean, simple GUI	The software is aimed at a wide range of students of differing abilities. The user interface must accommodate for this. A simple to use program with a bloat free design will make the planner easier to use. By only including the essential features of a planner as described in the analysis of the project, and reducing the amount of information displayed on any one screen, will limit the chance of user error. This is because it will help avoid confusion as unnecessary information causes distraction.
Appropriate input methods	Input methods must be chosen carefully so that entering data about a task is as easy as possible. The purpose of this feature is to avoid users from becoming reluctant to use the software as it could be too difficult or annoying to enter tasks. The computational solution must take advantage of this and adapt to improve ease of use.
Input validation	All data entered into the program must be validated. By doing so, the program will be protected from any potential errors from illegal entries. The program must be stable and no vulnerabilities to input methods should be available. Users may enter invalid data unknowingly or even knowingly and the data must not be allowed to be processed. By only allowing valid data, the tasks stored will become structured and well established with all required data needed for the user.

Clearly labelled buttons	This is an extension from the GUI feature. Buttons can be labelled in three main ways by assigning them a, label, an icon or both. To ensure that the user can easily understand the function of a button, the labels must be short and specific and any icons must be clear and visible. The reason for this is because it is important that the user can quickly understand what function a button will do without accidentally modifying their data with unexpected outcomes.

3.2.3 - Describe the Approach to Testing:

I will be using a testing table after each stage of development. This will help me test that the code I have written works as I intend. Each row in the table will be an individual test that I will perform. I will use four columns; the first will contain the test name, what I will be testing for. The second I will write in any test data that I will be using. For example, if I am testing an input into the program, I will write the data that I will use in there. The third column is where I am going to write my expected outcome of the test. Each case I will write as if the test will pass when it is performed. If this outcome is not achieved, then the test would be considered a fail and I will have to develop this further. The final column of the table is where I'll include a screenshot of the program to show the output. I could attach screenshots of the shell too if any errors are produced. This will be used when revisiting the code that caused these errors.

An example of the table that I will be using:

Test	Test Data	Expected Result	Actual Result / Evidence

Test Plan:

Success Criteria	Approach to Testing
The program should offer a sorting system so that the user can view their activities based on importance, time, subject and task. This would be useful as it could easily identify what the user needs to focus on and show which tasks are due soon compared to others.	I will insert a variety of tasks where there will be different data in each task. I will then apply each sorting algorithm and verify that the tasks have been reordered correctly.
Any tasks that have a due date that has expired should be highlighted red so that the user can clearly see which tasks are overdue.	I will set the due date for three tasks so that one task is the current date, the next is the day after and the other task is set to the date before the current. The task that is due in for the previous day should be the only task highlighted in red.
The computer could include an automatically deleting function where it can remove any older tasks automatically so the user can keep track of when tasks have been completed.	Tasks that have expired for a set number of days should could be deleted automatically. To test that this feature worked, I can create a task and set the auto delete function to a single day and then alter the due date to the day before its current due date and it should automatically be removed.
There must be a straightforward method to add tasks into the program. The data that should be collected include the task's name, subject, due date and the type.	I could incorporate this test with my usability testing whereby clients could use my program and rate how easy they thought it was to add tasks in this program. An average of these results could be used to assess the overall review by the users.
For any task that has been recorded, the user should also be able to modify any data related to the task and save these changes.	Insert a task into the program and then attempt to change data in each field. Upon saving these changes, the program should show the task with the updated information.
Tasks should be able to be deleted too.	I will edit this by inserting a task and then delete it afterwards. If the task is removed, the criteria has been met and the feature works.
The program should act as a student's planner; therefore, there should be multiple aspects to this program. I will add in features where the students can write their homework, coursework deadlines and exam dates. As well as this, I will include a data bank where information relevant to the school can be stored, for example: a school map, term dates, contact details, website links, etc.	Add a task for each task type to ensure that the different pages load the correct tasks. To verify that the school data loads correctly, I will load each data page and assert that they load correctly.
The software must be presented clearly so that it is easy to use. It should be presented in a logical manner where widgets are placed with care and appropriately. The design should be aesthetically pleasing.	I cannot test this myself as the appearance of the program and its design is an opinion of the observer. Therefore, I will use feedback from clients.

Test Data:

Test	Test Data	Justification
Add a task with valid details.	Differentiation Maths 12/12/2016 Homework	The purpose of this test is to verify that a task created with valid details should be saved. The planner must be able to save any task with details that pass all validation (such as the example given).
Add a task without a name.	 Maths 12/12/2016 Homework	This will test the input validation on the text entry for the task name. This will verify that an incorrect name will not be allowed to save. By doing so, this will protect the program from errors later on when it would attempt to load an empty piece of data.
Add a task without a subject.	Differentiation 12/12/2016 Homework	A test very similar to the previous where an invalid subject entry could cause potential crashes in the future. This must be tested to ensure that this cannot happened and keep the program running bug free.
Add a task without a due date.	Differentiation Maths Homework	This software will rely on the due date given as all tasks will be sorted with the tasks due soonest ordered first in the list. If the due date is not available, the task cannot be sorted and an error would occur attempting to sort a null variable. This test will confirm that this bug will not occur if it successful.
Add a task with an invalid date.	Differentiation Maths 31/31/2016 Homework	Invalid dates should not be allowed in any software. If this data is stored in a database table under the date data type, then an illegal date would cause the database to produce errors. The errors would result in the task not being saved and potentially lose any data that was entered by the user for the current task. To eliminate any chance of date related failures, the validation must be tested.
Delete a task.	N/A	Deleting a task is an important function within this program. There is no test data for this test as it will be simply removing a task, not entering any data.

3.3 - DEVELOPING THE SOLUTION

3.3.1 - Iterative Development Process:

During the development of this program, I will be creating many variables. Therefore, I will use appropriate names for each and every one. This is because I will need to keep track of them and monitor their changes throughout the running of the program. Assigning names that describe what the variable will hold will make the process easier and will not be likely to confuse them.

Program Setup:

The first thing I did when writing the program was creating a class for the whole program to run in. The reason I chose to do this was so that every variable or data in widgets could be easily collected and manipulated globally. This will reduce any errors when attempting to retrieve inputs or storing them in different functions. Every line of code written in line 2 and below (with at least a single indent) will be inside of this class.

```
1.  class StudentOrganiser(TK):
2.  
3.  interface = StudentOrganiser()
4.  interface.mainloop()
```

Before I added anything to this class, I had started to import some modules that I'll need for this to work. As I am using tkinter, a GUI widget set for Python, I must import it. As well as tkinter, I imported a few more modules. When adding text to the screen, I will be using different fonts and sizes as a part of my design, therefore I'll need the module, tkFont. tkMessageBox allows me to open a pop up window to alert the user as well as ask the user any questions as part of this pop up. This will be very useful for input validation later on, so I have imported it now as I plan to use this later on in the program. Finally, sqlite3 is a module that I will use to manage a database. I will be using a database to store all the tasks and the information about them and sqlite3 will do everything necessary with a database for my program to function.

```
from Tkinter import *
import tkFont
import tkMessageBox
import sqlite3
```

Once I had created the class and imported some modules, I needed to add the function to initialize the whole GUI environment, the window that the use will eventually interact with. There is a syntax that should be followed for this to work correctly, which can be seen in lines 1 and 2. The purpose of this function is to set the window up and declare all variables that will be used later.

Line 4 will set the title of the window to "Student Organiser." This is the name of my project and is a suitable name for the program to be. From lines 5 to 32, I imported 32 pictures that will be used as icons for buttons. I have assigned each button a variable so that I can use this shorter name when setting up buttons in the program.

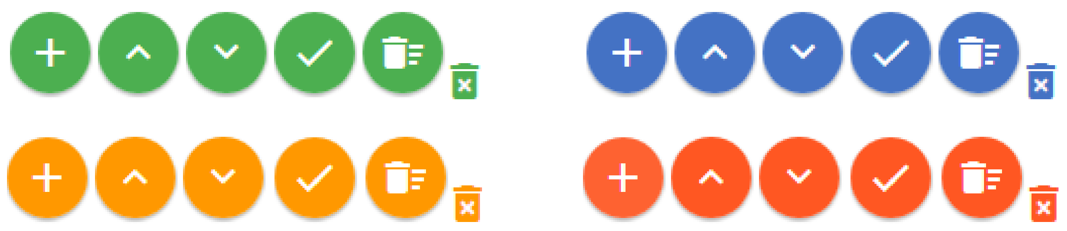
```
1.  def __init__(self):
2.      Tk.__init__(self)
3.  
4.      self.title(" Student Organiser")
5.      self.greenAdd = PhotoImage(file="buttons/green_add.gif")
6.      self.blueAdd = PhotoImage(file="buttons/blue_add.gif")
7.      self.yellowAdd = PhotoImage(file="buttons/yellow_add.gif")
8.      self.redAdd = PhotoImage(file="buttons/red_add.gif")
9.  
10.     self.greenAdd = PhotoImage(file="buttons/green_up.gif")
```

```
    self.blueAdd = PhotoImage(file="buttons/blue_up.gif")
    self.yellowAdd = PhotoImage(file="buttons/yellow_up.gif")
    self.redAdd = PhotoImage(file="buttons/red_up.gif")

    self.greenAdd = PhotoImage(file="buttons/green_down.gif")
    self.blueAdd = PhotoImage(file="buttons/blue_down.gif")
    self.yellowAdd = PhotoImage(file="buttons/yellow_down.gif")
    self.redAdd = PhotoImage(file="buttons/red_down.gif")

    self.greenAdd = PhotoImage(file="buttons/green_done.gif")
    self.blueAdd = PhotoImage(file="buttons/blue_done.gif")
    self.yellowAdd = PhotoImage(file="buttons/yellow_done.gif")
    self.redAdd = PhotoImage(file="buttons/red_done.gif")

    self.greenAdd = PhotoImage(file="buttons/green_delete.gif")
    self.blueAdd = PhotoImage(file="buttons/blue_delete.gif")
    self.yellowAdd = PhotoImage(file="buttons/yellow_delete.gif")
    self.redAdd = PhotoImage(file="buttons/red_delete.gif")

    self.greenAdd = PhotoImage(file="buttons/green_remove.gif")
    self.blueAdd = PhotoImage(file="buttons/blue_remove.gif")
    self.yellowAdd = PhotoImage(file="buttons/yellow_remove.gif")
    self.redAdd = PhotoImage(file="buttons/red_remove.gif")
```

I had created six different icons, each in 4 colours: green, blue, yellow and red. The four colours were used to differentiate between selected tabs as I had planned for each tab to have a unique colour. Whereby the overview tab would be green, homework would be blue, coursework is yellow and exams are red. The buttons I had created all serve a unique purpose in the program. Their purposes are to open the 'add task' page, to scroll up a page, to scroll down a page, to add a task, to discard a task in the 'add task' page and to remove a task.

To create the buttons, I used Google's Material Design. I used their icon pack as the main image inside each icon. I chose these icons as they looked great and the whole set is open source. Another reason was that they're all very high quality. Once I had chosen each icon, I used Material Design colours as the background colours of the icons. I applied the theme of the FAB button with a coloured, circular design featuring a white central icon. I had decided to use Material Design colours because they look really good and there are different pre-defined shades for each colour; making it very easy for me to use different variations of each colour for the four different tabs.

The buttons that I had created:



After defining my button images, I then created variables for each tab that would state which page was in view. This value will be a number where ‘0’ is the first page that loads, it may not contain any tasks but the current page will always start at ‘0.’ If there were multiple pages, this value will change higher or lower but I will explain the use of this variable later, for now, it is important that they are set to ‘0’ when the program loads so that it will always show the first page for each tab. The following are the variables that I created and set:

```
self.ovpage=0
self.hwpage=0
self.cwpage=0
self.expage=0
```

From my initial ideas, I will be creating a window that will be non-resizable. This is so that the program will appear the same on every display. Additionally, a static program would be much easier to create. A dynamically sizing program would be much harder for me as I am not using any scroll bars. Due to this, the paging system would have to adapt and would take a lot of time for me to achieve a functioning program.

The size for the canvas had been calculated. I set each tab to a width that all text for every tab would fit comfortably inside. I then allowed for a fixed gap between each tab and finally a 30-pixel margin between the inner left and right edge of the window boarders.

The code that sets the window to a fixed size and creates a canvas to suit is:

```
self.resizable(width=False, height=False)

self.canvas = Canvas(self, width=575, height=380, bg="#FFFFFF")
self.canvas.pack()
```

As I will be using a database to store all of the data for every task, I will first need to create it. However, I can’t go ahead and create this table every time that this program is used as it may overwrite the older table or crash due to their already being a database with the same name. Therefore, I have ensured that there will be a presence check to verify if a file called, ‘tasks.db’ is present, else it will go ahead and create a new, empty table with the correct columns and data formatting. The database will be created as follows:

Task ID	Primary Key
Task	Text
Subject	Text
Date	Date
Tab	Text

```
databasePresent = os.path.isfile("./tasks.db")
if databasePresent == False:
    conn = sqlite3.connect('tasks.db')
    conn.execute('''CREATE TABLE TASKS
            (ID INT PRIMARY KEY     NOT NULL,
            TASK            TEXT    NOT NULL,
            SUBJECT         TEXT    NOT NULL,
            DATE            DATE    NOT NULL,
            TAB             TEXT    NOT NULL);''')
    conn.close()
```

Menu Bar:

Once all of these instructions have completed, two functions will be called, the first is the menu bar and the second is the overview page. The menu bar function will set up the menu bar and will load for the entire program. Once loaded, there is no need to call it again. The menu bar is where I have decided to store buttons that I thought should be able to be accessed anywhere in the program without taking any space within the main body of the program's window. The menu bar will have three main buttons: file, help and sort. File will have basic shortcuts that can load the home page (the overview page,) add a task and exit. The help option will contain options to view the help guide and the about window for information about this software. Finally, the sort button will offer a list of sorting filters that can rearrange the data into any of the set methods. These may be useful for the user as they can quickly rearrange all of their tasks using a set rule and easily find tasks. The menu bar function:

```
def menu_bar(self):
    menu_bar = Menu(self)
    afile = Menu(menu_bar, tearoff=0)
    menu_bar.add_cascade(label="File", menu=afile)
    afile.add_command(label="New Task", command=self.add_item)
    afile.add_command(label="Home", command=self.setup_overview)
    afile.add_separator()
    afile.add_command(label="Exit", command=self.close)

    bhelp = Menu(menu_bar, tearoff=0)
    menu_bar.add_cascade(label="Help", menu=bhelp)
    bhelp.add_command(label="About", command=self.donothing)
    bhelp.add_separator()
    bhelp.add_command(label="Help", command=self.donothing)

    csort = Menu(menu_bar, tearoff=0)
    menu_bar.add_cascade(label="Sort", menu=csort)
    csort.add_command(label="Time: Soonest (Default)", command=self.sort_1)
    csort.add_command(label="Time: Oldest", command=self.sort_2)
    csort.add_command(label="Time: Task added", command=self.sort_7)
    csort.add_separator()
    csort.add_command(label="Task: A-Z", command=self.sort_3)
    csort.add_command(label="Task: Z-A", command=self.sort_4)
    csort.add_separator()
    csort.add_command(label="Subject: A-Z", command=self.sort_5)
    csort.add_command(label="Subject: Z-A", command=self.sort_6)

    self.config(menu=menu_bar)
```

Tab Selection:

Once the menu bar has loaded, the first page that the user can interact with will begin to load. The function that is called is named, 'setup_overview.' For efficiency, I have used a system where the selection of a tab will assign the 'self.tab' variable to the selected tab's name. By doing so, I will not have to write four large functions for controlling how each page loads; I can use a single variable and use this value when handling the data for loading the correct results and colour theme. It also allows me to split up my code and change the flow of the program easily. I have written these four functions for when the respective tab had been selected by the user:

```python
def setup_overview(self):
    self.tab = "Overview"
    self.setup_data()

def setup_homework(self):
    self.tab = "Homework"
    self.setup_data()

def setup_coursework(self):
    self.tab = "Coursework"
    self.setup_data()

def setup_exams(self):
    self.tab = "Exam"
    self.setup_data()
```

When any of these tabs have been selected and the 'self.tab' variable had been assigned a name, they all load the same function, 'setup_data.' This function has many purposes; it splits the database to show only relevant tasks for the tab, it sets up the page to hold and show the tasks and loads the correct theme. The first part of the function is assigning variables based on the selected tab. These variables will be used throughout the function and also adds to my efficiency of code.

```python
def setup_data(self):
    if self.tab == "Overview":
        self.colour = "#4caf50"
        self.rowColour = "#c8e6c9"
        self.add = self.greenAdd
        self.delete = self.greenDelete
        self.remove = self.greenRemove

    if self.tab == "Homework":
        self.colour = "#4472C4"
        self.rowColour = "#bbdefb"
        self.add = self.blueAdd
        self.delete = self.blueDelete
        self.remove = self.blueRemove

    if self.tab == "Coursework":
        self.colour = "#ff9800"
        self.rowColour = "#ffe0b2"
        self.add = self.yellowAdd
        self.delete = self.yellowDelete
        self.remove = self.yellowRemove

    if self.tab == "Exam":
        self.colour = "#ff5722"
        self.rowColour = "#ffcdd2"
        self.add = self.redAdd
        self.delete = self.redDelete
        self.remove = self.redRemove
```

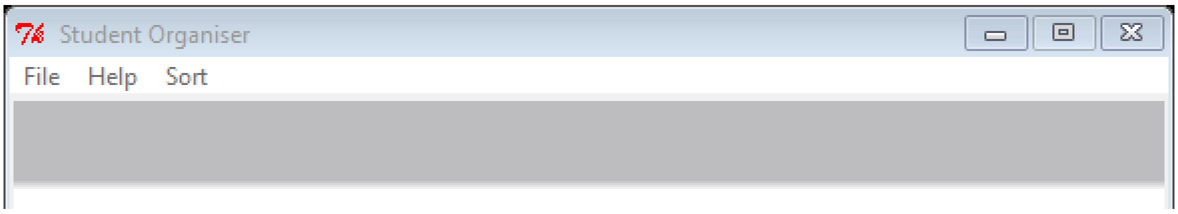
The first two variables in each function are used when loading the theme of the window. These are two hexadecimal colours. I had chosen these colours from Google's Material Design as they do look very nice and there is a range of different colours. As well as this, each colour has a set of varying shades, which is very useful as I can use the same shades of different colours to ensure a matching theme. The first colour corresponds to the colour of the background colour behind the tabs; the second controls the colour of the border for each row containing a task. The other three variables control the colour of the buttons to use. As previously stated, I have created 32 buttons, which include the four colours. Here, I am assigning the correct coloured button for the page and tab.

Page Setup:

Once the tab specific variables have been set, the canvas is cleared to ensure an empty window before the title bar is created.

```
    self.canvas.delete(ALL)
    self.draw_title_bar()
```

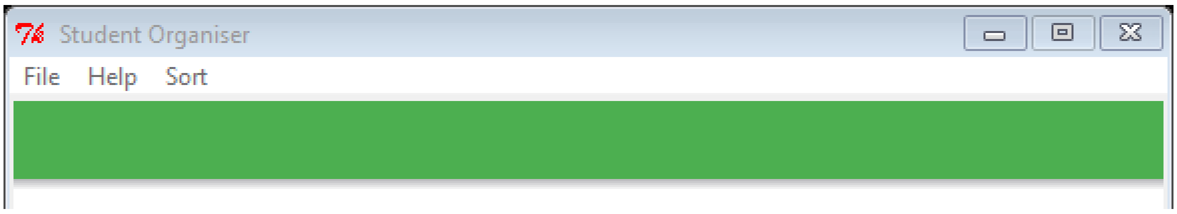
The title bar function only adds five rectangles to the canvas. Each rectangle has a varying shade of grey and aligned diagonally to one another. This forms a shadow effect, which will eventually become the shadow to the tab background.



The order in which this had been created was important. The most recent widget to be placed on the canvas will overlap any existing widget in any position where they meet. Therefore, the shadow will be created first and then I will add in the correct tab colour. For this preview, I will use the 'Overview' tab as this will act as the home screen.

The code that adds this coloured rectangle on top:

```
    self.canvas.create_rectangle(0, 40, 600, 0, fill=self.colour, 
outline=self.colour)
```

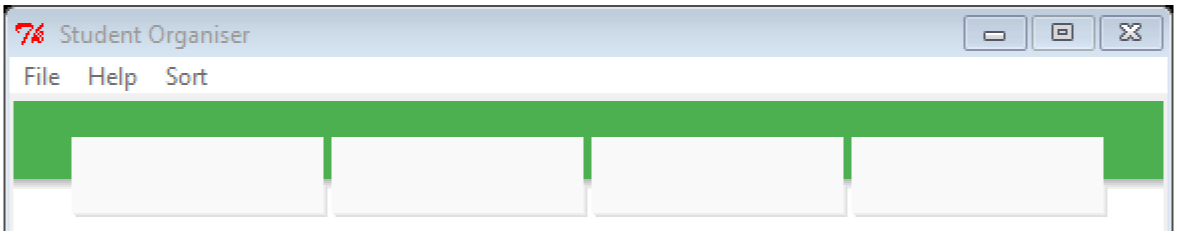


The function, 'self.draw_tabs' is then called to add the four tabs across the top of the window.

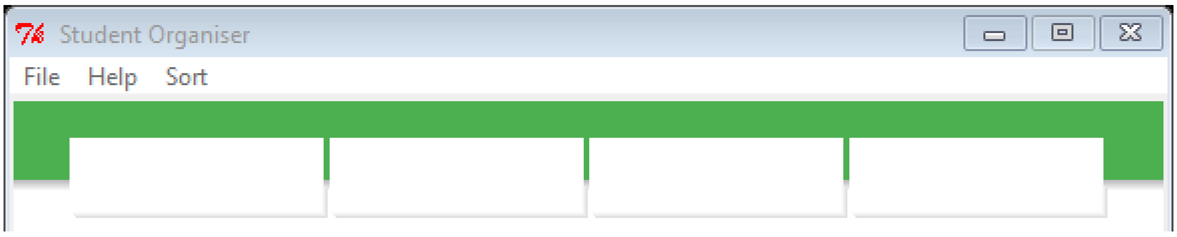
```
def draw_tabs(self):
    # TAB 1: Overview
    self.canvas.create_rectangle(33, 45, 158, 59, fill="#e0e0e0", 
outline="#e0e0e0")
    self.canvas.create_rectangle(32, 44, 157, 58, fill="#ededed", 
outline="#ededed")
    self.canvas.create_rectangle(31, 20, 156, 57, fill="#f9f9f9", 
outline="#f9f9f9")
    self.canvas.create_rectangle(30, 20, 155, 56, fill="#FFFFFF", 
outline="#FFFFFF")

    # TAB 2: Homework
    self.canvas.create_rectangle(163, 45, 288, 59, fill="#e0e0e0", 
outline="#e0e0e0")
    self.canvas.create_rectangle(162, 44, 287, 58, fill="#ededed", 
outline="#ededed")
    self.canvas.create_rectangle(161, 20, 286, 57, fill="#f9f9f9", 
outline="#f9f9f9")
```

```
    self.canvas.create_rectangle(160, 20, 285, 56, fill="#FFFFFF", outline="#FFFFFF")

    # TAB 3: Coursework
    self.canvas.create_rectangle(293, 45, 418, 59, fill="#e0e0e0", outline="#e0e0e0")
    self.canvas.create_rectangle(292, 44, 417, 58, fill="#ededed", outline="#ededed")
    self.canvas.create_rectangle(291, 20, 416, 57, fill="#f9f9f9", outline="#f9f9f9")
    self.canvas.create_rectangle(290, 20, 415, 56, fill="#FFFFFF", outline="#FFFFFF")

    # TAB 4: Exams
    self.canvas.create_rectangle(423, 45, 548, 59, fill="#e0e0e0", outline="#e0e0e0")
    self.canvas.create_rectangle(422, 44, 547, 58, fill="#ededed", outline="#ededed")
    self.canvas.create_rectangle(421, 20, 546, 57, fill="#f9f9f9", outline="#f9f9f9")
    self.canvas.create_rectangle(420, 20, 545, 56, fill="#FFFFFF", outline="#FFFFFF")

    # BUTTON 1: Overview
    button_overview = Button(self, text="Overview", font=self.tab_font, command=self.setup_overview, anchor = W)
    button_overview.configure(border=0, relief=FLAT, fg="#4caf50", activeforeground="#4caf50", bg="white", activebackground="white")
    button_overview_window = self.canvas.create_window(46, 20, anchor=NW, window=button_overview)

    # BUTTON 2: Homework
    button_homework = Button(self, text="Homework", font=self.tab_font, command=self.setup_homework, anchor = W)
    button_homework.configure(border=0, relief=FLAT, fg="#4472C4", activeforeground="#4472C4", bg="white", activebackground="white")
    button_homework_window = self.canvas.create_window(173, 20, anchor=NW, window=button_homework)

    # BUTTON 3: Coursework
    button_coursework = Button(self, text="Coursework", font=self.tab_font, command=self.setup_coursework, anchor = W)
    button_coursework.configure(border=0, relief=FLAT, fg="#ff9800", activeforeground="#ff9800", bg="white", activebackground="white")
    button_coursework_window = self.canvas.create_window(298, 20, anchor=NW, window=button_coursework)

    # BUTTON 4: Exams
    button_exams = Button(self, text="Exams", font=self.tab_font, command=self.setup_exams, anchor = W)
    button_exams.configure(border=0, relief=FLAT, fg="#ff5722", activeforeground="#ff5722", bg="white", activebackground="white")
    button_exams_window = self.canvas.create_window(449, 20, anchor=NW, window=button_exams)
```

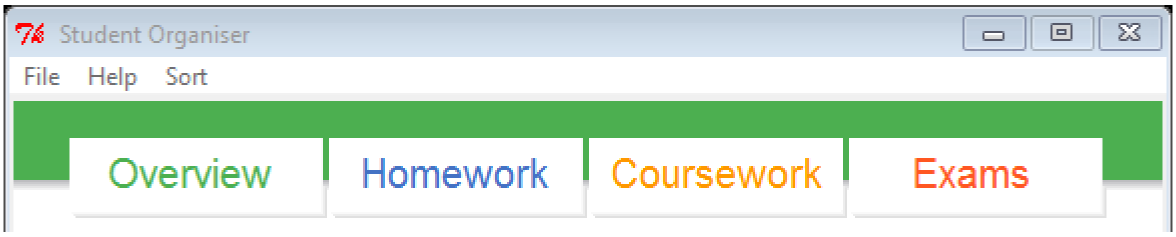
The first half of this function sets up the background for each tab. It adds in 16 rectangles in total. Out of these, 12 are grey rectangles, positioned to form a shadow effect behind each tab. They have all been carefully positioned to ensure an even gap between each tab and the inner margins of the main window.



Once the shadows are in place, white rectangles will be added on top. These white rectangles will act as the background of the button. I chose white as it is a neutral colour but does not fade in with the background of the title bar.



The second half of this function is responsible for adding in the buttons that will select the tab. I had chosen to use buttons as I can easily adjust the size of the button (to fill the entire tab) and easily add coloured text. The main reason is that it was a method of calling a function within my program, which made this widget a necessity. Each button, created identically whereby only the position, label, text colour and command change. The buttons will contain the name of the tab that they represent, as well as the tab's colour. When clicked, they all have different commands as they all have different purposes, loading different pages.

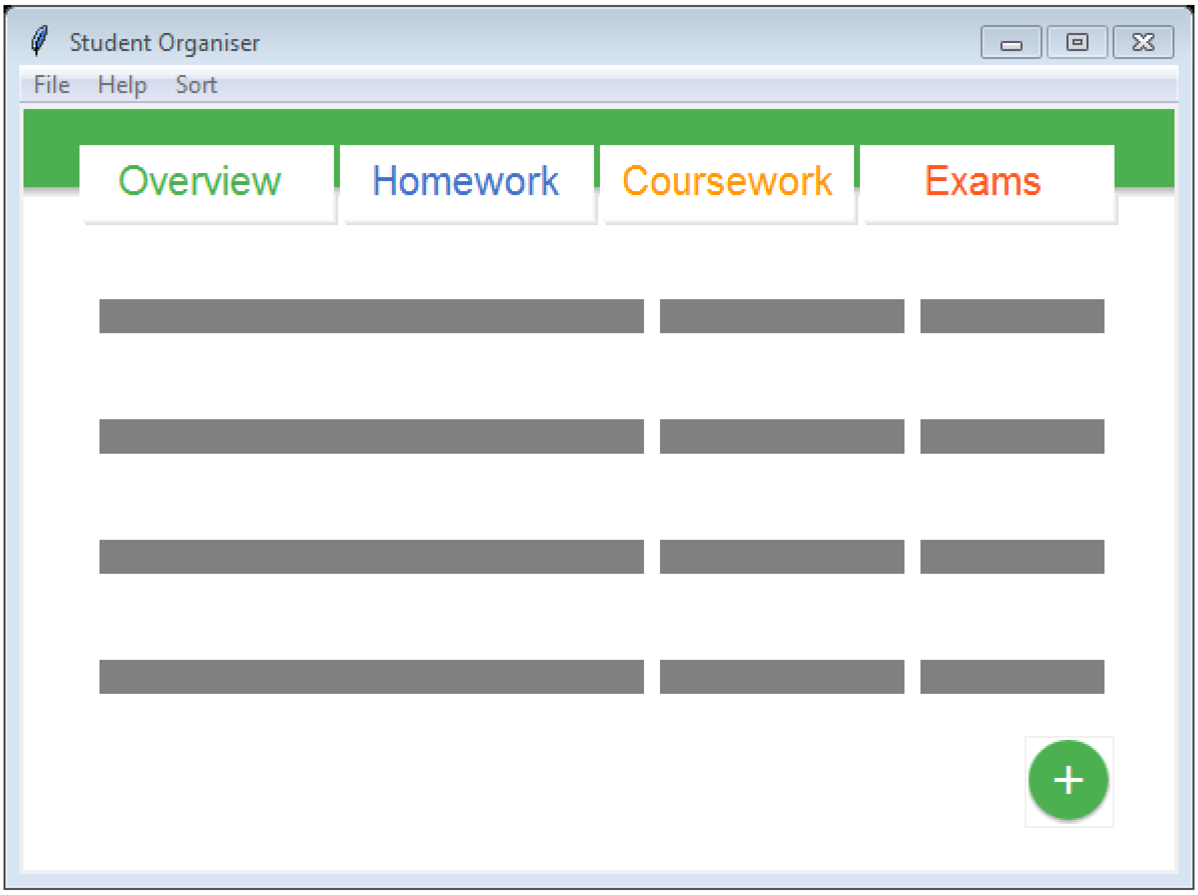


When the tab bar has been set up, the button to add tasks is inserted. This button is to be positioned in the lower right hand corner. I will leave a margin around this button so that it has an even space between the window edges. For the user to add their tasks to the database, they will use this button. The code to create this button:

```
button_add = Button(self, image=self.add, command=self.add_item, anchor = W)
button_add.configure(border=0, relief = FLAT)
button_add_window = self.canvas.create_window(510, 315, anchor=NW,
window=button_add)
```

The next widgets to add to the canvas is twelve entry boxes. These will be used to input the task data so that they are presented clearly to the user. I initially planned to use text and place that on the canvas. However, the problem with that solution was that, when aligning the text, the coordinates would map the centre of the text. This will mean that, depending on the task name, the tasks would all be out of line and would not look great. To fix this, I used entry boxes. I can still write to the boxes so that the tasks will be shown to the user and every task can be aligned to the left. By setting the width of each entry box and the position, I can create a table like structure.

I have adjusted the background colour of each entry box so that each entry box can be seen in the documentation, however, in the code, there will be no boarder as per my design.



This image shows four rows with three columns. Each row will contain a task, where the first column is the task name, the second will be the subject and the third should be the due date. The code that I had written to achieve this:

```
# Row 1
self.name_1 = Entry(self,width=45)
self.name_1.configure(border=0, relief=FLAT, bg="white")
name_1_window = self.canvas.create_window(40, 97, anchor=NW, window=self.name_1)
self.subject_1 = Entry(self,width=20)
self.subject_1.configure(border=0, relief=FLAT, bg="white")
subject_1_window = self.canvas.create_window(320, 97, anchor=NW,
window=self.subject_1)
self.date_1 = Entry(self,width=15)
self.date_1.configure(border=0, relief=FLAT, bg="white")
date_1_window = self.canvas.create_window(450, 97, anchor=NW, window=self.date_1)

# Row 2
self.name_2 = Entry(self,width=45)
self.name_2.configure(border=0, relief=FLAT, bg="white")
name_2_window = self.canvas.create_window(40, 157, anchor=NW, window=self.name_2)
self.subject_2 = Entry(self,width=20)
self.subject_2.configure(border=0, relief=FLAT, bg="white")
subject_2_window = self.canvas.create_window(320, 157, anchor=NW,
window=self.subject_2)
```

```
self.date_2 = Entry(self,width=15)
self.date_2.configure(border=0, relief=FLAT, bg="white")
date_2_window = self.canvas.create_window(450, 157, anchor=NW, window=self.date_2)

# Row 3
self.name_3 = Entry(self,width=45)
self.name_3.configure(border=0, relief=FLAT, bg="white")
name_3_window = self.canvas.create_window(40, 217, anchor=NW, window=self.name_3)
self.subject_3 = Entry(self,width=20)
self.subject_3.configure(border=0, relief=FLAT, bg="white")
subject_3_window = self.canvas.create_window(320, 217, anchor=NW,
window=self.subject_3)
self.date_3 = Entry(self,width=15)
self.date_3.configure(border=0, relief=FLAT, bg="white")
date_3_window = self.canvas.create_window(450, 217, anchor=NW, window=self.date_3)

# Row 4
self.name_4 = Entry(self,width=45)
self.name_4.configure(border=0, relief=FLAT, bg="white")
name_4_window = self.canvas.create_window(40, 277, anchor=NW, window=self.name_4)
self.subject_4 = Entry(self,width=20)
self.subject_4.configure(border=0, relief=FLAT, bg="white")
subject_4_window = self.canvas.create_window(320, 277, anchor=NW,
window=self.subject_4)
self.date_4 = Entry(self,width=15)
self.date_4.configure(border=0, relief=FLAT, bg="white")
date_4_window = self.canvas.create_window(450, 277, anchor=NW, window=self.date_4)
```

Once the entry boxes have been inserted. The contents of each one bust be removed. This is because, when the next page loads (if there are multiple pages) this function will be called. When adding the data, it would add the new task data to the last position in the entry box and therefore, it will just add this to the current data. This is an undesirable outcome with a simple solution, clear the contents of all entry boxes before they are used.

```
# Clear contents of every row
self.name_1.delete(0, END)
self.subject_1.delete(0, END)
self.date_1.delete(0, END)
self.name_2.delete(0, END)
self.subject_2.delete(0, END)
self.date_2.delete(0, END)
self.name_3.delete(0, END)
self.subject_3.delete(0, END)
self.date_3.delete(0, END)
self.name_4.delete(0, END)
self.subject_4.delete(0, END)
self.date_4.delete(0, END)
```

Database – Reading Tasks:

Now, there is an aligned grid of empty entry boxes ready to receive data. The next stage is to take all of the data from the database and split it up into different arrays for use later on. I need to split the data up into an array per task. Then each task will be added to a larger, two dimensional array of every task. When creating the arrays, multiple steps occur to ensure that the data is converted into the correct format that I had defined.

In the database, tasks are written as:	ID, Name, Subject, Date YYYY/MM/DD, TAB
I will convert the tasks to the format:	Name, Subject, Date DD/MM/YYYY, TAB, ID

To start, I open a connection to the database 'tasks.db.' Once a connection had been established, I used a SQL query to select every task from the database for the current tab. For example, for the overview page, every task is selected. However, for any of the other tasks, the selected tasks are based on the values in the 'TAB' column.

To do this, I used the following code with the SQL statements:

```python
conn = sqlite3.connect('tasks.db')
self.tasks = []
if self.tab == "Overview":
    cursor = conn.execute("SELECT * FROM tasks ORDER BY DATE ASC")
else:
    cursor = conn.execute("SELECT * FROM tasks WHERE TAB='"+str(self.tab)+"' ORDER BY DATE ASC")
```

I have chosen to pre-sort these tasks. When each row in the database had been read from the database, it is sorted by date ascending. I chose to have the data sorted in this way as it shows the user immediately which tasks are due the soonest. With the tasks sorted in chronological order, they are then split up to form the two-dimensional array.

```python
for row in cursor:
    single = []
    single.append(row[1])
    single.append(row[2])

    # Reverse the date format (YYYY/MM/DD => DD/MM/YYYY)
    dateData = (row[3].strip().split('/'))
    dateDay = dateData[2]
    dateMonth = dateData[1]
    dateYear = dateData[0]
    dateNew = dateDay + "/" + dateMonth + "/" + dateYear

    single.append(dateNew)
    single.append(row[4])
    single.append(row[0])
    self.tasks.append(single)
conn.close()
```

I chose to create a two-dimensional array because it would be very easy to handle the data later on when splitting them into rows. First, I create an empty array called ‘single.’ This will contain the data for a single task. The, for every task in the selected data, I perform the following process with the data. The values for the task name and subject are appended to the ‘single’ array. Before adding the date this this array, I will change the format in which it is written. For storing a date in a database record, the format is as follows: YYYY/MM/DD. As the ‘single’ array contains the data that will be presented to the user, it should be presented in the conventional format that the user would be used to, DD/MM/YYYY.

To convert the date, I took the third item from the task, the date, and split the content up at every ‘/’. This will create a new array called, ‘dateData’ where the first item is the year, the second is the month and the third is the day. The final step is to concatenate these values in the correct order whilst adding in the ‘/’ where appropriate. This new date is added the array followed by the tab name and the ID. When this has completed, the whole array is added to the tasks array and the database connection is closed.

Page Sorting:

Once the tasks array has been created, the page system should be created. This will sort each task by date soonest and allow a maximum of four tasks per page. Every task in the selected tab should appear once in a sorted order.

To start, I calculated the number of tasks in the selected section. This number is then divided up to find how many full pages of four that there will be. This will be a value that stores the number of pages, the remainder will be any additional tasks left over. Each task in the data set will be split into these sets where there will be four tasks per set and the remainder will match with any tasks not in a full set. As there are now tasks stored within a set array, stored within a tasks array, I can use this to output the correct data depending on the current page or tab.

The code that I wrote to create this sorting algorithm:

```python
# Get number of tasks and create array of them
self.tasknumber = 0
for i in self.tasks:
    self.tasknumber += 1

# Get number of pages
self.setsremainder = self.tasknumber % 4
if self.tasknumber == 0:
    self.sets = 0
elif self.tasknumber < 5:
    self.sets = 1
elif self.setsremainder > 0:
    self.sets = self.tasknumber // 4
    self.sets += 1
elif self.setsremainder == 0:
    self.sets = self.tasknumber // 4

# Get sets of data
self.progress=[]
for i in range(self.sets):
    self.mintask = (i) * 4
    self.maxtask = self.mintask + 3
    self.progress.append(self.mintask)
    self.progress.append(self.maxtask)
```

Development Review:

At this point, the program has a functioning database capable of inserting data. The data that can be stored includes: the task’s ID, name, subject, due date in the form, YYYY/MM/DD, and the task type. Tasks stored in the database can be read from and ordered to form pages. This process involves sorting the tasks by their due date, followed by splitting up the tasks in groups of four, where a remainder can become a set where applicable.

Although none of the measurable success criteria have been met at this point, the program is able to handle the main feature, storing and splitting tasks.

Load Rows:

Once the data has been split into the appropriate sets, I then created a function that will output this data into the rows in the program. Following on from the tab selection, I used the same technique at the start of the function. Depending on the selected tab, the page number (current page of the data in the selected tab), the colour of the up and down button will all be set here.

```python
def load_rows(self):
    if self.tab == "Overview":
        self.page = self.ovpage
        self.up = self.greenUp
        self.down = self.greenDown

    if self.tab == "Homework":
        self.page = self.hwpage
        self.up = self.blueUp
        self.down = self.blueDown

    if self.tab == "Coursework":
        self.page = self.cwpage
        self.up = self.yellowUp
        self.down = self.yellowDown

    if self.tab == "Exam":
        self.page = self.expage
        self.up = self.redUp
        self.down = self.redDown
```

Once every tab specific variable has been assigned, I first set the button controller up. The buttons should only work if they need to. As the role of the button is to adjust the page number and progress, there can be crashes or ‘out of range’ errors if the user would advance too far and there are not enough entries in the arrays to load. To fix this, I decided it would be a suitable idea to disable any buttons that cannot be used.

The criteria that I set out for these buttons:

- If the current page is the first possible, disable the up button
- If the current page is the last possible, disable the down button
- If the user changes page at any point where the above situations have not been met, both buttons should be enabled.
- If an enabled button has been clicked, adjust the page number accordingly.

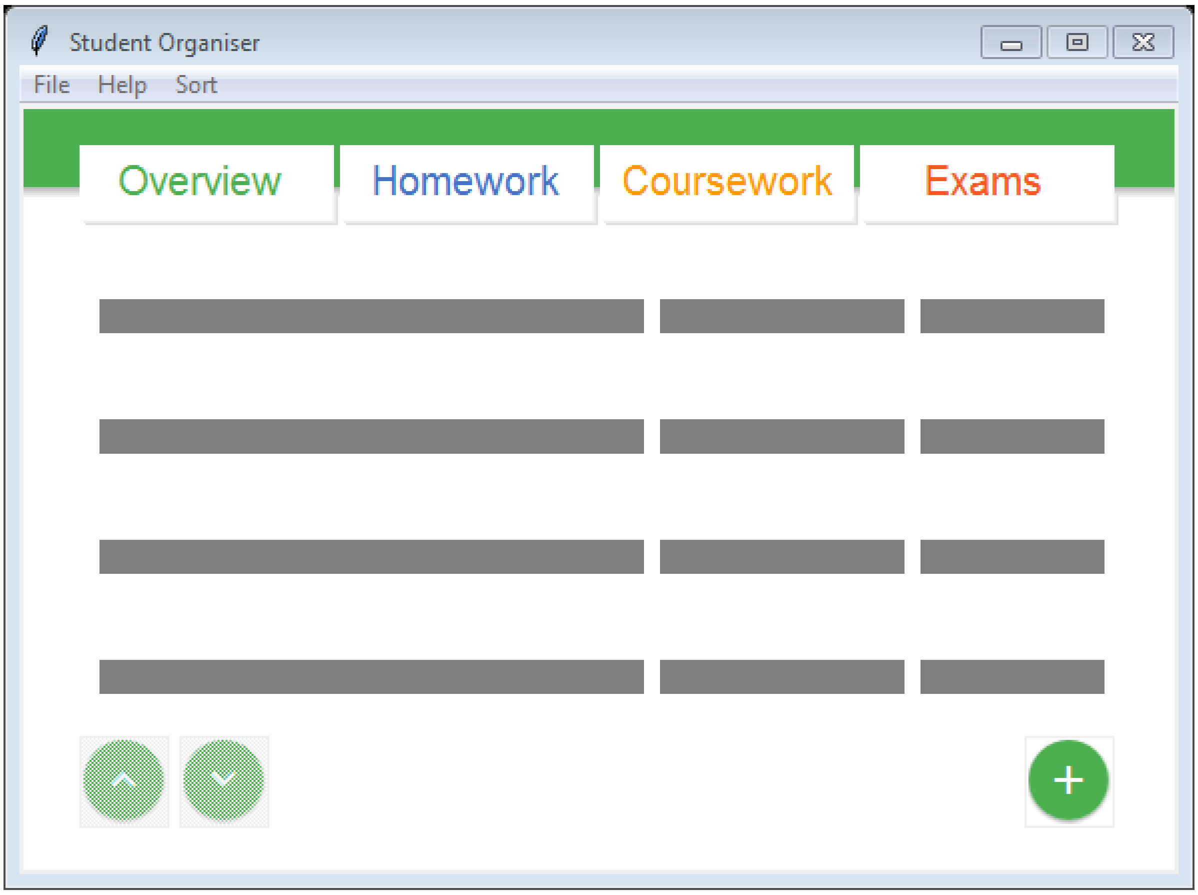
I wrote this with a series of ‘if statements’ which check the page number and criteria:

```python
# Change state of up/down buttons
if self.page == 0:
    self.up_state = "disabled"
    self.down_state = "active"
if self.page > 0:
    self.up_state = "active"
    self.down_state = "active"
if self.page == ((self.sets)*2)-2:
    self.up_state = "normal"
    self.down_state = "disabled"
if self.sets == 1:
    self.up_state = "disabled"
    self.down_state = "disabled"
if self.tasknumber == 0:
    self.up_state = "disabled"
    self.down_state = "disabled"
```

To prepare the window for inserting data, I will delete the content of every row before any data is loaded into the rows. This will ensure that any new data entered into the entry boxes will be the only data present. To delete every row, I just used the 'delete()' command for every row:

```
# Clear contents of every row
self.name_1.delete(0, END)
self.subject_1.delete(0, END)
self.date_1.delete(0, END)
self.name_2.delete(0, END)
self.subject_2.delete(0, END)
self.date_2.delete(0, END)
self.name_3.delete(0, END)
self.subject_3.delete(0, END)
self.date_3.delete(0, END)
self.name_4.delete(0, END)
self.subject_4.delete(0, END)
self.date_4.delete(0, END)
```

Before any data can be added, I add the up / down buttons. These will be the primary method for the user to navigate through each page of tasks.



As there is currently no data in this database, both buttons are disabled and the rows are empty. As before, I have filled the entry boxes in a grey colour to show their presence for the documentation.

```
# Add up/down buttons
button_up = Button(self, image=self.up, command=self.page_up, anchor = W)
button_up.configure(border=0, relief = FLAT, state=self.up_state)
button_up_window = self.canvas.create_window(30, 315, anchor=NW, window=button_up)

button_down= Button(self, image=self.down, command=self.page_down, anchor = W)
button_down.configure(border=0, relief = FLAT, state=self.down_state)
button_down_window = self.canvas.create_window(80, 315, anchor=NW,
window=button_down)
```

I configured the foreground colour of every entry box to black. This means that the text that will be inserted in to those will be black in font colour. This needed to be defined as the background colour had been changed to white. The purpose of this was to hide them from view so only entry boxes that contain a task will appear.

```
self.name_1.configure(fg="#000000")
self.subject_1.configure(fg="#000000")
self.date_1.configure(fg="#000000")
self.name_2.configure(fg="#000000")
self.subject_2.configure(fg="#000000")
self.date_2.configure(fg="#000000")
self.name_3.configure(fg="#000000")
self.subject_3.configure(fg="#000000")
self.date_3.configure(fg="#000000")
self.name_4.configure(fg="#000000")
self.subject_4.configure(fg="#000000")
self.date_4.configure(fg="#000000")
```

Now that the whole page has been set up, I wrote the code to start handling the data. I started by defining the current date. This is very important as this will define whether a task is overdue. I can use an 'if statement' to declare whether the due date is before the current date, if so, the task is overdue and should be highlighted red, as per my specification.

The code to import the date:

```
todayDate = (time.strftime("%d/%m/%Y"))
currentDate = time.strptime(todayDate, "%d/%m/%Y")
```

To protect against crashes, I checked if the number of tasks is equal to zero. If so, the function should 'pass.' This works as any arrays have not been operated on. As the arrays would be empty, the program would attempt to import data and would receive an 'out of range' error.

```
if self.tasknumber == 0:
    return
```

If this has passed, another decision is made. As it has been discovered that there is at least one item in the array of tasks, the next decision is to check if the current page does not have all four tasks. I chose to split this part up into four sections. Each will handle an increasing number of tasks. If there is not a full set, then there will be a series of 'if statements' to determine how many rows are full.

Depending on how many rows are full, will decide how many rows will be handled. This was a solution that I found to the problem where I would get 'out of range' errors when accessing data in an array that isn't there. It also allows me to create the correct number of row boxes to separate the tasks.

If there are more than 0 rows full, but less than 4, the following two statements will return as true:

```
self.tasknumber<=self.progress[self.page+1]and
self.tasknumber>self.progress[self.page]
```

For example, a page with the following data will return False:
self.progress = [0, 3, 4, 7]
self.tasknumber = 5
self.page = 0

self.tasknumber <= self.progress[self.page+1] and self.tasknumber > self.progress[self.page]
5 <= 3 and 5 > 0
False and True
False

For example, a page with the following data will return True:
self.progress = [0, 3, 4, 7]
self.tasknumber = 5
self.page = 2

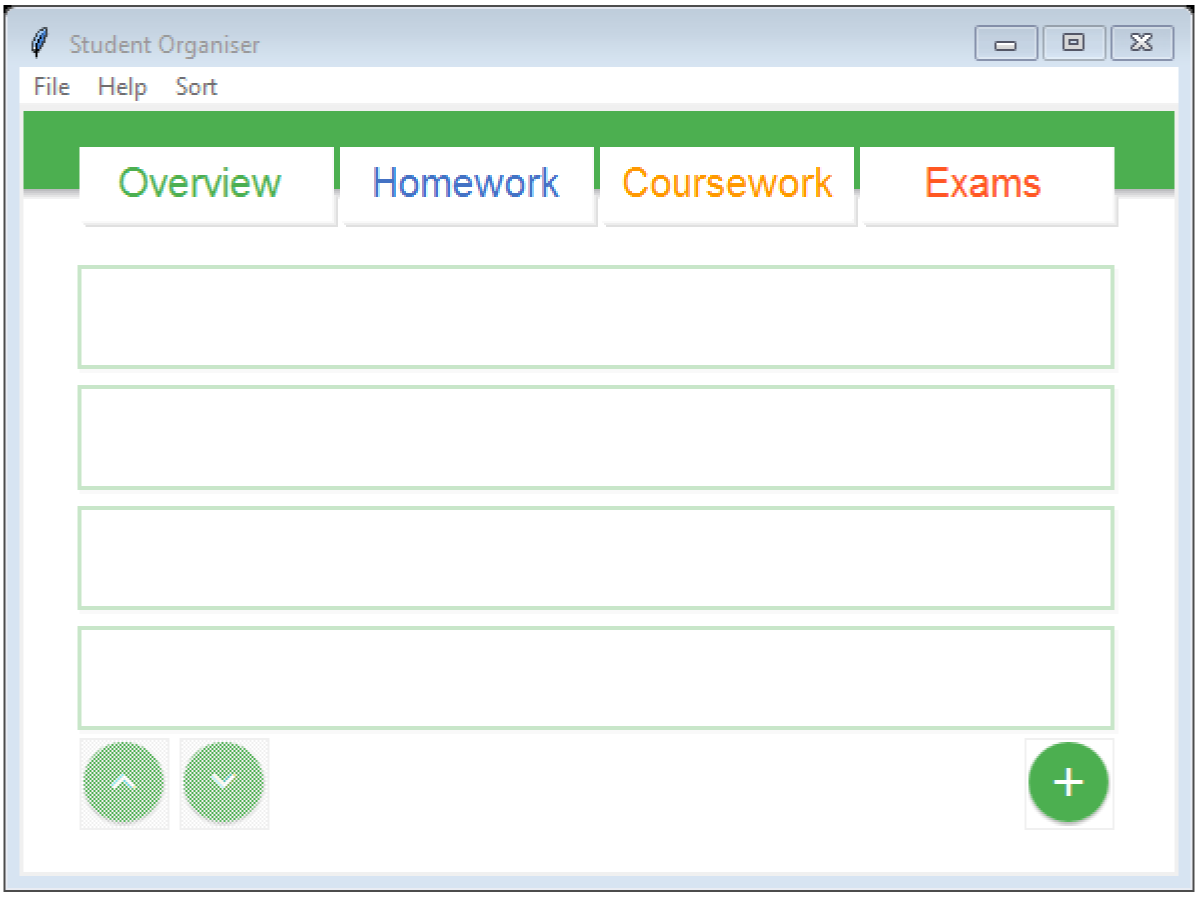
self.tasknumber <= self.progress[self.page+1] and self.tasknumber > self.progress[self.page]
5 <= 7 and 5 > 4
True and True
False

If the outcome is true, then there can be one, two or three tasks on the current page. The next stage is to create the set of data needed to output into the entry boxes. The set will be created within a certain range of tasks.

```
self.set=[]
for i in self.tasks[self.progress[self.page]:self.tasknumber]:
    self.set.append(i)

self.fullRows = self.tasknumber - self.progress[self.page]
```

This works by using the progress array that I created earlier. It uses the current page to determine the range of tasks to load from the tasks array and adds them into the set array. To narrow down the exact the number of full rows, I subtracted the current progress value from the total number of tasks. As this part of the program, will load after all full sets in this tab, if any, have loaded; this will always work and give the correct number of full rows on the last page, if there are not four.

Using the number of full rows, I am able to add design to the page. I will use a rectangle design around each task to clearly show each individual task. This is how the page will look if there is a full set of tasks.



The code below will be used if there is only a single full row:

```
if self.fullRows == 1:
    row1 = []
    for i in self.set[0]:
        row1.append(i)
    self.name_1.insert(INSERT, row1[0])
    self.subject_1.insert(INSERT, row1[1])
    self.date_1.insert(INSERT, row1[2])

    taskDate_1 = time.strptime(row1[2], "%d/%m/%Y")
    taskOverdue_1 = currentDate > taskDate_1

    if taskOverdue_1 == True:
        self.name_1.configure(fg="#f44336")
        self.subject_1.configure(fg="#f44336")
        self.date_1.configure(fg="#f44336")

    button_remove_1 = Button(self, image=self.remove, command=self.delete_row_1,
anchor=W)
    button_remove_1.configure(border=0, relief = FLAT)
    button_remove_1_window = self.canvas.create_window(520, 94, anchor=NW,
window=button_remove_1, tags="rowBox")
    return
```

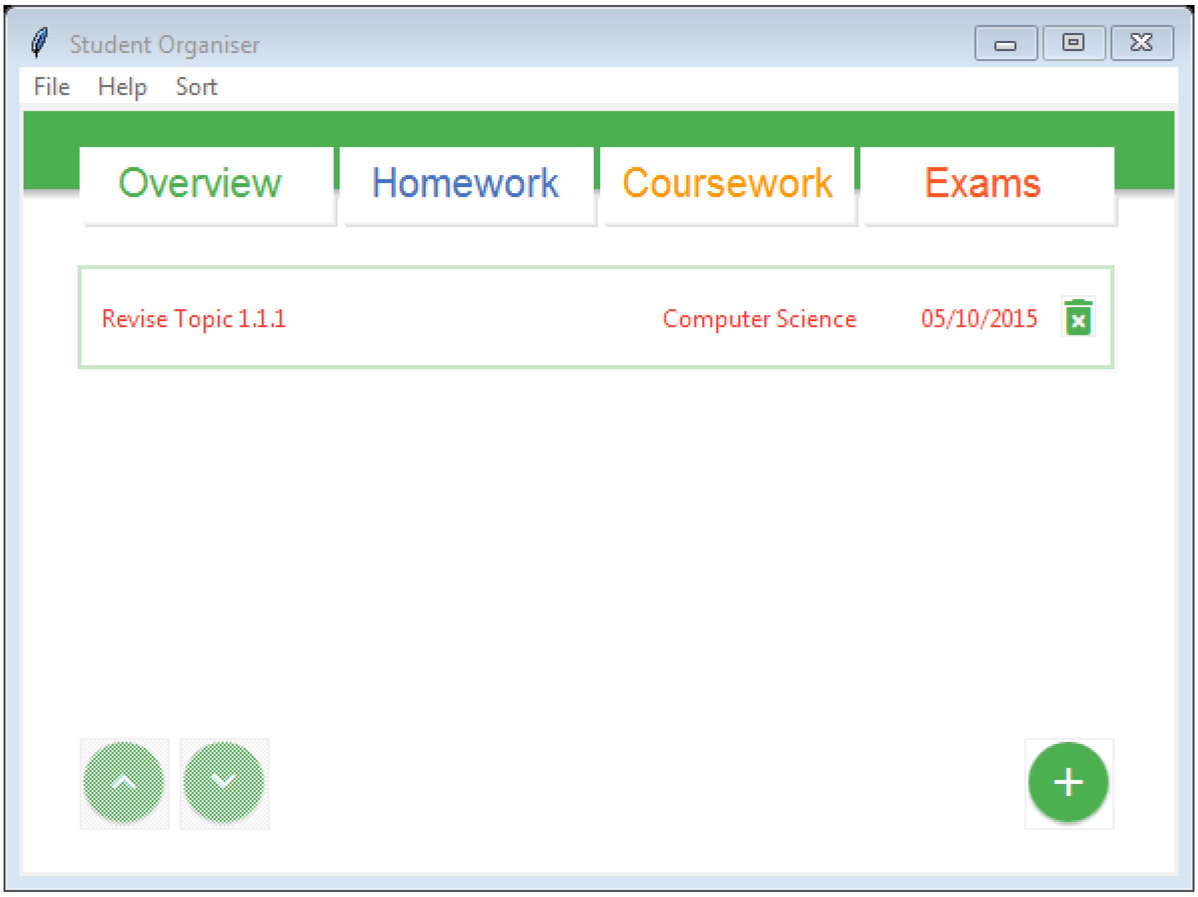
If there is only one task to be displayed on this page, then an empty array called ‘row1’ will be created. The data needed for the first row will be stored in here. For every item in the set, the row array is appended with this item, leaving row1 with every detail about the first task in the set.

This data is then inserted into the appropriate entry boxes where the task name is inserted into the first entry box on the first row, the subject is inserted into the second and the due date is inserted into the last entry box on that row. At this stage, all the information relevant to the user has been outputted to the user in these entry boxes. There is no need to output the ID or the tab name as these values are only used by the program during sorting or choosing the data to present.

To highlight any tasks that are overdue in red, it checks if the current date is greater than the due date. If true, the foreground colour of every entry box for this row will become red; making the text inside that entry box become red.

The final part to this statement is to add the delete button. I have added an individual delete button per row which allows the user to delete that row of data. This will remove the task from the database, load every row again and, if necessary, change the current page backwards if there are no tasks on this page. If, however, this is the first page, the page count will remain at 0 and no data will be shown.

If I were to add, Revise topic 1.1.1, for Computing as a homework task, it would appear:

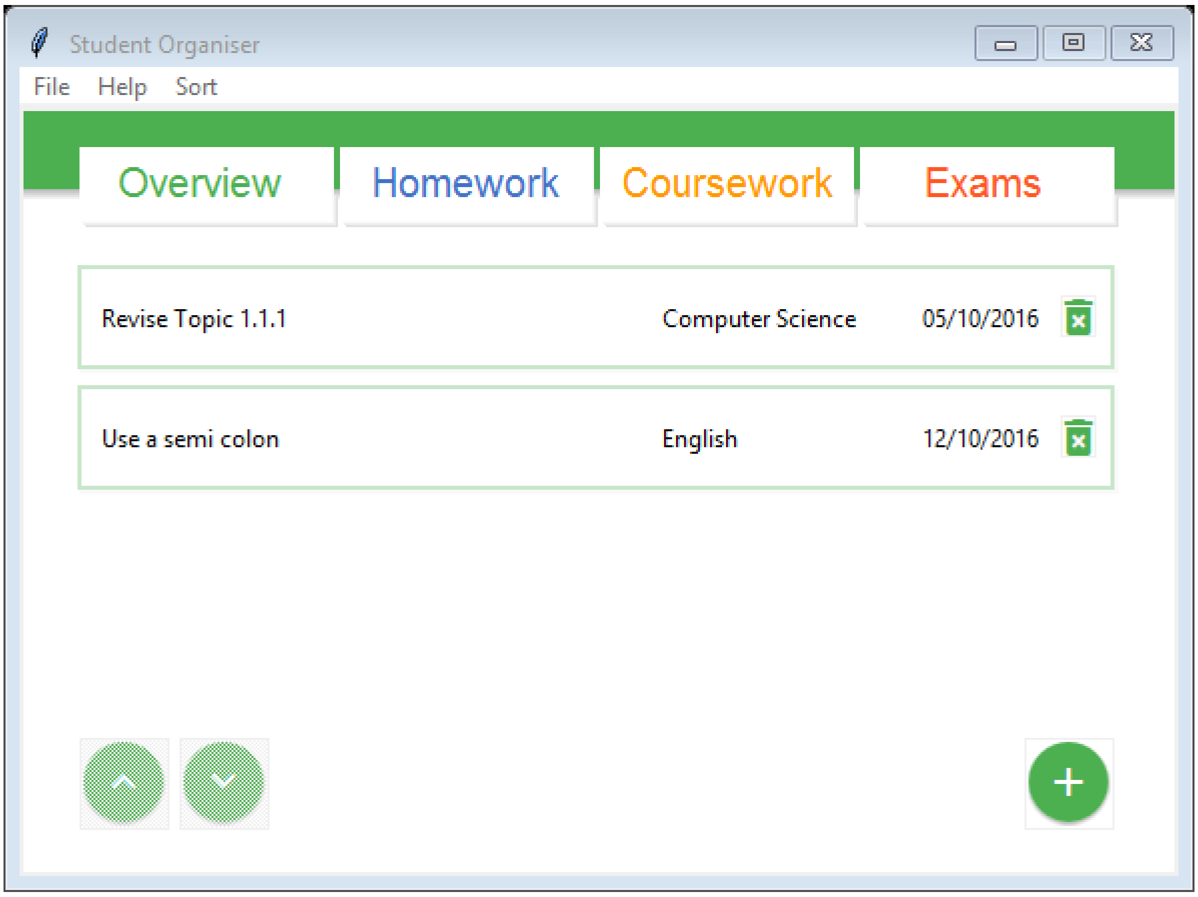


Revise Topic 1.1.1 Computer Science 05/10/2015

The above code is repeated for differing amount of rows available. Some code is copied but slightly changed every time to accommodate for an increasing number of rows. The below code works for two rows:

```python
if self.fullRows == 2:
    row1 = []
    for i in self.set[0]:
        row1.append(i)
    row2 = []
    for i in self.set[1]:
        row2.append(i)
    self.name_1.insert(INSERT, row1[0])
    self.subject_1.insert(INSERT, row1[1])
    self.date_1.insert(INSERT, row1[2])
    self.name_2.insert(INSERT, row2[0])
    self.subject_2.insert(INSERT, row2[1])
    self.date_2.insert(INSERT, row2[2])

    taskDate_1 = time.strptime(row1[2], "%d/%m/%Y")
    taskOverdue_1 = currentDate > taskDate_1
    taskDate_2 = time.strptime(row2[2], "%d/%m/%Y")
    taskOverdue_2 = currentDate > taskDate_2

    if taskOverdue_1 == True:
        self.name_1.configure(fg="#f44336")
        self.subject_1.configure(fg="#f44336")
        self.date_1.configure(fg="#f44336")
    if taskOverdue_2 == True:
        self.name_2.configure(fg="#f44336")
        self.subject_2.configure(fg="#f44336")
        self.date_2.configure(fg="#f44336")

    button_remove_1 = Button(self, image=self.remove, command=self.delete_row_1,
anchor = W)
    button_remove_1.configure(border=0, relief = FLAT)
    button_remove_1_window = self.canvas.create_window(520, 94, anchor=NW,
window=button_remove_1, tags="rowBox")

    button_remove_2 = Button(self, image=self.remove, command=self.delete_row_2,
anchor = W)
    button_remove_2.configure(border=0, relief = FLAT)
    button_remove_2_window = self.canvas.create_window(520, 154, anchor=NW,
window=button_remove_2, tags="rowBox")
    return
```

If I were to add 'Revise topic 1.1.1' as a homework task for computer science and 'Use a semi colon' as a homework task for English, it would appear:



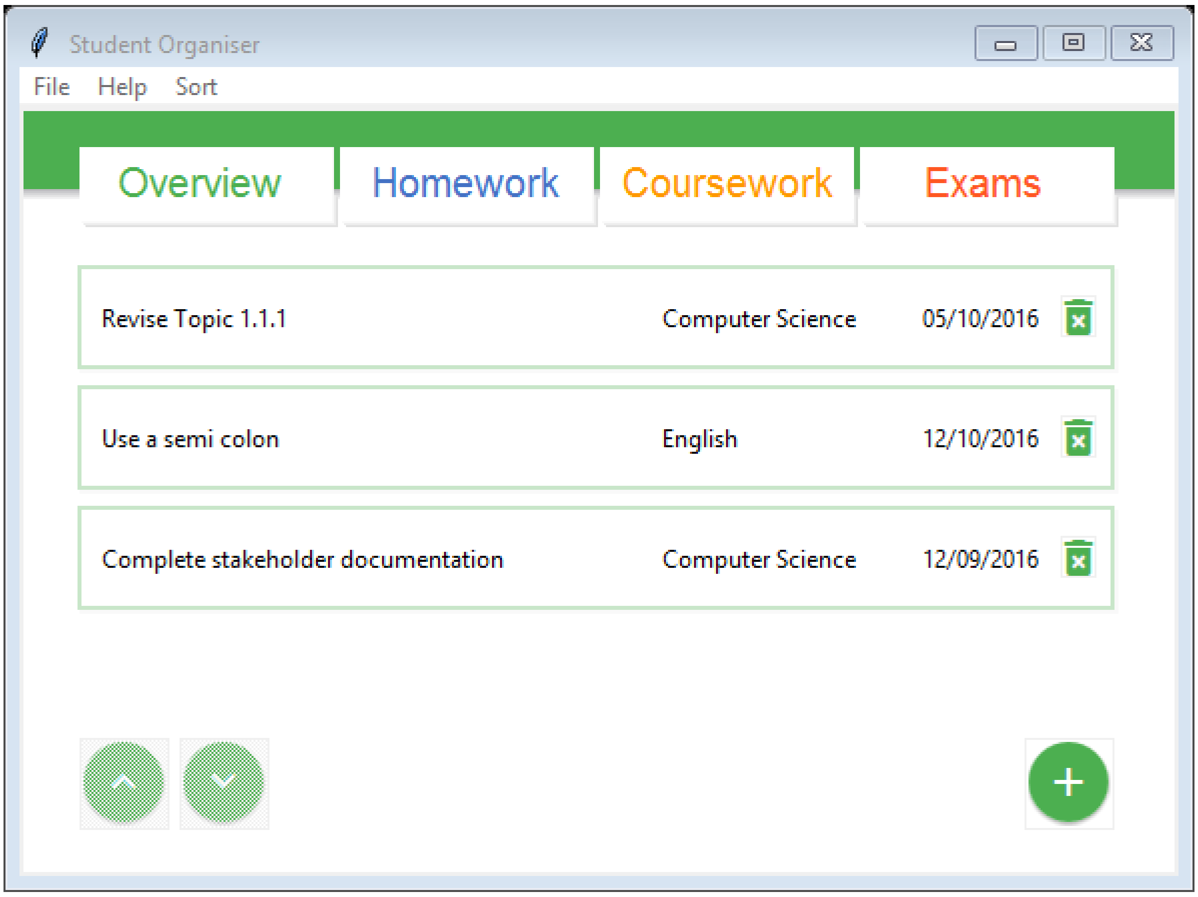
Revise Topic 1.1.1 Computer Science 05/10/2016

Use a semi colon English 12/10/2016

The below code works for three rows:

```python
if self.fullRows == 3:
    row1 = []
    for i in self.set[0]:
        row1.append(i)
    row2 = []
    for i in self.set[1]:
        row2.append(i)
    row3 = []
    for i in self.set[2]:
        row3.append(i)
    self.name_1.insert(INSERT, row1[0])
    self.subject_1.insert(INSERT, row1[1])
    self.date_1.insert(INSERT, row1[2])
    self.name_2.insert(INSERT, row2[0])
    self.subject_2.insert(INSERT, row2[1])
    self.date_2.insert(INSERT, row2[2])
    self.name_3.insert(INSERT, row3[0])
    self.subject_3.insert(INSERT, row3[1])
    self.date_3.insert(INSERT, row3[2])

    taskDate_1 = time.strptime(row1[2], "%d/%m/%Y")
    taskOverdue_1 = currentDate > taskDate_1
    taskDate_2 = time.strptime(row2[2], "%d/%m/%Y")
    taskOverdue_2 = currentDate > taskDate_2
    taskDate_3 = time.strptime(row3[2], "%d/%m/%Y")
    taskOverdue_3 = currentDate > taskDate_3

    if taskOverdue_1 == True:
        self.name_1.configure(fg="#f44336")
        self.subject_1.configure(fg="#f44336")
        self.date_1.configure(fg="#f44336")
    if taskOverdue_2 == True:
        self.name_2.configure(fg="#f44336")
        self.subject_2.configure(fg="#f44336")
        self.date_2.configure(fg="#f44336")
    if taskOverdue_3 == True:
        self.name_3.configure(fg="#f44336")
        self.subject_3.configure(fg="#f44336")
        self.date_3.configure(fg="#f44336")

    button_remove_1 = Button(self, image=self.remove, command=self.delete_row_1, 
anchor = W)
    button_remove_1.configure(border=0, relief = FLAT)
    button_remove_1_window = self.canvas.create_window(520, 94, anchor=NW, 
window=button_remove_1, tags="rowBox")

    button_remove_2 = Button(self, image=self.remove, command=self.delete_row_2, 
anchor = W)
    button_remove_2.configure(border=0, relief = FLAT)
    button_remove_2_window = self.canvas.create_window(520, 154, anchor=NW, 
window=button_remove_2, tags="rowBox")

    button_remove_3 = Button(self, image=self.remove, command=self.delete_row_3, 
anchor = W)
    button_remove_3.configure(border=0, relief = FLAT)
    button_remove_3_window = self.canvas.create_window(520, 214, anchor=NW, 
window=button_remove_3, tags="rowBox")
    return
```

If I were to add ‘Revise topic 1.1.1’ as a homework task for computer science, ‘use a semi Colon’ as a homework task for English and ‘complete stakeholder documentation’ for computer science as a coursework task, it would appear:



Revise Topic 1.1.1 Computer Science 05/10/2016

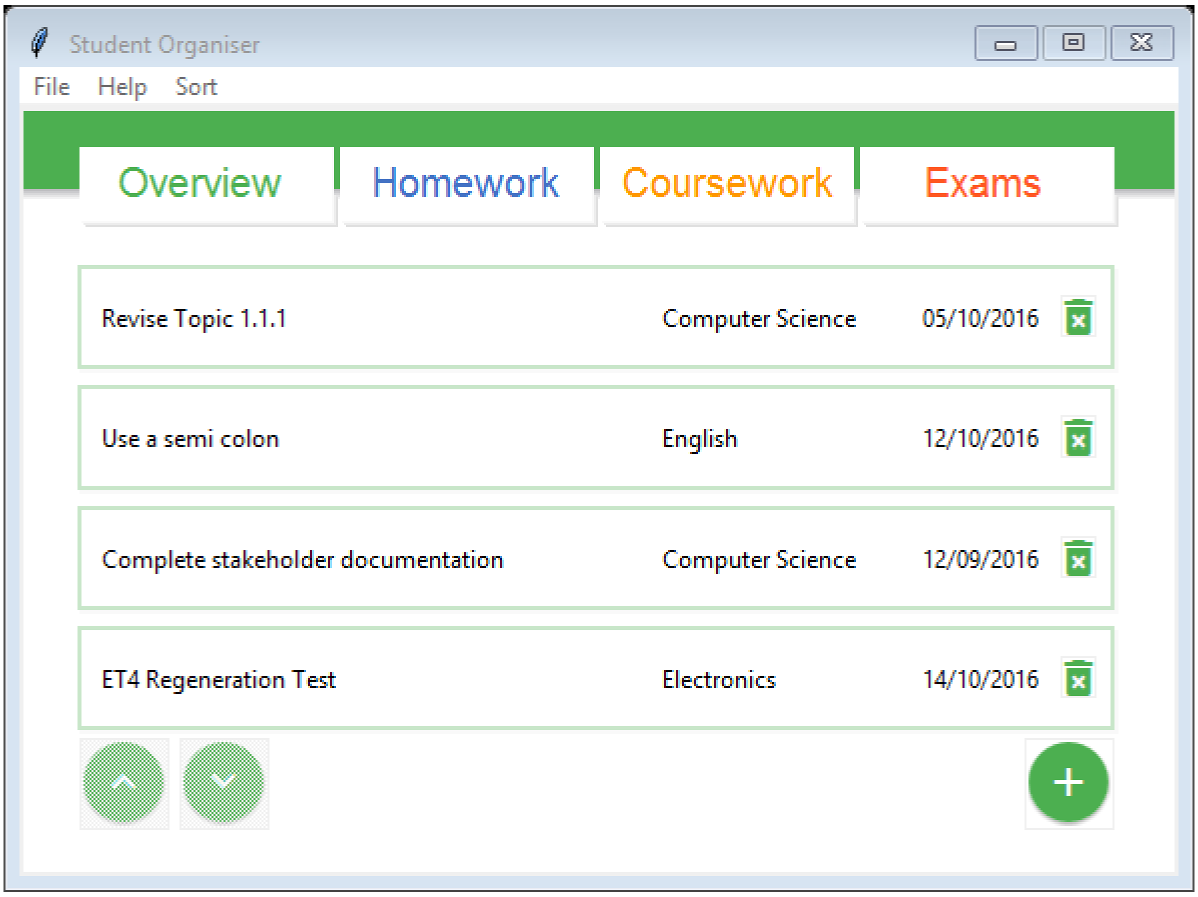
Use a semi colon English 12/10/2016

Complete stakeholder documentation Computer Science 12/09/2016

The below code works for a full set of rows:

```
if self.fullRows == 3:
else:
    self.set=[]
    for i in self.tasks[self.progress[self.page]:self.progress[self.page+1]+1]:
        self.set.append(i)

    # Insert rows
    self.t = 80
    self.b = 130
    for i in range(4):
        self.canvas.create_rectangle(30, self.b+2, 548, self.t+1, fill="#f9f9f9", outline="#f9f9f9", tags="rowBox")
        self.canvas.create_rectangle(30, self.b, 546, self.t, fill="white", outline=self.rowColour, width=2, tags="rowBox")
        self.t += 60
        self.b += 60

    # Split previous array into 4 arrays (1 per row)
    row1 = []
    for i in self.set[0]:
        row1.append(i)
    row2 = []
    for i in self.set[1]:
        row2.append(i)
    row3 = []
    for i in self.set[2]:
        row3.append(i)
    row4 = []
    for i in self.set[3]:
        row4.append(i)

    self.name_1.insert(INSERT, row1[0])
    self.subject_1.insert(INSERT, row1[1])
    self.date_1.insert(INSERT, row1[2])
    self.name_2.insert(INSERT, row2[0])
    self.subject_2.insert(INSERT, row2[1])
    self.date_2.insert(INSERT, row2[2])
    self.name_3.insert(INSERT, row3[0])
    self.subject_3.insert(INSERT, row3[1])
    self.date_3.insert(INSERT, row3[2])
    self.name_4.insert(INSERT, row4[0])
    self.subject_4.insert(INSERT, row4[1])
    self.date_4.insert(INSERT, row4[2])

    taskDate_1 = time.strptime(row1[2], "%d/%m/%Y")
    taskOverdue_1 = currentDate > taskDate_1
    taskDate_2 = time.strptime(row2[2], "%d/%m/%Y")
    taskOverdue_2 = currentDate > taskDate_2
    taskDate_3 = time.strptime(row3[2], "%d/%m/%Y")
    taskOverdue_3 = currentDate > taskDate_3
    taskDate_4 = time.strptime(row4[2], "%d/%m/%Y")
    taskOverdue_4 = currentDate > taskDate_4

    if taskOverdue_1 == True:
        self.name_1.configure(fg="#f44336")
        self.subject_1.configure(fg="#f44336")
        self.date_1.configure(fg="#f44336")
    if taskOverdue_2 == True:
        self.name_2.configure(fg="#f44336")
        self.subject_2.configure(fg="#f44336")
        self.date_2.configure(fg="#f44336")
    if taskOverdue_3 == True:
        self.name_3.configure(fg="#f44336")
        self.subject_3.configure(fg="#f44336")
        self.date_3.configure(fg="#f44336")
    if taskOverdue_4 == True:
        self.name_4.configure(fg="#f44336")
        self.subject_4.configure(fg="#f44336")
        self.date_4.configure(fg="#f44336")

    button_remove_1 = Button(self, image=self.remove, command=self.delete_row_1, anchor = W)
    button_remove_1.configure(border=0, relief = FLAT)
    button_remove_1_window = self.canvas.create_window(520, 94, anchor=NW, window=button_remove_1, tags="rowBox")

    button_remove_2 = Button(self, image=self.remove, command=self.delete_row_2, anchor = W)
    button_remove_2.configure(border=0, relief = FLAT)
    button_remove_2_window = self.canvas.create_window(520, 154, anchor=NW, window=button_remove_2, tags="rowBox")

    button_remove_3 = Button(self, image=self.remove, command=self.delete_row_3, anchor = W)
    button_remove_3.configure(border=0, relief = FLAT)
    button_remove_3_window = self.canvas.create_window(520, 214, anchor=NW, window=button_remove_3, tags="rowBox")

    button_remove_4 = Button(self, image=self.remove, command=self.delete_row_4, anchor = W)
    button_remove_4.configure(border=0, relief = FLAT)
    button_remove_4_window = self.canvas.create_window(520, 274, anchor=NW, window=button_remove_4, tags="rowBox")
```

Below is an example when all four rows have been filled:



Revise Topic 1.1.1	Computer Science	05/10/2016
Use a semi colon	English	12/10/2016
Complete stakeholder documentation	Computer Science	12/09/2016
ET4 Regeneration Test	Electronics	14/10/2016

Page Up:

The next function will be called every time the up arrow is clicked. The function will control the scrolling of the pages in the backward direction. When the user wants to scroll to a previous page, this button will be used. I have written in safeguards so that the user cannot scroll too far ahead or too far before the first and last pages. The code that controls this button:

```python
def page_up(self):
    # Previous page
    if self.tab == "Overview":
        self.ovpage -= 2
    elif self.tab == "Homework":
        self.hwpage -= 2
    elif self.tab == "Coursework":
        self.cwpage -= 2
    elif self.tab == "Exam":
        self.expage -= 2
    self.canvas.delete("rowBox")
    self.load_rows()
```

I used 'if statements' and 'elif statements' to alter the path of this function depending on the current page. This allowed me to use a single function to control the same button across different pages, saving time and increasing efficiency. I chose to use 'elif statements' because the program will only select one of these if the outcome is true, else, it will check the other up until the correct paths has been selected. When the correct tab has been identified, it will remove two from the tab's page count. The reason I selected two was that this number is used in conjunction with my progress array to determine the tasks to load.

The progress array contains values in pairs, e.g. [0,3,4,7,8,11] If the page number was 2, it will load the fourth to the seventh tasks in the array. If the user clicks the up arrow, the page number will become two less than before to make zero. Now, the first to the third tasks will be selected to load. When this has variable has changed, every widget on the canvas with the ID 'rowBox' will be deleted. I created a select set of widgets using this ID so that they all con be removed easily. These are the rectangular outlines around each task. These will all be removed so that the correct amount can be added on the load of the next page; leaving the next set of tasks with the appropriate amount of boxes. Finally, the 'load_rows' function is called and the page is set up with every task on the previous page.

Page Down:

This function will be called every time the down arrow is clicked. The function will control the scrolling of the pages in the forward direction. When the user wants to scroll to a next page, this button will be used. The button is very similar to page up function. The code that controls this button:

```python
def page_down(self):
    # Next page
    if self.tab == "Overview":
        self.ovpage += 2
    elif self.tab == "Homework":
        self.hwpage += 2
    elif self.tab == "Coursework":
        self.cwpage += 2
    elif self.tab == "Exam":
        self.expage += 2
    self.canvas.delete("rowBox")
    self.load_rows()
```

Similarly, to the page up function, the correct tab is identified and the appropriate page value is increased by two. Once this has completed, every row box is deleted and the function, ‘load_rows’ is called, placing every task for the next page into the correct entry boxes.

Development Review:

Tasks can now be displayed. This can handle outputting tasks where the page contains one, two, three or a full set of four tasks. Each piece of task data that is essential to the task is presented to the user in a logical, organised structure. The types of data appear in columns whilst the tasks appear in rows. A table design has been created.

Page buttons have been included to scroll between and sets of tasks that have been created. IF there is only one set in the tab, these buttons will be disabled. If there are multiple, they will be enabled if the direction is available. This supports the criteria that, the software must be presented clearly so that it is easy to use. The user cannot scroll out of the range of pages available.

As tasks are loaded, any task with a date which is before the current date will be written in red text. This was involved with this sprint as it is very closely linked with the loading of the tasks.

I then tested that tasks loaded correctly with sample data that I had written to the database file. I verified that the solution would work with one task, up to and including a full set. In the first test, where I only had a single task, I set the date to before the current and it loaded in red text, showing that this feature had worked.

As all code written at this point functions, I will continue to the next iteration.

Add Task:

One of the most important functions of this program is the ability to add tasks to the database. This will be a separate function called when the ‘add’ button has been clicked. This function will clear the canvas and add new widgets. I will have two rectangular boxes that will house two parts of data inputs. In total, I will add two entry boxes on the left, one for task name and the other for the subject. In the right box, I will have three dropdown menus. These will be used for the date, where the first contains the numbers 1 to 31 for the day, the second will contain all twelve months and the third will have five years to select. The years will be chosen by selecting the current year and allowing the next four years in succession to be available. The overview page is slightly different where I add another dropdown menu to select the type of tasks. I chose to do this, as this function will be used when any tab is selected. If a tab that is not the overview tab is selected, the type that the task is saved as is the tab name.

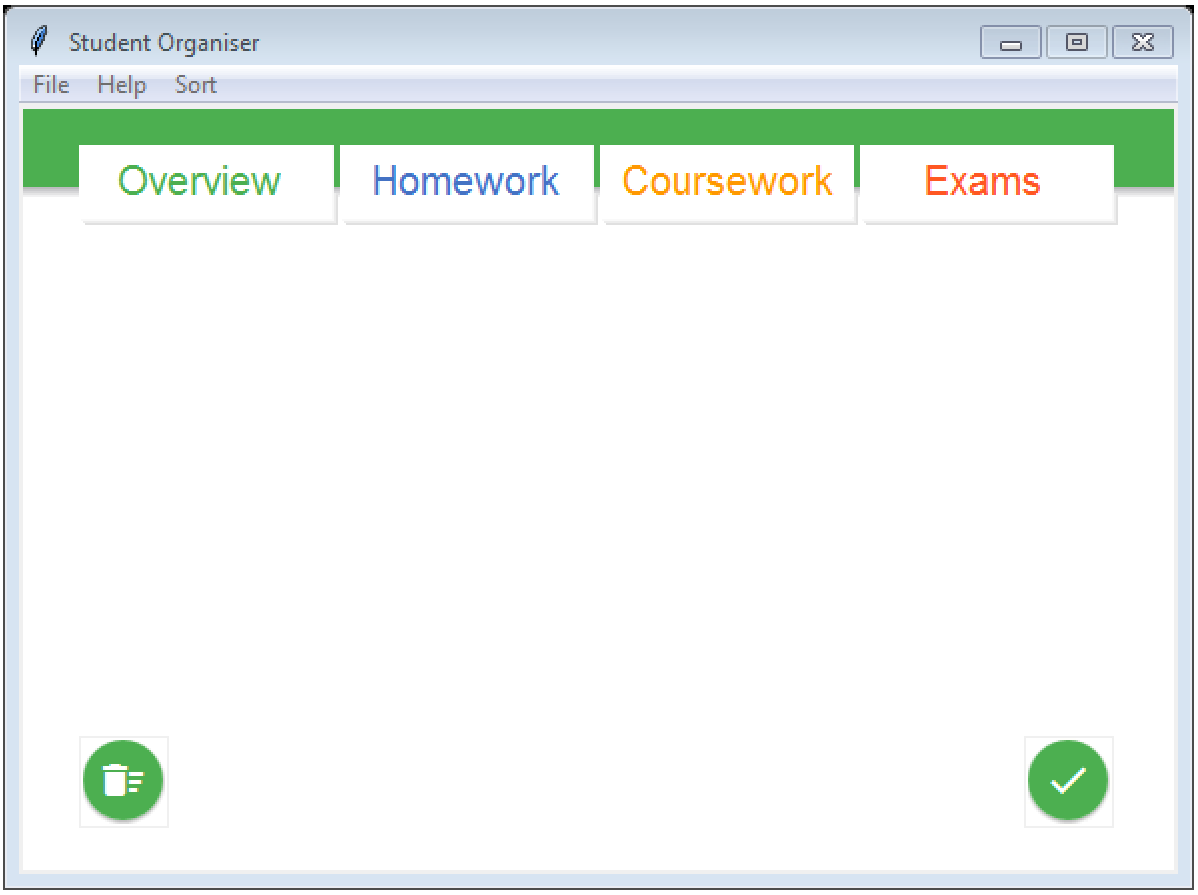
This function starts by assigning the variables for the colours for the theme and the done and delete buttons. These will be the two buttons that will replace the up/down page controls and the add button.

```
1.  def add_item(self):
2.      if self.tab == "Overview":
3.          self.colour = "#4caf50"
4.          self.done = self.greenDone
5.          self.delete = self.greenDelete
6.      if self.tab == "Homework":
7.          self.colour = "#4472C4"
8.          self.done = self.blueDone
9.          self.delete = self.blueDelete
10.     if self.tab == "Coursework":
11.         self.colour = "#ff9800"
12.         self.done = self.yellowDone
13.         self.delete = self.yellowDelete
14.     if self.tab == "Exam":
15.         self.colour = "#ff5722"
16.         self.done = self.redDone
17.         self.delete = self.redDelete
```

Once these have been assigned, the canvas is cleared. The title bar and other widgets that make up the theme are added into this canvas in lines 2, 3 and 4. Then the two main buttons are added. The first will be placed in the same position as the page up button. This button will be the delete button that will delete any progress in this page and return the user to the tasks screen. The second is the done button and will be positioned where the add button would be. The purpose of this button is to submit the task details entered by the user. These details should be validated and then added into the tasks database.

```
        self.canvas.delete(ALL)
        self.draw_title_bar()
        self.canvas.create_rectangle(0, 40, 600, 0, fill=self.colour,
    outline=self.colour)
        self.draw_tabs()

        button_done = Button(self, image=self.done, command=self.check_add_name,
    anchor=W)
        button_done.configure(border=0, relief = FLAT)
        button_done_window = self.canvas.create_window(502, 315, anchor=NW,
    window=button_done)

        button_delete = Button(self, image=self.delete, command=self.return_rows,
    anchor=W)
        button_delete.configure(border=0, relief = FLAT)
        button_delete_window = self.canvas.create_window(30, 315, anchor=NW,
    window=button_delete)
```

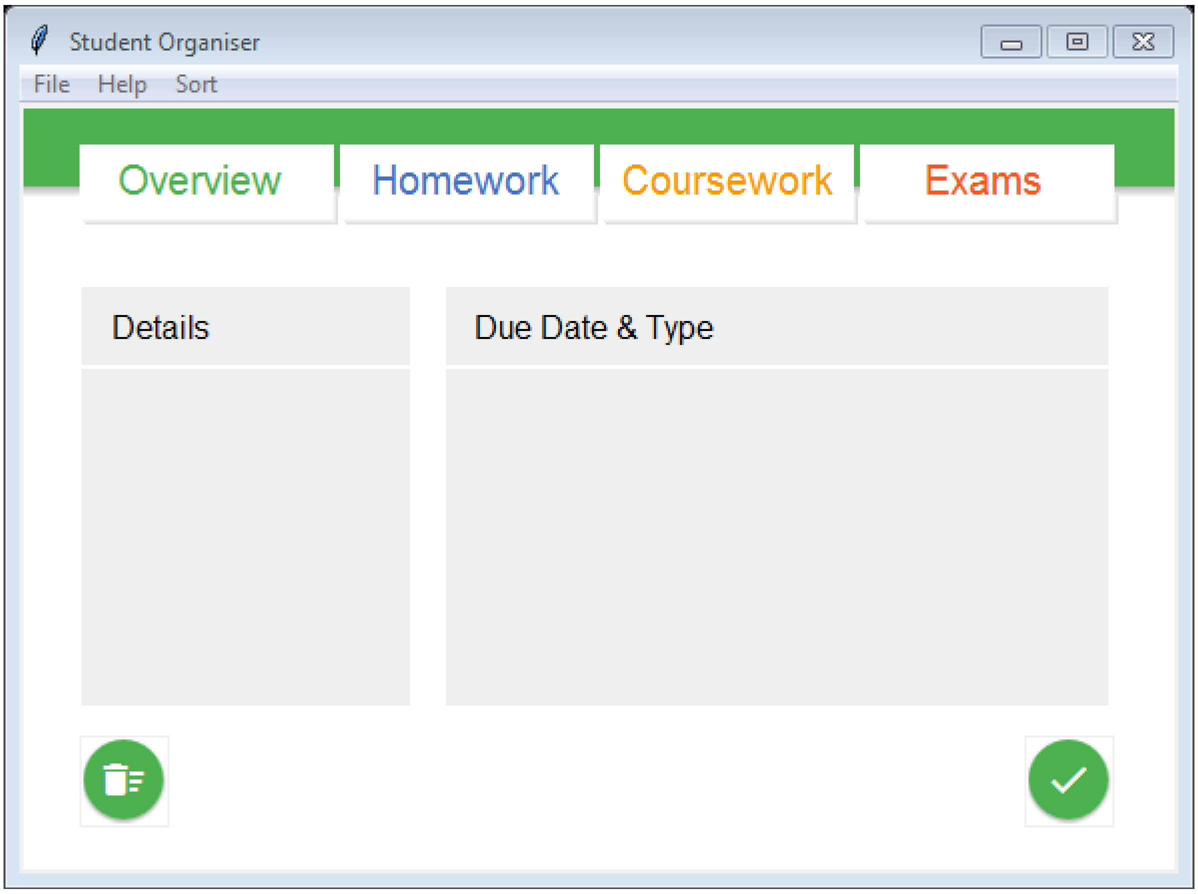
At this stage, the add task page would appear:



The next part of this function is to load in the widgets that the user will interact with to create a task. Here, the two grey rectangles that divide the two sets of inputs will be placed are drawn with the appropriate coordinates. The box on the left will be titled, 'Details' as it will contain the two entry boxes for the task name and subject. The box on the right has differing names depending on the selected tab. If the overview tab is selected, the box will be titled, 'Due Date & Type' else if any other tab is selected, it will become, 'Due Date.' This is because, as stated previously, the overview page will show every task, therefore any task created under this tab could be for any of the three; whereas the other three tabs should automatically assign the task being created to the selected tab type.

```
    self.canvas.create_rectangle(30, 300, 195, 90, fill="#eeeeee",
outline="#FFFFFF")
    self.canvas.create_rectangle(212, 300, 544, 90, fill="#eeeeee",
outline="#FFFFFF")
    self.canvas.create_rectangle(30, 131, 544, 130, fill="#FFFFFF",
outline="#FFFFFF")

    self.canvas.create_text(70, 111, text="Details", font=self.box_heading_font,
fill= "black")
    if self.tab == "Overview":
        self.canvas.create_text(287, 111, text="Due Date & Type",
font=self.box_heading_font, fill="black")
    else:
        self.canvas.create_text(260, 111, text="Due Date",
font=self.box_heading_font, fill="black")
```

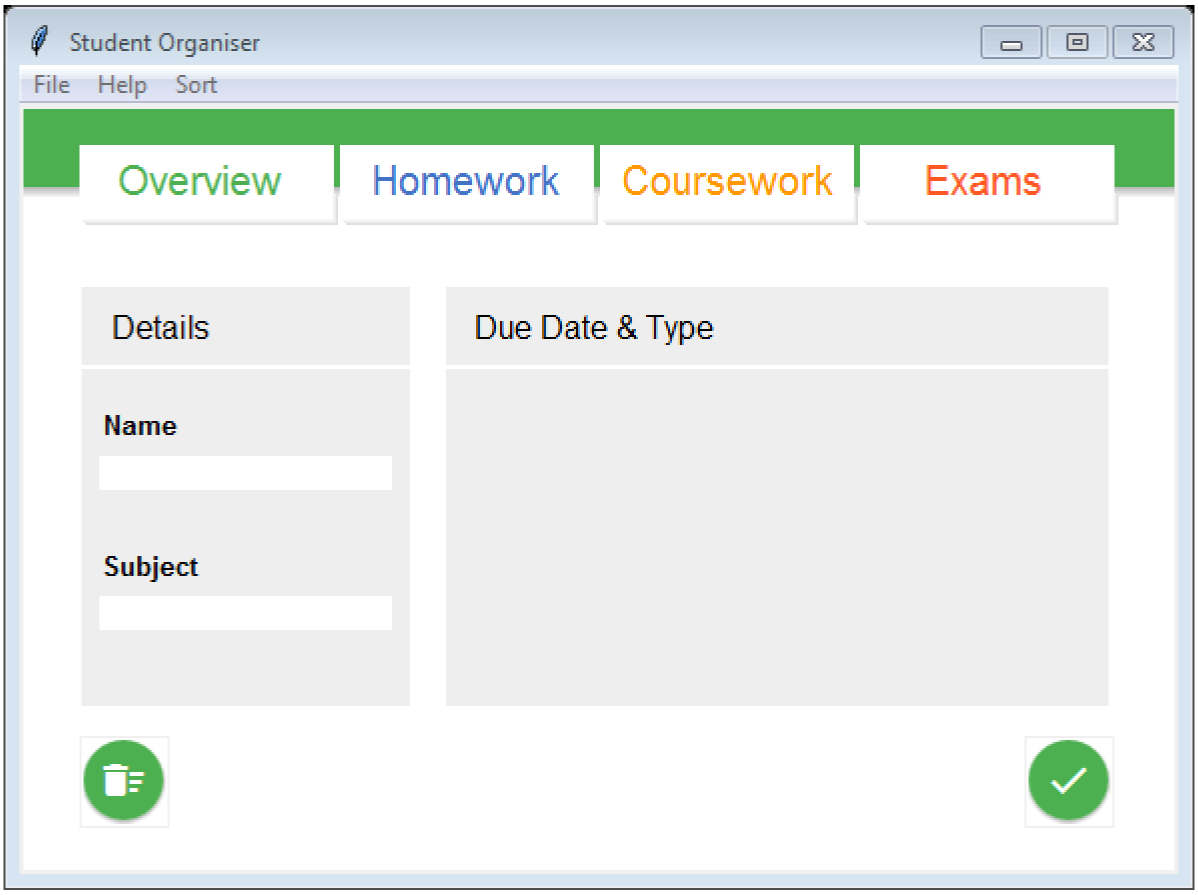
With the two rectangles added:



Then the widgets are added in. Starting with the left hand box, the task name entry box is inserted with a width of 24, which I chose as it fitted the best in the space given; the subject entry box is equal width so that they match. I also added text to the canvas that labels each entry box, showing the user what box is for. I have changed the font for these text widgets so that they are equal but different to that of the headings of the rectangles. This shows that they are for a different purpose and the size is much smaller and in bold to highlight their importance on the canvas.

```
    self.canvas.create_text(60, 160, text="Name", font=self.box_content_font,
fill="black")
    self.task_name = Entry(self,width=24)
    self.task_name.configure(border=0, relief=FLAT, bg="#FFFFFF")
    task_name_window = self.canvas.create_window(40, 175, anchor=NW,
window=self.task_name)

    self.canvas.create_text(65, 230, text="Subject", font=self.box_content_font,
fill="black")
    self.task_subject = Entry(self,width=24)
    self.task_subject.configure(border=0, relief=FLAT, bg="#FFFFFF")
    task_subject_window = self.canvas.create_window(40, 245, anchor=NW,
window=self.task_subject)
```

With the two entry boxes:

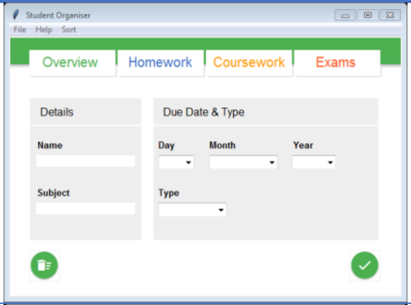
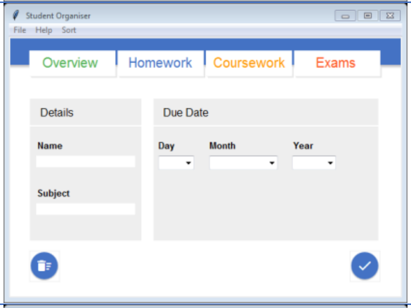
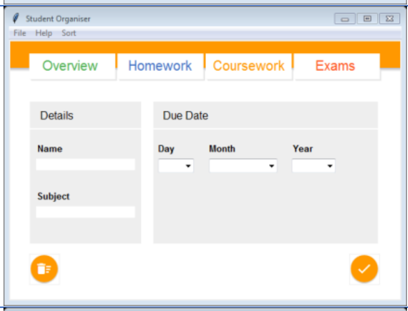
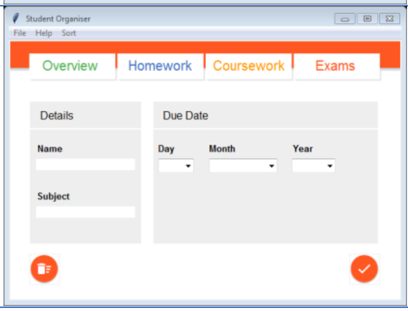


For the right hand rectangle, this is where I add in the widgets to select the date and task type, if the overview tab is selected. I used three entry boxes to select the due date. The first would contain increasing numerical values from ‘1’ to ‘31’ to select the day, the second would contain every month in ascending order and the final will have a choice of five years. I had decided to allow the user to select the current year and the next four years after. This means that the program will allow the user to select the current year and time this program is ever used and the next four years should be a suitable range to plan any events.

Below these comboboxes, I have added a fourth combobox. This will be used to select the test type. This will only be shown for the ‘overview’ tab as any task added here could be any of the three. AS for the other tabs, it is assumed that the current tab determines the task type.

```python
    # Every number from 1 to 31 is assigned to the array called self.days
    # A text label is created 'Day'
    # A combobox is created with the contents of the self.days array
    # The combobox is positioned on the canvas
    self.days = [1,2,3,4,5,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31]
    self.canvas.create_text(232, 160, text="Day", font=self.box_content_font, fill="black")
    self.task_day = ttk.Combobox(self, values=self.days, state='readonly', width=5)
    task_day_window = self.canvas.create_window(220, 175, anchor=NW, window=self.task_day)

    # Every month from January to December is assigned to the array called self.months
    # A text label is created 'Month'
    # A combobox is created with the contents of the self.months array
    # The combobox is positioned on the canvas
    self.months = ['January','February','March','April','May','June','July','August','September','October','November','December']
    self.canvas.create_text(314, 160, text="Month", font=self.box_content_font, fill="black")
    self.task_month = ttk.Combobox(self, values=self.months, state='readonly', width=13)
    task_month_window = self.canvas.create_window(295, 175, anchor=NW, window=self.task_month)

    # The current year is assigned to self.year
    # An empty array is created 'self.years'
    # The current year and the next four years are added to the self.years array
    self.year = (time.strftime("%Y"))
    self.years=[]
    self.years.append(int(self.year))
    self.years.append(int(self.year) + 1)
    self.years.append(int(self.year) + 2)
    self.years.append(int(self.year) + 3)
    self.years.append(int(self.year) + 4)

    # A text label is created 'Year'
    # A combobox is created with the contents of the self.years array
    # The combobox is positioned on the canvas
    self.canvas.create_text(433, 160, text="Year", font=self.box_content_font, fill="black")
    self.task_year = ttk.Combobox(self, values=self.years, state='readonly', width=7)
    task_year_window = self.canvas.create_window(417, 175, anchor=NW, window=self.task_year)

    # If the array tab is selected:
    # A text label is created 'self.type'
    # A combobox is created with the contents of the self.type array
    # The combobox is positioned on the canvas
    if self.tab == "Overview":
        self.type = ['Homework','Coursework','Exam']
        self.canvas.create_text(235, 230, text="Type", font=self.box_content_font, fill="black")
        self.task_type = ttk.Combobox(self, values=self.type, state='readonly', width=13)
        task_type_window = self.canvas.create_window(220, 245, anchor=NW, window=self.task_type)
```

Here, I will test the two types of page where I will add in the task.

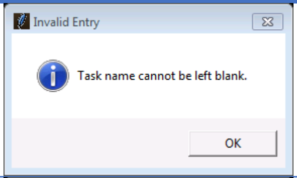
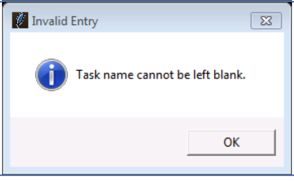
Test	Test Data	Expected Result	Actual Result / Evidence
Add a task when the overview tab is selected	N/A	The program should show the add task page with a dropdown menu for the three types.	 Student Organiser File Help Sort Overview Homework Coursework Exams Details Name Subject Due Date & Type Day Month Year Type
Add a task when the homework tab is selected	N/A	The program should show the add task page without a dropdown menu for the three types. The task type should be based on the current tab, homework.	 Student Organiser File Help Sort Overview Homework Coursework Exams Details Name Subject Due Date Day Month Year
Add a task when the coursework tab is selected	N/A	The program should show the add task page without a dropdown menu for the three types. The task type should be based on the current tab, coursework.	 Student Organiser File Help Sort Overview Homework Coursework Exams Details Name Subject Due Date Day Month Year
Add a task when the exams tab is selected	N/A	The program should show the add task page without a dropdown menu for the three types. The task type should be based on the current tab, exams.	 Student Organiser File Help Sort Overview Homework Coursework Exams Details Name Subject Due Date Day Month Year

Input Validation:

Once the user has entered data or clicked the 'done' button then the data must be validated. This protect the program as it will not crash if an invalid input is used. In addition, it will ensure that the data entered will be useful for the student and all necessary data is present and written to the database correctly. In the case of an invalid entry, the program should open a dialogue box with the relevant message detailing the issue found. I had chosen this method to alert the user of the issue so that they can make the appropriate amendments and the user cannot attempt to use the program whilst the message window is still open, making the alert very obvious.

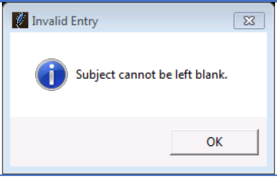
The first check is to verify that the task name is present. If not, a dialogue message should appear with the content, "Task name cannot be left blank" and no other changes to the program should occur.

```
def check_add_name(self):
    if self.task_name.get() == "":
        messagebox.showinfo("Invalid Entry", "Task name cannot be left blank.")
        pass
    else:
        self.check_add_subject()
```

Test	Test Data	Expected Result	Actual Result / Evidence
Do not enter anything		An error message appears with the text, 'Task name cannot be left blank.'	 Invalid Entry Task name cannot be left blank. OK
Do not enter a task name		An error message appears with the text, 'Task name cannot be left blank.'	 Invalid Entry Task name cannot be left blank. OK

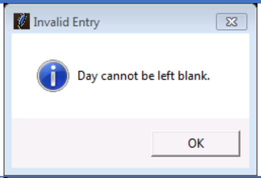
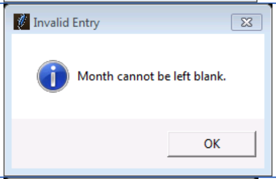
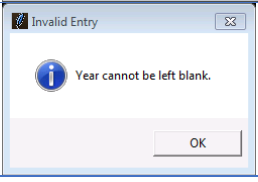
If this is present, the subject entry box is tested to confirm an input is existent. If not, a dialogue message should appear with the content, "Subject cannot be left blank" and no other changes to the program should occur.

```
def check_add_subject(self):
    if self.task_subject.get() == "":
        messagebox.showinfo("Invalid Entry", "Subject cannot be left blank.")
        pass
    else:
        self.check_add_date()
```

Test	Test Data	Expected Result	Actual Result / Evidence
Do not enter a subject	name	An error message appears with the text, 'Subject cannot be left blank.'	 Invalid Entry Subject cannot be left blank. OK

If this is present, the date entry is tested to confirm that every combobox involved in selecting the date has a value selected. If not, a dialogue message should appear with the content, “Day cannot be left blank” or “Month cannot be left blank” or “Year cannot be left blank,” depending on the combobox without a selection. I have chosen to create this validation in such a way that the user will only be altered of one error at a time. This avoids numerous dialogue boxes from opening and possibly annoying the user. If there were multiple mistakes’ they will still see every error that is made as, when the first correction occurs, and the user selects the ‘done’ button, the next error will be found and the user will be informed. The process will repeat each time and each time, it cycles through every test.

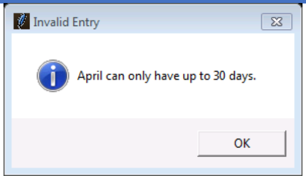
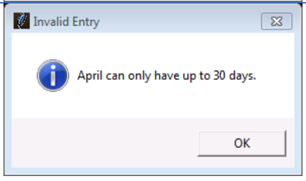
```
def check_add_date(self):
    if self.task_day.get() == "":
        messagebox.showinfo("Invalid Entry", "Day cannot be left blank.")
        pass
    else:
        if self.task_month.get() == "":
            messagebox.showinfo("Invalid Entry", "Month cannot be left blank.")
            pass
        else:
            if self.task_year.get() == "":
                messagebox.showinfo("Invalid Entry", "Year cannot be left blank.")
                pass
            else:
                self.check_add_valid_date()
```

Test	Test Data	Expected Result	Actual Result / Evidence
Do not select a day	Name subject	An error message appears with the text, ‘Day cannot be left blank.’	 Invalid Entry Day cannot be left blank. OK
Do not select a month	Name Subject 1	An error message appears with the text, ‘Month cannot be left blank.’	 Invalid Entry Month cannot be left blank. OK
Do not select a year	Name Subject 1 January	An error message appears with the text, ‘Year cannot be left blank.’	 Invalid Entry Year cannot be left blank. OK

No other changes to the program should occur. This is because the date entered must be validated. As it is confirmed the date is present, it is now possible to check the authenticity of the input. This will avoid any errors when writing to or reading from the database as the data type for this value is date. If an invalid date is entered, errors will occur. To fix this, I used an algorithm that starts by assigning every month a numerical value. Once this is complete, two arrays are created. The first contains the numerical values of the months that can have a maximum of 30 days and the other will contain the numerical values of the months that can have a maximum of 31 days. February is the only month not be included in this, as it will depend on the year.

Two checks are performed; if the current day selected is greater than 30 and the month is in the array defining the months with a maximum of 30 days, a dialogue message should appear with the content, “<month> can only have up to 30 days” and no other changes to the program should occur.

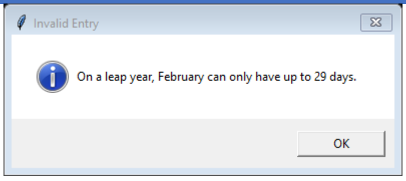
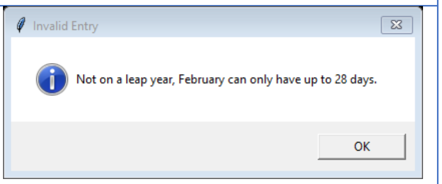
```python
def check_add_valid(self):
    # Every month is assigned a numerical value
    if self.task_month.get() == "January":
        self.month_number = "01"
    if self.task_month.get() == "February":
        self.month_number = "02"
    if self.task_month.get() == "March":
        self.month_number = "03"
    if self.task_month.get() == "April":
        self.month_number = "04"
    if self.task_month.get() == "May":
        self.month_number = "05"
    if self.task_month.get() == "June":
        self.month_number = "06"
    if self.task_month.get() == "July":
        self.month_number = "07"
    if self.task_month.get() == "August":
        self.month_number = "08"
    if self.task_month.get() == "September":
        self.month_number = "09"
    if self.task_month.get() == "October":
        self.month_number = "10"
    if self.task_month.get() == "November":
        self.month_number = "11"
    if self.task_month.get() == "December":
        self.month_number = "12"

    # The maximum number of days for each month is determined
    # 2 arrays are created
    # The first contains every month that can have up to 30 days
    # The second contains every month that can have up to 31 days
    days_30 = ('4','6','9','11')
    days_31 = ('1','3','5','7','8','10','12')
    # If the user enters a day value greater than 30 for a 30 day month, a message will be shown
    # If the user enters a day value greater than 31 for a 31 day month, a message will be shown
    if self.month_number in days_30 and int(self.task_day.get()) > 30:
        messagebox.showinfo("Invalid Entry", (str(self.task_month.get())+ " can only have up to 30 days."))
        pass
    elif self.month_number in days_31 and self.task_day.get() > 31:
        messagebox.showinfo("Invalid Entry", str(self.task_month.get())+str(" can only have up to 31 days."))
        pass
    else:
        self.check_leap_year()
```

Test	Test Data	Expected Result	Actual Result / Evidence
Enter all values with a 31 day for a 30 day month	31 April 2016	An error message appears with the text, 'April can only have up to 30 days.'	 Invalid Entry April can only have up to 30 days. OK
Enter all values with a 31 day less than 31 for a 30 day month	31 April 2016	An error message appears with the text, 'April can only have up to 30 days.'	 Invalid Entry April can only have up to 30 days. OK

If the correct day and month combination is selected, it then checks for leap years. This ensures that February will have the correct day value too. A year is a leap year if it is divisible by four and can be divided by 400, however, if it is divisible by 4 and divisible by 100, it is not a leap year. This works by using the user's input to the year combobox and performing the calculation of selected year mod 4. If this value is equal to zero, then it is divisible by 4. Then the calculation, selected year mod 100 is performed and if this equals zero, it is stated that the year is not a leap year. Else, the result of the selected year mod 400 should be a zero, in which case, it is a leap year.

Using the result of the previous set of calculations, the day selected must be checked if the selected month is February. If the month is February, it is a leap year and the selected day is greater than 29, a dialogue message should appear with the content, "On a leap year, February can only have up to 29 days" and no other changes to the program should occur. If February has been chosen, it is not a leap year and the selected day is greater than 28, a dialogue message should appear with the content, "Not on a leap year, February can only have up to 28 days" and as before, no other changes to the program should occur.

```
def check_leap_year(self):
    # Check if the year selected is a leap year
    # A variable 'leap_year' is created an is assigned as false
    # If the selected year can be divided by 4, it is considered a leap year
    # But, if the selected year can be divided by 100, it is no longer a leap year
    # If the selected year can be divided by 400, it is a leap year
    leap_year=False
    if (int(self.task_year.get())%4) == 0:
        leap_year=True
        if (int(self.task_year.get())%100) == 0:
            leap_year=False
        elif (int(self.task_year.get())%400) == 0:
            leap_year=True

    # If the user has selected February and a leap_year is true, the day cannot be greater than 29
    # If the user has selected February and a leap_year is false, the day cannot be greater than 28
    # If the day is valid, the write_task function is called
    if self.month_number == 2 and leap_year and int(self.task_day.get()) > 29:
        messagebox.showinfo("Invalid Entry", “On a leap year, ”+str(self.task_month.get())+(" can only have up to 29 days.")
        pass
    elif self.month_number == 2 and not leap_year and int(self.task_day.get()) > 28:
        messagebox.showinfo("Invalid Entry", “Not on a leap year, ”+str(self.task_month.get())+" can only have up to 28 days.")
        pass
    else:
        self.write_task()
```

Test	Test Data	Expected Result	Actual Result / Evidence
Enter all values with a day greater than 29 for February on a leap year	30 February 2016	An error message appears with the text, 'On a leap year, February can only have up to 29 days.'	 Invalid Entry On a leap year, February can only have up to 29 days. OK
Enter all values with a day greater than 28 for February not on a leap year	31 February 2017	An error message appears with the text, 'Not on a leap year, February can only have up to 28 days.'	 Invalid Entry Not on a leap year, February can only have up to 28 days. OK

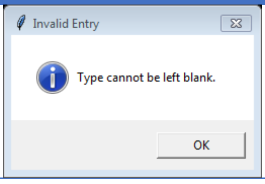
Writing to Database:

Once all data has been validated, it must be written to the tasks table in the database. Each record in this table will contain a task. To ensure that the data is in the correct format for creating a record, any day values that are single digits must become double, therefore adding a zero to the beginning would solve this. The data is converted into the correct format of YYYY/MM/DD as this is how the databases stores the date. A date variable is constructed by concatenating the three variables in this order.

```python
self.date = str(self.task_year.get()) + "/" + str(self.month_number) + "/" +
str(self.day_number)
```

Before a connection to the database is established, another validation takes place. If the overview tab had been selected, the program will check if a task type had been selected. This ensures that the task is written to the correct tab.

```python
# Check if type has been selected
if self.tab == "Overview":
    if self.task_type.get() == "":
        tkMessageBox.showinfo("Invalid Entry", "Type cannot be left blank.")
        return
```

Test	Test Data	Expected Result	Actual Result / Evidence
Enter all values without tab selected	N/A	An error message appears with the text, 'Type cannot be left blank.'	 Invalid Entry Type cannot be left blank. OK

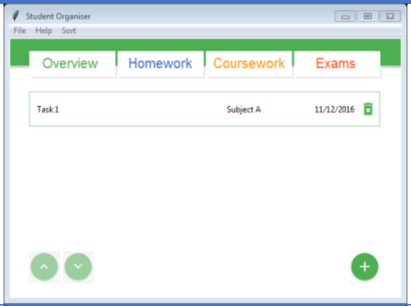
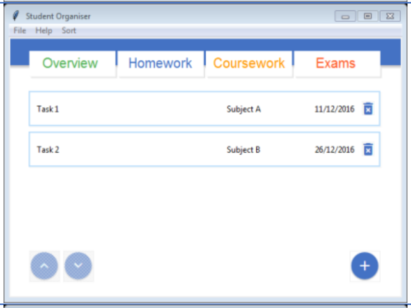
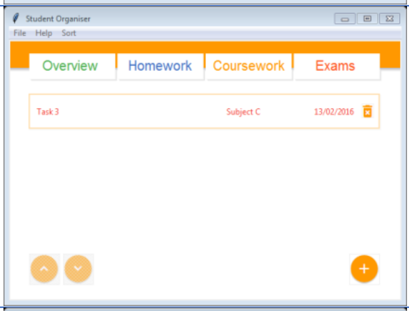
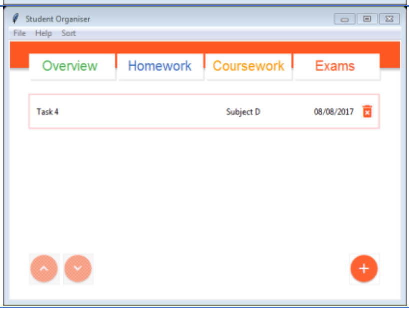
Now, the data must be written to the database. To start, a connection to the database is established, and the ID of the last record in the database is found. This is important as it will be used when inserting a new record. With this value, a validation process occurs. If the value of the last row is null, the variable is assigned zero, as this will be the first record in the table to add.

Depending on the tab selected, decides how the task is written. If the overview tab is selected, the tab selected in the dropdown menu is used, if another tab is selected before adding the task, that value is assigned.

```python
# A connection to the database is established
conn = sqlite3.connect('tasks.db')
# The ID of the last record in the table is found
# If this value doesn't exist, it is assigned '0'
lastRow = conn.execute("SELECT MAX(id) FROM TASKS")
for row in lastRow:
    self.id = row[0]
if self.id == None:
    self.id = 0
if self.tab == "Overview":
    conn.execute("INSERT INTO TASKS(ID, TASK, SUBJECT, DATE, TAB) values (?, ?, ?,
?, ?)", (self.id+1, self.task_name.get(), self.task_subject.get(), self.date,
self.task_type.get()))
else:
    conn.execute("INSERT INTO TASKS(ID, TASK, SUBJECT, DATE, TAB) values (?, ?, ?,
?, ?)", (self.id+1, self.task_name.get(), self.task_subject.get(), self.date,
self.tab))
 conn.commit()
 conn.close()
```

Once the user had entered all the valid information needed and then clicks the done button, the program needs to show the tasks page. For example, when adding a task on the homework page, the homework tasks page should load.

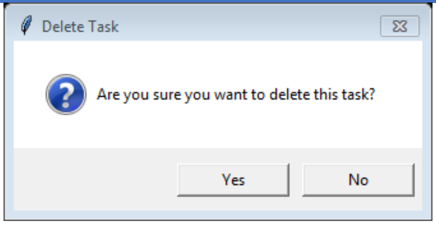
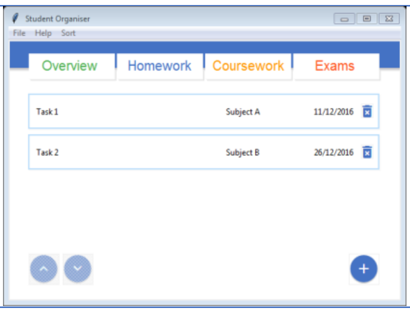
```python
# The program will show the relevant tasks page once added
if self.tab == "Overview":
    self.setup_overview()
if self.tab == "Homework":
    self.setup_homework()
if self.tab == "Coursework":
    self.setup_coursework()
if self.tab == "Exam":
    self.setup_exams()
```

Test	Test Data	Expected Result	Actual Result / Evidence
Enter all values when adding a task for the overview tab	Task 1 Subject A 11/12/2016 Homework	The program should show every task on a page with the new task added in.	 Student Organiser File Help Sort Overview Homework Coursework Exams Task 1 Subject A 11/12/2016
Enter all values when adding a task for the homework tab	Task 2 Subject B 26/12/2016 Homework	The program should show the page for homework tasks with the new task added in.	 Student Organiser File Help Sort Overview Homework Coursework Exams Task 1 Subject A 11/12/2016 Task 2 Subject B 26/12/2016
Enter all values when adding a task for the coursework tab	Task 3 Subject C 13/02/2016 Coursework	The program should show the page for coursework tasks with the new task added in.	 Student Organiser File Help Sort Overview Homework Coursework Exams Task 3 Subject C 13/02/2016
Enter all values when adding a task for the exams tab	Task 4 Subject D 08/08/2017 Exams	The program should show the page for exam tasks with the new task added in.	 Student Organiser File Help Sort Overview Homework Coursework Exams Task 4 Subject D 08/08/2017

Cancel Adding a Task:

If the user does not want to create the task, they can click the bin icon to clear the fields and return to the previous page. This allows the user to quickly delete the task and return to where they left off. I had written a small function to control this process. If the user clicks this button, a message box will appear with the message, “Are you sure you want to delete this task?” There will be two buttons, ‘Yes’ and ‘No’. If the user opts for the affirmative, then the tasks page will load with the relevant tab selected. Else, the message window will close and the data entered will remain unaltered.

```python
def return_rows(self):
    if messagebox.askyesno("Delete Task", "Are you sure you want to delete this task?"):
        if self.tab == "Overview":
            self.setup_overview()
        if self.tab == "Homework":
            self.setup_homework()
        if self.tab == "Coursework":
            self.setup_coursework()
        if self.tab == "Exam":
            self.setup_exams()
    else:
        pass
```

Test	Test Data	Expected Result	Actual Result / Evidence
Enter all values when adding a task and click the delete button	Task 5 Subject E 28/10/2016 Exams	A message appears with the options, ‘Yes’, ‘No’ and the text, ‘Type cannot be left blank.’	 Delete Task Are you sure you want to delete this task?
Enter all values when adding a task for the homework tab	Task 2 Subject B 26/12/2016 Homework	The program should show the page for homework tasks with the new task added in.	 Student Organiser File Help Sort Overview Homework Coursework Exams Task 1 Subject A 11/12/2016 Task 2 Subject B 26/12/2016

Client Meeting:

This is the first meeting with the client. My client had said that they would like to have the test type dropdown menu to appear in every add task page regardless of the current tab. The value in this list should automatically be select for the current tab. If the overview tab is selected, no values should be pre-selected.

Adjustments:

I changed a few parts of the code so that the clients brief would be met. The first change was to always display the title, “Due Date & Type” on every page where the task is added, regardless on the selected tab.

The code before:

```python
if self.tab == "Overview":
    self.canvas.create_text(287, 111, text="Due Date & Type", font=self.box_heading_font, fill="black")
else:
    self.canvas.create_text(260, 111, text="Due Date", font=self.box_heading_font, fill="black")
```

The code after:

```python
self.canvas.create_text(287, 111, text="Due Date & Type", font=self.box_heading_font, fill="black")
```

This will now always display the revised title for every page. The next change was to adjust the default selected values when the combobox is inserted into the page. This is where the value in the list will be selected based on the selected tab.

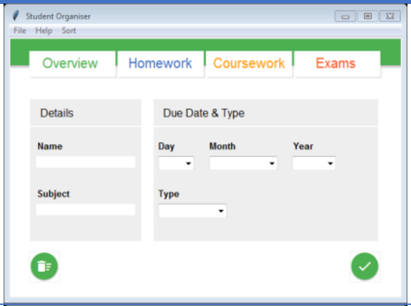
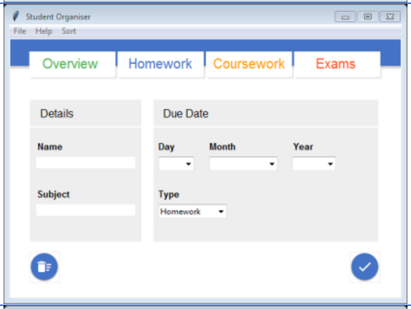
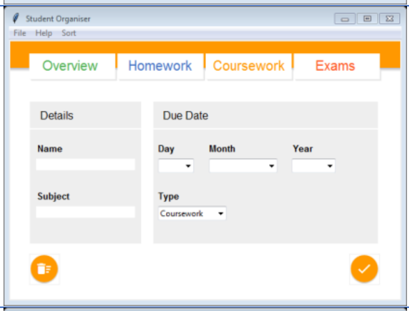
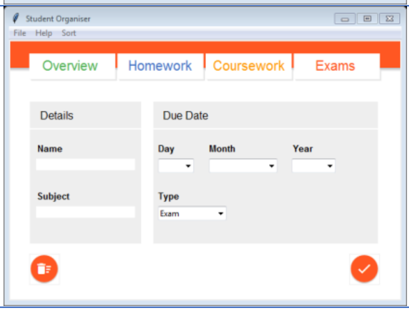
The code before:

```python
# If the overview tab is selected, a dropdown menu of task types will be displayed.
self.type = ['Homework','Coursework','Exam']
if self.tab == "Overview":
    self.type = ['Homework','Coursework','Exam']
    self.canvas.create_text(235, 230, text="Type", font=self.box_content_font, fill="black")
    self.task_type = ttk.Combobox(self, values=self.type, state='readonly', width=13)
    task_type_window = self.canvas.create_window(220, 245, anchor=NW, window=self.task_type)
```

The code after:

```python
# The dropdown menu will select the value for the current tab by default
# If the overview tab is selected, no values are automatically selected.
self.type = ['Homework','Coursework','Exam']
self.canvas.create_text(235, 230, text="Type", font=self.box_content_font, fill="black")
self.task_type = ttk.Combobox(self, values=self.type, state='readonly', width=13)
if self.tab == "Homework":
    self.task_type.set('Homework')
elif self.tab == "Coursework":
    self.task_type.set('Coursework')
elif self.tab == "Exam":
    self.task_type.set('Exam')
task_type_window = self.canvas.create_window(220, 245, anchor=NW, window=self.task_type)
```

Testing of the Amendments:

Test	Test Data	Expected Result	Actual Result / Evidence
Add a task when the overview tab is selected	N/A	The program should show the add task page with a dropdown menu for the three types where there is no value selected.	 Student Organiser File Help Sort Overview Homework Coursework Exams Details Due Date & Type Name Day Month Year Subject Type
Add a task when the homework tab is selected	N/A	The program should show the add task page with a dropdown menu for the three types where 'Homework' is selected.	 Student Organiser File Help Sort Overview Homework Coursework Exams Details Due Date Name Day Month Year SubjectType: Homework
Add a task when the coursework tab is selected	N/A	The program should show the add task page with a dropdown menu for the three types where 'Coursework' is selected.	 Student Organiser File Help Sort Overview Homework Coursework Exams Details Due Date Name Day Month Year SubjectType: Coursework
Add a task when the exams tab is selected	N/A	The program should show the add task page with a dropdown menu for the three types where 'Exams' is selected.	 Student Organiser File Help Sort Overview Homework Coursework Exams Details Due Date Name Day Month Year SubjectType: Exam

Development Review:

Tasks can now be entered by the user. The data in my previous review had been entered externally, directly into the database table in order to test the output of the program. The task is added by clicking the add button to load a page dedicated for this. This page offers a variety of input methods where I have selected the most appropriate entry method for the data type requested. All entries have been labelled for the easy to use success criteria and they have been split into two sections for clarity.

At this stage in development, this version has implemented the following success criteria fully, an option to load a page specifically to add a task.

Sorting:

Here I added in one of the main functions to the program, sorting. I asked the stakeholder which sorting methods they would find most useful in this program. The result was that they would like to be able to sort the tasks by time based on their due date and alphabetically for task name and the subject. Each method should be able to sort in ascending and descending order.

From this, I created this set of sorting methods:

- Time: Soonest (Default)
- Time: Oldest
- Task: A-Z
- Task: Z-A
- Subject: A-Z
- Subject: Z-A

I have decided that all tasks will be sorted by their due date; therefore, I have included this as the first sorting option with the 'default' label applied. I have grouped these options so that navigating the sorting list would be simple and logical. The stakeholder has agreed that this satisfies their requirements.

Time: Soonest

To sort by time soonest, I first connected to the database where the tasks are stored. Then an empty tasks array is created. This is the same array as what is used when loading the tasks. If overview is the selected tab, then it will select every task in the table and append them to the tasks array by the date with the earliest at the top. Else, it will only select the tasks only for that tab and append those to the array. Once this has completed, the array, 'tasks' will have the required tasks in the selected order and to load these into the rows, the function, 'self.refresh_task_list()' loads.

```python
def sort_1(self):
    self.conn = sqlite3.connect('tasks.db')
    self.tasks = []
    if self.tab == "Overview":
        self.cursor = self.conn.execute("SELECT * FROM tasks ORDER BY DATE ASC")
    else:
        self.cursor = self.conn.execute("SELECT * FROM tasks WHERE TAB='"+str(self.tab)+"' ORDER BY DATE ASC")
    self.refresh_task_list()
```

Time: Oldest

To sort by time oldest, I used the same function as sorting by tasks with the earliest time first. The only change to reverse this order was within the SQL statement. I replaced 'ASC' with 'DESC' so that the tasks would be appended to the tasks array in the reverse order.

```python
def sort_2(self):
    self.conn = sqlite3.connect('tasks.db')
    self.tasks = []
    if self.tab == "Overview":
        self.cursor = self.conn.execute("SELECT * FROM tasks ORDER BY DATE DESC")
    else:
        self.cursor = self.conn.execute("SELECT * FROM tasks WHERE TAB='"+str(self.tab)+"' ORDER BY DATE DESC")
    self.refresh_task_list()
```

Task: A-Z

Another method of sorting the tasks is by the task name. As this is text, there are only two methods that this can be sorted; alphabetically A to Z or Z to A. I used a similar SQL statement to the other sorting methods where I changed the 'order by' parameter to 'task' which is the field containing the task name.

```python
def sort_3(self):
    self.conn = sqlite3.connect('tasks.db')
    self.tasks = []
    if self.tab == "Overview":
        self.cursor = self.conn.execute("SELECT * FROM tasks ORDER BY TASK ASC")
    else:
        self.cursor = self.conn.execute("SELECT * FROM tasks WHERE TAB='"+str(self.tab)+"' ORDER BY TASK ASC")
    self.refresh_task_list()
```

Task: Z-A

This is the opposite to the previous sorting method, with the exception that it will reverse the order in which the tasks are added to the task array.

```python
def sort_4(self):
    self.conn = sqlite3.connect('tasks.db')
    self.tasks = []
    if self.tab == "Overview":
        self.cursor = self.conn.execute("SELECT * FROM tasks ORDER BY TASK DESC")
    else:
        self.cursor = self.conn.execute("SELECT * FROM tasks WHERE TAB='"+str(self.tab)+"' ORDER BY TASK DESC")
    self.refresh_task_list()
```

Subject: A-Z

This follows the same concept as sorting the tasks alphabetically. This will sort the task's subject in alphabetical order.

```python
def sort_5(self):
    self.conn = sqlite3.connect('tasks.db')
    self.tasks = []
    if self.tab == "Overview":
        self.cursor = self.conn.execute("SELECT * FROM tasks ORDER BY SUBJECT ASC")
    else:
        self.cursor = self.conn.execute("SELECT * FROM tasks WHERE TAB='"+str(self.tab)+"' ORDER BY SUBJECT ASC")
    self.refresh_task_list()
```

Subject: Z-A

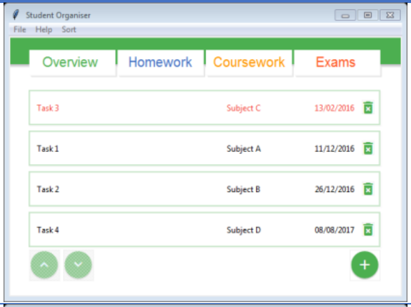
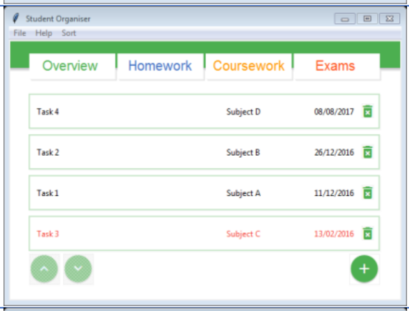
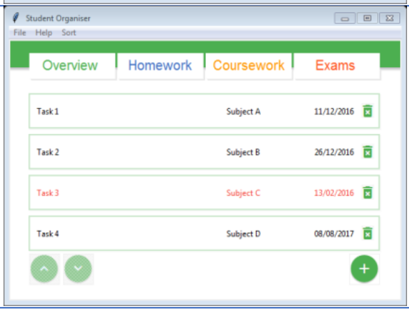
This is the opposite to the previous sorting method, with the exception that it will reverse the order in which the tasks are added to the task array.

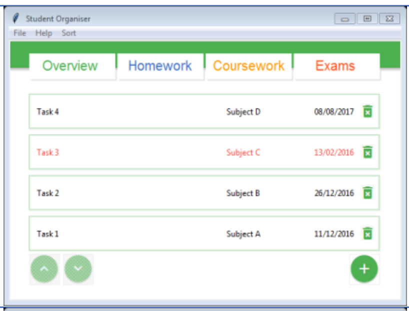
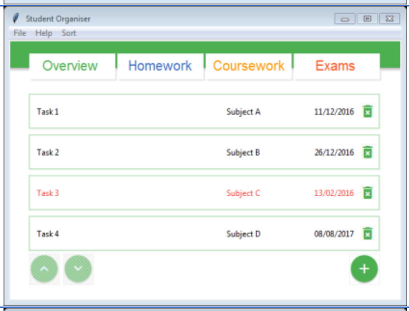
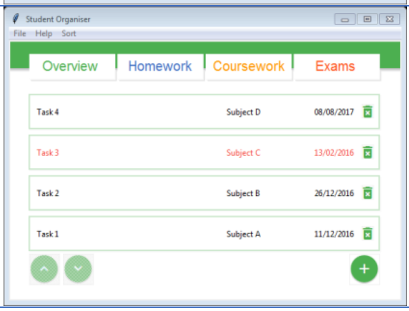
```python
def sort_4(self):
    self.conn = sqlite3.connect('tasks.db')
    self.tasks = []
    if self.tab == "Overview":
        self.cursor = self.conn.execute("SELECT * FROM tasks ORDER BY SUBJECT DESC")
    else:
        self.cursor = self.conn.execute("SELECT * FROM tasks WHERE TAB='"+str(self.tab)+"' ORDER BY SUBJECT DESC")
    self.refresh_task_list()
```

Refreshing Task List:

At the end of every sort function, another function is called, 'self.refresh_task_list().' This will load the records from the database in the order s0pecified within the sort function. Whilst accessing this data, it will append each piece of data to the tasks array so that they can be loaded in to the correct rows. As before, the data is reversed so that it is readable in the correct standard. Once the tasks array has been created with the relevant data, the 'load_rows' function is called as this uses this array to display the data.

```python
def refresh_task_list(self):
    for row in self.cursor:
        single = []
        single.append(row[1])
        single.append(row[2])
        # Reverse the date format (YYYY/MMM/DD => DD/MM/YYYY)
        dateData = (row[3].strip().split('/'))
        dateDay = dateData[2]
        dateMonth = dateData[1]
        dateYear = dateData[0]
        dateNew = dateDay + "/" + dateMonth + "/" + dateYear
        single.append(dateNew)
        single.append(row[4])
        single.append(row[0])
        self.tasks.append(single)
    self.conn.close()
    self.load_rows()
```

Test	Test Data	Expected Result	Actual Result / Evidence
Sort by task soonest	N/A	The program should show every task on a page with the task due soonest at the start.	 Student Organiser File Help Sort Overview Homework Coursework Exams Task 3 Subject C 13/02/2016 Task 1 Subject A 11/12/2016 Task 2 Subject B 26/12/2016 Task 4 Subject D 08/08/2017
Sort by task oldest	N/A	The program should show every task on a page with the task due soonest at the end.	 Student Organiser File Help Sort Overview Homework Coursework Exams Task 4 Subject D 08/08/2017 Task 2 Subject B 26/12/2016 Task 1 Subject A 11/12/2016 Task 3 Subject C 13/02/2016
Sort by task a-z	N/A	The program should show every task on a page with the tasks sorted alphabetically.	 Student Organiser File Help Sort Overview Homework Coursework Exams Task 1 Subject A 11/12/2016 Task 2 Subject B 26/12/2016 Task 3 Subject C 13/02/2016 Task 4 Subject D 08/08/2017

Sort by task z-a	N/A	The program should show every task on a page with the tasks sorted in reverse alphabetical order.	 Student Organiser File Help Sort Overview Homework Coursework Exams Task 4 Subject D 08/08/2017 Task 3 Subject C 13/02/2016 Task 2 Subject B 26/12/2016 Task 1 Subject A 11/12/2016
Sort by subject a-z	N/A	The program should show every task on a page with the tasks sorted by their subject alphabetically.	 Student Organiser File Help Sort Overview Homework Coursework Exams Task 1 Subject A 11/12/2016 Task 2 Subject B 26/12/2016 Task 3 Subject C 13/02/2016 Task 4 Subject D 08/08/2017
Sort by subject z-a	N/A	The program should show every task on a page with the tasks sorted by their subject in reverse alphabetical order.	 Student Organiser File Help Sort Overview Homework Coursework Exams Task 4 Subject D 08/08/2017 Task 3 Subject C 13/02/2016 Task 2 Subject B 26/12/2016 Task 1 Subject A 11/12/2016

Client Meeting:

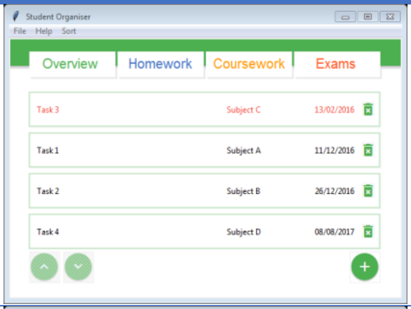
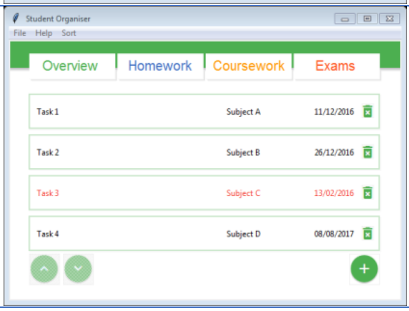
This is the second meeting with the client. My client has agreed that my sorting methods are very useful but would like an additional one to be added. The client had stated that they would like to be able to sort the tasks by the time in which they were added into the program. This should work by ordering the tasks with the most recently added record appearing first and the first task added should appear last.

Adjustments:

I added a new sorting method where the others have been written. I followed a similar structure to the others too. This will be the order in which the user has added the task to the program. As I have used an incrementing integer as the record ID for every task, I can sort the tasks by this value. By doing so, the tasks will be inserted into the array where the first task will be the first added by the user and the last will be the most resent task added in.

```python
def sort_7(self):
    self.conn = sqlite3.connect('tasks.db')
    self.tasks = []
    if self.tab == "Overview":
        self.cursor = self.conn.execute("SELECT * FROM tasks ORDER BY ID ASC")
    else:
        self.cursor = self.conn.execute("SELECT * FROM tasks WHERE TAB='"+str(self.tab)+"' ORDER BY ID ASC")
    self.refresh_task_list()
```

Testing of the Amendments:

Test	Test Data	Expected Result	Actual Result / Evidence
Open the program	N/A	The program should show every task on a page with the task due soonest at the top.	 Student Organiser File Help Sort Overview Homework Coursework Exams Task 3 Subject C 13/02/2016 Task 1 Subject A 11/12/2016 Task 2 Subject B 26/12/2016 Task 4 Subject D 08/08/2017
Sort by task added	N/A	The program should show every task on a page with the tasks sorted with the first inserted task appearing first.	 Student Organiser File Help Sort Overview Homework Coursework Exams Task 1 Subject A 11/12/2016 Task 2 Subject B 26/12/2016 Task 3 Subject C 13/02/2016 Task 4 Subject D 08/08/2017

The complete set of sorting methods becomes:

- Time: Soonest (Default)
- Time: Oldest
- Time: Task Added
- Task: A-Z
- Task: Z-A
- Subject: A-Z
- Subject: Z-A

Development Review:

This version has included the additional feature of sorting tasks. Originally, the program would sort tasks by their due date, task and subject. I wrote the functions to reorder the tasks before they were to be presented the user. This was mainly a slight variation of the SQL required to read the data from the database. Once all sorting options were accessible in the user interface, I trialled them with a sample of test data. Upon completion of these tests, I concluded that they all worked as expected.

To verify that these sorting methods were useful and if any adjustments were necessary, I met with a client. After a meeting with a client, I have added an additional sorting option to load the tasks by the time in which they were added. I developed this program further to meet their requirements and as a result, there are now a total of seven task sorting methods available to the user.

Therefore, I can state that the requirement that the program should offer a sorting system has been met to its fullest extent.

Delete Task:

Once tasks have been added, the user may want to remove them. To allow the user to delete individual tasks, I added in a delete icon inside every row. This button will delete the tasks beside it and refresh the rows to show the changes.

Each button is assigned a different function. These functions differ slightly depending on the row that the user selects to delete. Each function works in a similar way, just the code selecting the data to be deleted is different.

If the user deletes the first row, then the task ID is determined by the first item in the set array (the record in row 1) and the fifth piece of data inside this record (the ID). When the ID has been selected, I use a simple SQL statement to delete this task from the database. Once this task has been deleted and there was only one task on that page, the page counter is reduced by ‘2’ for the relevant tab. This ensures that the program will not show a blank page without any tasks showing. It will only show the pages with tasks on, the previous page.

Then another function is called, inside this function is a list of instructions that will run one after another. The idea of this would be to reduce repeated code by allowing just a single function to be called per row deleted, rather than three instructions time. The first of the three will commit any changes to the database, the task will be deleted ad saved. The connection to the database will be closed as it is no longer needed. Finally, the ‘setup_data’ function is called to reload the tasks in the program, ensuring that the deleted task will not show in the row.

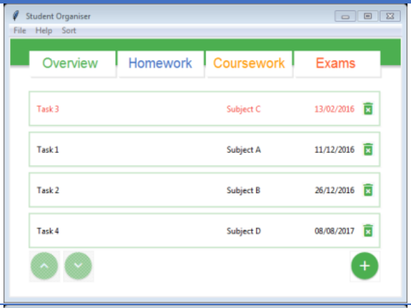
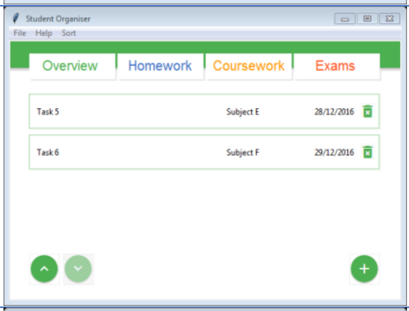
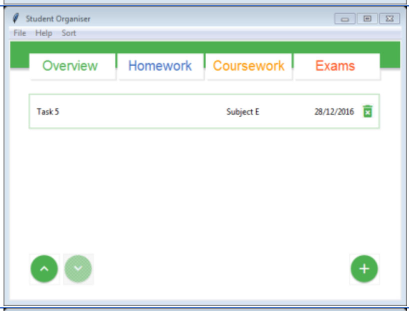
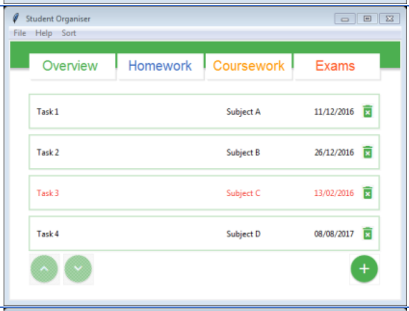
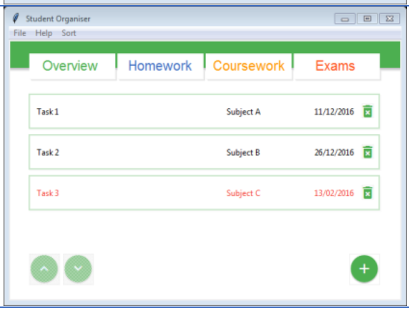
```python
def delete_row_1(self):
    self.task_id = self.set[0][4]
    self.conn = sqlite3.connect('tasks.db')
    self.conn.execute("DELETE from TASKS where ID='"+ str(self.task_id)+"';")
    if self.fullRows == 1:
        if self.tab == "Overview":
            self.ovpage -= 2
        elif self.tab == "Homework":
            self.hwpage -= 2
        elif self.tab == "Coursework":
            self.cwpage -= 2
        elif self.tab == "Exam":
            self.expage -= 2
    self.refresh_task_list_remove()
```

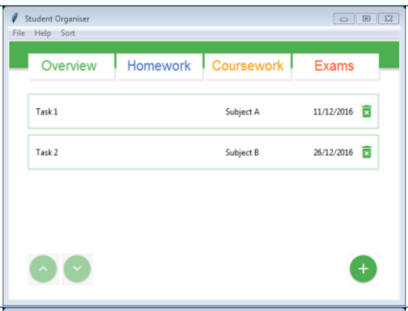
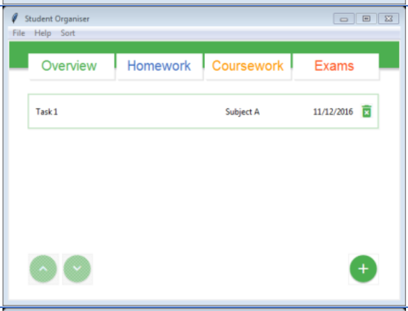
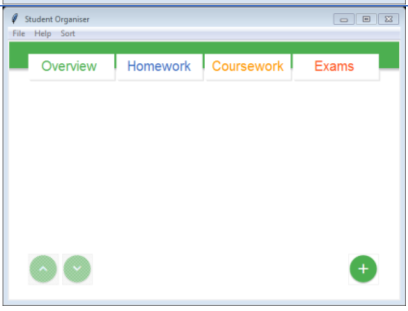
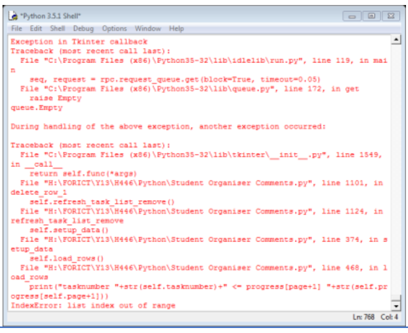
For every other row, deleting the data is very similar, only the task ID is determined with the current row in which it is positioned. There is also no need to reduce the page counter when deleting a task which is in row two, three or four. This is because there will always be at least one remaining task on the page once it has been removed.

```python
def delete_row_2(self):
    self.task_id = self.set[1][4]
    self.conn = sqlite3.connect('tasks.db')
    self.conn.execute("DELETE from TASKS where ID='"+ str(self.task_id)+"';")
    self.refresh_task_list_remove()
```

```python
def delete_row_3(self):
    self.task_id = self.set[2][4]
    self.conn = sqlite3.connect('tasks.db')
    self.conn.execute("DELETE from TASKS where ID='"+ str(self.task_id)+"';")
    self.refresh_task_list_remove()
```

```
def delete_row_4(self):
    self.task_id = self.set[3][4]
    self.conn = sqlite3.connect('tasks.db')
    self.conn.execute("DELETE from TASKS where ID='"+ str(self.task_id)+"';")
    self.refresh_task_list_remove()
```

Test	Test Data	Expected Result	Actual Result / Evidence
Add six tasks	N/A	The program should show every task on a page with the task due soonest at the top.	 Student Organiser File Help Sort Overview Homework Coursework Exams Task 3 Subject C 13/02/2016 Task 1 Subject A 11/12/2016 Task 2 Subject B 26/12/2016 Task 4 Subject D 08/08/2017
Select the second page	N/A	The program should show tasks 5 and 6 on a page.	 Student Organiser File Help Sort Overview Homework Coursework Exams Task 5 Subject E 28/12/2016 Task 6 Subject F 29/12/2016
Delete task 6	N/A	The program should show tasks 5 on a page.	 Student Organiser File Help Sort Overview Homework Coursework Exams Task 5 Subject E 28/12/2016
Delete task 5	N/A	The program should go to the previous page and show tasks 1,2,3 and 4.	 Student Organiser File Help Sort Overview Homework Coursework Exams Task 1 Subject A 11/12/2016 Task 2 Subject B 26/12/2016 Task 3 Subject C 13/02/2016 Task 4 Subject D 08/08/2017
Delete task 4	N/A	The program should show tasks 1,2 anfirstd 3 on a page.	 Student Organiser File Help Sort Overview Homework Coursework Exams Task 1 Subject A 11/12/2016 Task 2 Subject B 26/12/2016 Task 3 Subject C 13/02/2016

Delete task 3	N/A	The program should show tasks 1 and 2 on a page.	 Task 1 Subject A 11/12/2016; Task 2 Subject B 26/12/2016
Delete task 2	N/A	The program should show tasks 1 on a page.	 Task 1 Subject A 11/12/2016
Delete task 1	N/A	The program should show no tasks on the page.	 

```
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Program Files (x86)\Python35-32\lib\idlelib\run.py", line 119, in mai
n
    seq, request = rpc.request_queue.get(block=True, timeout=0.05)
  File "C:\Program Files (x86)\Python35-32\lib\queue.py", line 172, in get
    raise Empty
queue.Empty

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "C:\Program Files (x86)\Python35-32\lib\tkinter\__init__.py", line 1549,
in __call__
    return self.func(*args)
  File "H:\FORICT\Y13\H446\Python\Student Organiser Comments.py", line 1101, in
delete_row_1
    self.refresh_task_list_remove()
  File "H:\FORICT\Y13\H446\Python\Student Organiser Comments.py", line 1124, in
refresh_task_list_remove
    self.setup_data()
  File "H:\FORICT\Y13\H446\Python\Student Organiser Comments.py", line 374, in s
etup_data
    self.load_rows()
  File "H:\FORICT\Y13\H446\Python\Student Organiser Comments.py", line 468, in l
oad_rows
    print("tasknumber "+str(self.tasknumber)+" <= progress[page+1] "+str(self.pr
ogress[self.page+1]))
IndexError: list index out of range
```

The error displayed in the final screenshot shows an issue caused when deleting the final task from the program. The issue is caused when the program reduces the page count although there are no tasks remaining. To fix this, I need to ensure that the page count will remain at '0' when removing the final task and a message should be shown that there are no tasks to be presented.

To fix this, I adjusted the function that deleted row 1 from the database. I added in an additional embedded 'if statement' that would only reduce the page count if the current page number was greater than '0', else the page number will remain the same, '0' and will pass.

```python
def delete_row_1(self):
    self.task_id = self.set[0][4]
    self.conn = sqlite3.connect('tasks.db')
    self.conn.execute("DELETE from TASKS where ID='"+ str(self.task_id)+"';")
    if self.page > 0:
        if self.fullRows == 1:
            if self.tab == "Overview":
                self.ovpage -= 2
            elif self.tab == "Homework":
                self.hwpage -= 2
            elif self.tab == "Coursework":
                self.cwpage -= 2
            elif self.tab == "Exam":
                self.expage -= 2
    else:
        pass
    self.refresh_task_list_remove()
```

By doing so, the program will not adjust the page number, remove the task and call the function, 'load_rows.' Within this function, I adjusted a small piece of code that will run when the number of tasks is equal to zero. If this is true, then a piece of text is inserted into the canvas stating, "No tasks to display." This shows the user that there are no tasks present in the program. I have added a tag to this object, 'noData' as I can easily delete any widget with this task later on in the program.

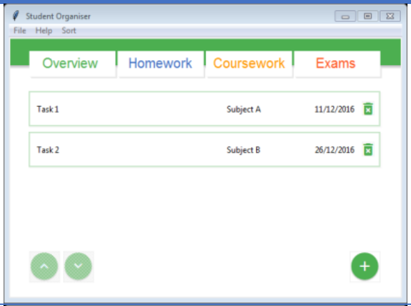
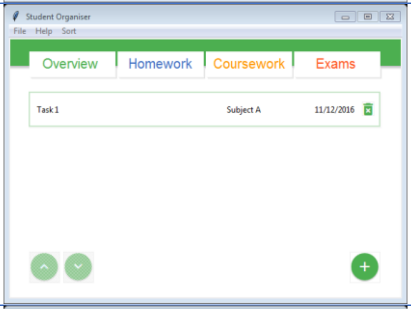
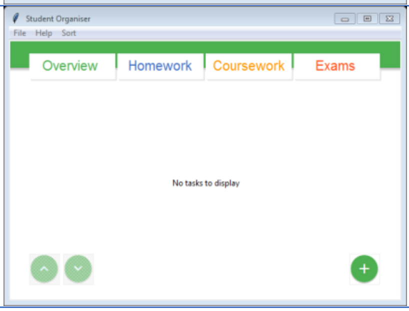
```python
if self.tasknumber == 0:
    self.canvas.create_text(290, 210, text="No tasks to display", fill="black",
tags="noData")
    return
```

I removed every widget with the tag of 'noData' when there was at least one task to display. By doing so, the text informing the user that there are no tasks to display is therefore removed and the data is loaded in.

```python
# If a page isn't full, use this array
elif self.tasknumber<=self.progress[self.page+1] and
self.tasknumber>self.progress[self.page]:
    self.canvas.delete("noData")
```

```python
# If all pages are full, use this array
else:
    self.canvas.delete("noData")
```

Testing of the Amendments:

Test	Test Data	Expected Result	Actual Result / Evidence
Add two tasks	N/A	The program should show every task on a page with the task due soonest at the top.	 Student Organiser File Help Sort Overview Homework Coursework Exams Task 1 Subject A 11/12/2016 Task 2 Subject B 26/12/2016
Delete task 2	N/A	The program should show tasks 1 on a page.	 Student Organiser File Help Sort Overview Homework Coursework Exams Task 1 Subject A 11/12/2016
Delete task 1	N/A	The program should show no tasks on the page with the text, No tasks to display."	 Student Organiser File Help Sort Overview Homework Coursework Exams No tasks to display

Development Review:

The addition of code since the last version is the addition of the delete button. The aim of this sprint was to target the requirement that tasks must be able to be deleted.

During this development, I had written functions to remove tasks from the database when a task ID had been given. This function would be called by use of a button assigned to each visible task on display.

Once the solution had been coded, I conducted a series of tests to ensure that the feature worked as intended. However, I found an error occurred when deleting the final task from the page. I then fixed this issue and repeated the tests with success.

Client Meeting:

This is the third meeting with the client. My client has suggested that I add a page numbering system to the program. At the centre, in grey text, there should be a page numbering system which informs the user which page of tasks that is currently loaded. The notation requested was, 'Page x of y.'

Adjustments:

To position text on the canvas when a new page loads, I would need to place the necessary code within the load_rows function. This function is called every time the page changes, whether it is to another tab or scrolling up or down, making this an ideal position to insert the page counter.

Before I could insert the text, I would need to use some variables that I had previously used to determine the page numbers. To get the current page that the user was currently viewing, I used the self.page value. This value is used when loading the correct set of tasks into the rows on the pages. This value is also twice the amount of the page that the user is currently on, therefore I performed a DIV of this value. The total number of pages was calculated by using the self.sets variable. This is used when sorting the tasks ready for displaying them in the rows. A set can contain up to four tasks. Each task is split across into groups of up to four and each set is a separate page. Therefore, the total number of sets can be considered the maximum number of pages available in any given tab.

To make up the text that will be presented to the user, I created a variable called 'self.pageText' which included, 'Page ' followed by the current page, then, ' of ' followed by the maximum pages.

```
self.pageText="Page "+str((self.page//2)+1)+" of "+str(self.sets)
```

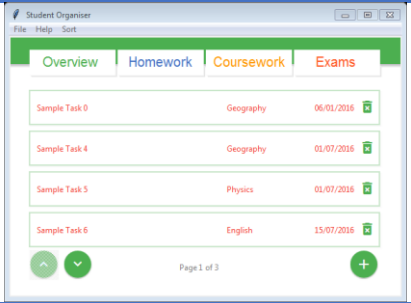
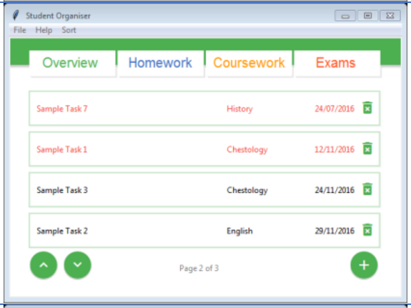
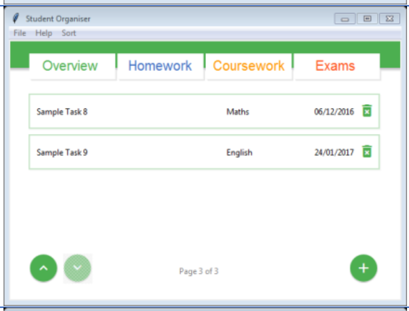
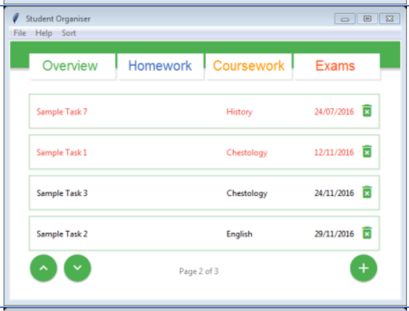
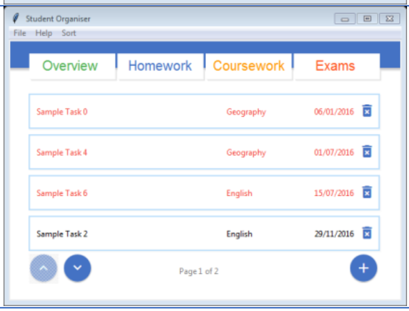
Once the text that will be displayed has been constructed into a single variable, I wrote some code that will output this data onto the canvas with the client's preferences. I positioned it in the center of the screen at the bottom with the coordinates of 280,340. I had also altered the text colour top grey too. The final addition that I included was the tags at the end.

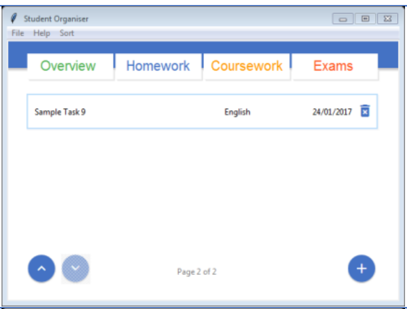
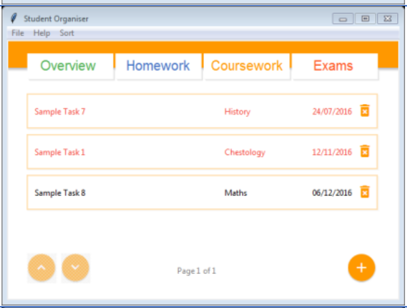
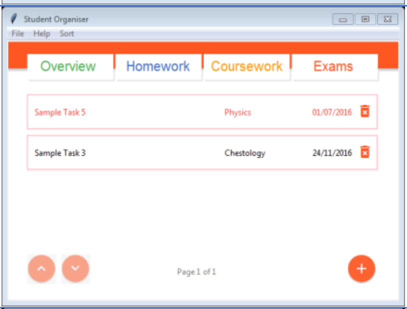
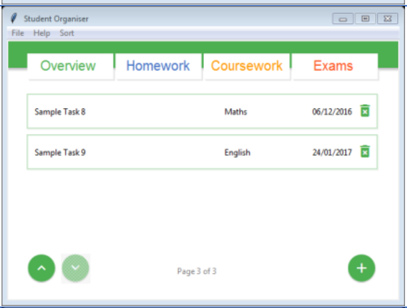
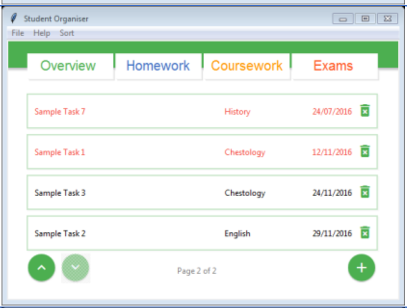
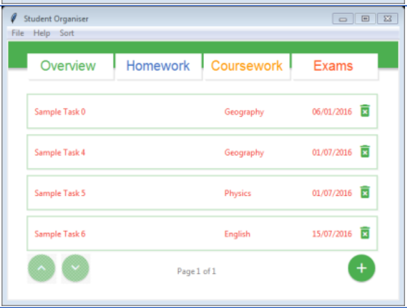
```
self.canvas.create_text(280, 340, text=self.pageText, fill="grey", tags="page")
```

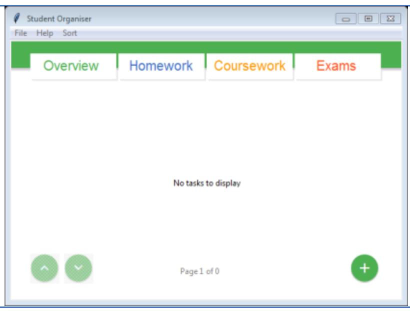
I chose to include the tag of "page" as I can easily delete all objects containing the tag of "Page" at any point in this program. This is very important, as I will delete any objects with this tag at the start of the load_pages function. This will ensure that the text displayed will always be the latest value. This will remove the issue that the page number text will overlap each other every time the page is changed, as the previous text is not removed from the canvas.

```
self.canvas.delete("page")
```

Testing of the Amendments:

Test	Test Data	Expected Result	Actual Result / Evidence
Create a set of 10 tasks (5 homework, 3 coursework and 2 of exam type) and select overview tab	N/A	The program should show every task on a page with the text, 'Page 1 of 3.'	 Sample Task 0 Geography 06/01/2016 Sample Task 4 Geography 01/07/2016 Sample Task 5 Physics 01/07/2016 Sample Task 6 English 15/07/2016 Page 1 of 3
Scroll down a page	N/A	The program should show the second page of results with the text, 'Page 2 of 3.'	 Sample Task 7 History 24/07/2016 Sample Task 1 Chestology 12/11/2016 Sample Task 3 Chestology 24/11/2016 Sample Task 2 English 29/11/2016 Page 2 of 3
Scroll down a page	N/A	The program should show the second page of results with the text, 'Page 3 of 3.'	 Sample Task 8 Maths 06/12/2016 Sample Task 9 English 24/01/2017 Page 3 of 3
Scroll up a page	N/A	The program should show the second page of results with the text, 'Page 2 of 3.'	 Sample Task 7 History 24/07/2016 Sample Task 1 Chestology 12/11/2016 Sample Task 3 Chestology 24/11/2016 Sample Task 2 English 29/11/2016 Page 2 of 3
Select the homework tab	N/A	The program should show the homework tasks with the text, 'Page 1 of 2.'	 Sample Task 0 Geography 06/01/2016 Sample Task 4 Geography 01/07/2016 Sample Task 6 English 15/07/2016 Sample Task 2 English 29/11/2016 Page 1 of 2

Scroll down a page	N/A	The program should show the second page of results with the text, 'Page 2 of 2.'	 Student Organiser File Help Sort Overview Homework Coursework Exams Sample Task 9 English 24/01/2017 Page 2 of 2
Select the coursework tab	N/A	The program should show the homework tasks with the text, 'Page 1 of 1.'	 Student Organiser File Help Sort Overview Homework Coursework Exams Sample Task 7 History 24/07/2016 Sample Task 1 Chestology 12/11/2016 Sample Task 8 Maths 06/12/2016 Page 1 of 1
Select the exams tab	N/A	The program should show the homework tasks with the text, 'Page 1 of 1.'	 Student Organiser File Help Sort Overview Homework Coursework Exams Sample Task 5 Physics 01/07/2016 Sample Task 3 Chestology 24/11/2016 Page 1 of 1
Select the overview tab	N/A	The program should show the final 2 tasks on a page with the text, 'Page 3 of 3.'	 Student Organiser File Help Sort Overview Homework Coursework Exams Sample Task 8 Maths 06/12/2016 Sample Task 9 English 24/01/2017 Page 3 of 3
Select the overview tab and delete 2 tasks	N/A	The program should show the tasks with the text, 'Page 1 of 2.'	 Student Organiser File Help Sort Overview Homework Coursework Exams Sample Task 7 History 24/07/2016 Sample Task 1 Chestology 12/11/2016 Sample Task 3 Chestology 24/11/2016 Sample Task 2 English 29/11/2016 Page 2 of 2
Delete a further 4 tasks	N/A	The program should show the tasks with the text, 'Page 1 of 1.'	 Student Organiser File Help Sort Overview Homework Coursework Exams Sample Task 0 Geography 06/01/2016 Sample Task 4 Geography 01/07/2016 Sample Task 5 Physics 01/07/2016 Sample Task 6 English 15/07/2016 Page 1 of 1

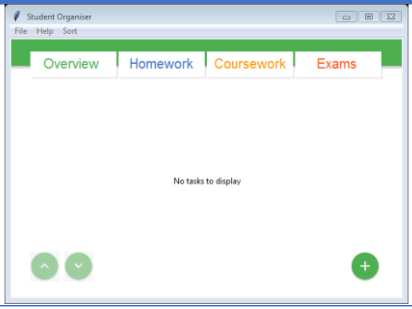
Delete a further 4 tasks	N/A	The program should not show a page counter.	 Student Organiser File Help Sort Overview Homework Coursework Exams No tasks to display Page 1 of 0

As a result of this testing, I have found that the page numbering system is working as expected where the correct number of pages is calculated. When the page or tab is changed, the page number is calculated correctly. As well as this, if all tasks are removed from a page, the page is scrolled up automatically to the tasks before and the page numbers are updated.

If there are no tasks available on a page, I planned for the page counter to not be displayed. However, it shows, 'Page 1 of 0.' This is correct, however it should be removed as there are no pages to load and the statement 1 of 0 is not logical. To fix this issue, I used the same code to delete every object with the tag of 'page' in the same load_rows function within an 'if statement.' Where there are no tasks, the, 'No tasks to display' message is presented to the user. I placed the code blow this so that when there are no tasks, the page counter is deleted and the user should never see this.

```
# If there are no tasks to display:
#     Show the text 'No tasks to display'
#     Remove the page counter from the page
if self.tasknumber == 0:
    self.canvas.delete("page")
    self.canvas.create_text(290, 210, text="No tasks to display", fill="black", tags="noData")
    return
```

Testing of the Amendments:

Test	Test Data	Expected Result	Actual Result / Evidence
Load a page with zero tasks.	N/A	The program should show the message, "No tasks to display" and the page counter should not be present.	 Student Organiser File Help Sort Overview Homework Coursework Exams No tasks to display

The outcome is now as expected meaning that the additional code has worked.

Modify Tasks:

The user is able to add in tasks and delete them but unable to modify any tasks currently written in the program. I will now be writing in the code that will allow the user to modify existing tasks that have been created in this program.

To start, I will need to collect the ID for each task that will be present on any given page. This is vital information, as I will need to use this when removing the old tasks from the database. To do this, I created an array for the task ID when each page is loaded. Another piece of data that I will need is the task type as this will be used when writing the new task to the database. It is important that the same task type is used. These arrays are created within the load_rows function.

If there is a single full row, the ID and task type is appended to two individual arrays respectively.

```
# Append the ID of every task on the page to an array
self.sets_id=[]
self.sets_id.append(row1[4])

# Append the task type of every task on the page to an array
self.sets_type=[]
self.sets_type.append(row1[3])
```

If there are two tasks on the page, then the following applies.

```
# Append the ID of every task on the page to an array
self.sets_id=[]
self.sets_id.append(row1[4])
self.sets_id.append(row2[4])

# Append the task type of every task on the page to an array
self.sets_type=[]
self.sets_type.append(row1[3])
self.sets_type.append(row2[3])
```

If there are three tasks on the page, then the following applies.

```
# Append the ID of every task on the page to an array
self.sets_id=[]
self.sets_id.append(row1[4])
self.sets_id.append(row2[4])
self.sets_id.append(row3[4])

# Append the task type of every task on the page to an array
self.sets_type=[]
self.sets_type.append(row1[3])
self.sets_type.append(row2[3])
self.sets_type.append(row3[3])
```

If there is a full set of tasks on the page, then the following applies.

```
# Append the ID of every task on the page to an array
self.sets_id=[]
self.sets_id.append(row1[4])
self.sets_id.append(row2[4])
self.sets_id.append(row3[4])
self.sets_id.append(row4[4])

# Append the task type of every task on the page to an array
self.sets_type=[]
self.sets_type.append(row1[3])
self.sets_type.append(row2[3])
self.sets_type.append(row3[3])
self.sets_type.append(row4[3])
```

Once these two arrays have been created, I created a new function that is called, save_changes. This function will be involved in saving the changes to the tasks. Once called, it checks if there are zero tasks in the database. If so, it should return and no changes should occur, as there are no tasks to modify. Then the content of every entry box within the tasks page is assigned a variable. These variables are for the task name, subject and due date. Each row and column has its own unique variable.

```
def save_changes(self):
    # If there are no tasks present, return
    if self.tasknumber == 0:
        return

    # Get data from the rows
    t1n=self.name_1.get()
    t1s=self.subject_1.get()
    t1d=self.date_1.get()
    t2n=self.name_2.get()
    t2s=self.subject_2.get()
    t2d=self.date_2.get()
    t3n=self.name_3.get()
    t3s=self.subject_3.get()
    t3d=self.date_3.get()
    t4n=self.name_4.get()
    t4s=self.subject_4.get()
    t4d=self.date_4.get()
```

Then, I will open a connection to the database in preparation for updating the data.

```
    self.conn = sqlite3.connect('tasks.db')
```

The next part of this function is reliant on the number of tasks on the page. If there is not a full set of tasks (less than four but greater than zero) then the number of full rows is calculated.

```
    # If a page isn't full, use this array
    if self.tasknumber<=self.progress[self.page+1] and
self.tasknumber>self.progress[self.page]:
        self.fullRows = self.tasknumber - self.progress[self.page]
```

If there is only a single task present on the page then, the date of this task is reversed ready for insertion into the database. The ID of the last row in the database is calculated and the task name, subject and date is inserted into the database. Then the corresponding record in the first row is removed from the database. This gives the effect that the task has been updated as it has been replaced with a task of newer data modified by the user.

```python
        if self.fullRows == 1:
            date1=t1d.split('/')
            t1d=date1[2]+"/"+date1[1]+"/"+date1[0]
            lastRow = self.conn.execute("SELECT MAX(id) FROM TASKS")
            for row in lastRow:
                self.id = row[0]
            self.conn.execute("INSERT INTO TASKS(ID, TASK, SUBJECT, DATE, TAB) values (?, ?, ?, ?, ?)", (self.id+1, t1n, t1s, t1d, self.sets_type[0]))
            self.conn.execute("DELETE from TASKS where ID='"+str(self.sets_id[0])+"';")
```

If there is only are two tasks present on the page than, a similar process will occur that will update two tasks.

```python
        if self.fullRows == 2:
            date1=t1d.split('/')
            date2=t2d.split('/')
            t1d=date1[2]+"/"+date1[1]+"/"+date1[0]
            t2d=date2[2]+"/"+date2[1]+"/"+date2[0]
            lastRow = self.conn.execute("SELECT MAX(id) FROM TASKS")
            for row in lastRow:
                self.id = row[0]
            self.conn.execute("INSERT INTO TASKS(ID, TASK, SUBJECT, DATE, TAB) values (?, ?, ?, ?, ?)", (self.id+1, t1n, t1s, t1d, self.sets_type[0]))
            self.conn.execute("INSERT INTO TASKS(ID, TASK, SUBJECT, DATE, TAB) values (?, ?, ?, ?, ?)", (self.id+2, t2n, t2s, t2d, self.sets_type[1]))
            self.conn.execute("DELETE from TASKS where ID='"+str(self.sets_id[0])+"';")
            self.conn.execute("DELETE from TASKS where ID='"+str(self.sets_id[1])+"';")

```

If there is only are three tasks present on the page than, a similar process will occur that will update all three tasks.

```python
        if self.fullRows == 3:
            date1=t1d.split('/')
            date2=t2d.split('/')
            date3=t3d.split('/')
            t1d=date1[2]+"/"+date1[1]+"/"+date1[0]
            t2d=date2[2]+"/"+date2[1]+"/"+date2[0]
            t3d=date3[2]+"/"+date3[1]+"/"+date3[0]
            lastRow = self.conn.execute("SELECT MAX(id) FROM TASKS")
            for row in lastRow:
                self.id = row[0]
            self.conn.execute("INSERT INTO TASKS(ID, TASK, SUBJECT, DATE, TAB) values (?, ?, ?, ?, ?)", (self.id+1, t1n, t1s, t1d, self.sets_type[0]))
            self.conn.execute("INSERT INTO TASKS(ID, TASK, SUBJECT, DATE, TAB) values (?, ?, ?, ?, ?)", (self.id+2, t2n, t2s, t2d, self.sets_type[1]))
            self.conn.execute("INSERT INTO TASKS(ID, TASK, SUBJECT, DATE, TAB) values (?, ?, ?, ?, ?)", (self.id+3, t3n, t3s, t3d, self.sets_type[2]))
            self.conn.execute("DELETE from TASKS where ID='"+str(self.sets_id[0])+"';")
            self.conn.execute("DELETE from TASKS where ID='"+str(self.sets_id[1])+"';")
            self.conn.execute("DELETE from TASKS where ID='"+str(self.sets_id[2])+"';")
```

If there is a full set of tasks, there is no need to calculate the number of tasks in the page as every row is full and can be assumed that all data in the current page should be updated. This is a similar process to the previous three, with the exception that all four rows are involved.

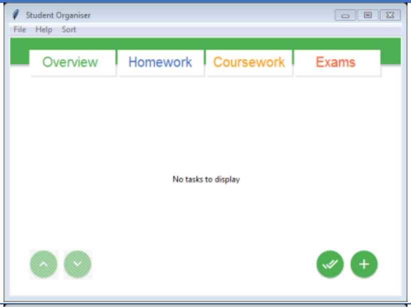
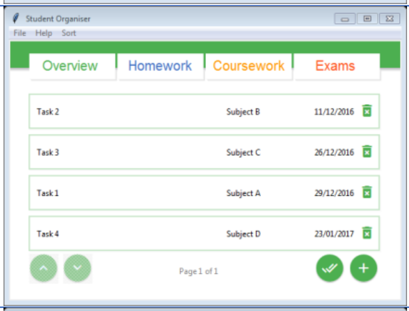
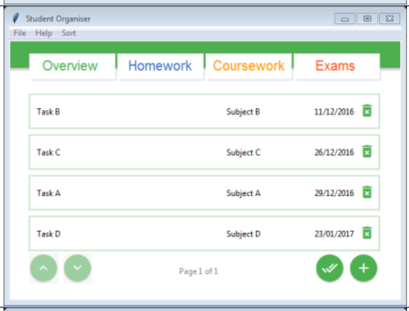
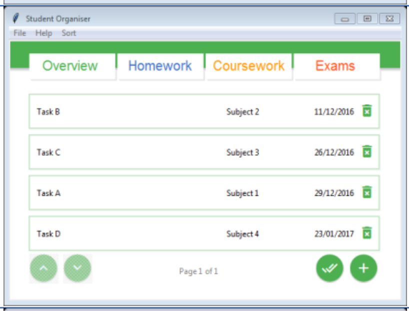
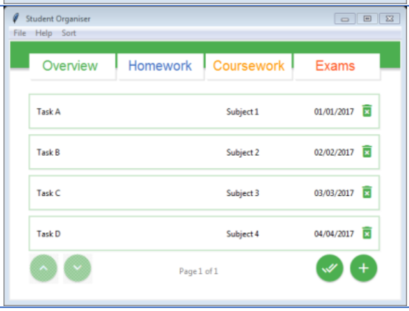
```
    else:
        # Rearrange dates from the rows
        date1=t1d.split('/')
        date2=t2d.split('/')
        date3=t3d.split('/')
        date4=t4d.split('/')
        t1d=date1[2]+"/"+date1[1]+"/"+date1[0]
        t2d=date2[2]+"/"+date2[1]+"/"+date2[0]
        t3d=date3[2]+"/"+date3[1]+"/"+date3[0]
        t4d=date4[2]+"/"+date4[1]+"/"+date4[0]
        # Add current data stored in all entry boxes to the database
        lastRow = self.conn.execute("SELECT MAX(id) FROM TASKS")
        for row in lastRow:
            self.id = row[0]
        self.conn.execute("INSERT INTO TASKS(ID, TASK, SUBJECT, DATE, TAB) values
(?, ?, ?, ?, ?)", (self.id+1, t1n, t1s, t1d, self.sets_type[0]))
        self.conn.execute("INSERT INTO TASKS(ID, TASK, SUBJECT, DATE, TAB) values
(?, ?, ?, ?, ?)", (self.id+2, t2n, t2s, t2d, self.sets_type[1]))
        self.conn.execute("INSERT INTO TASKS(ID, TASK, SUBJECT, DATE, TAB) values
(?, ?, ?, ?, ?)", (self.id+3, t3n, t3s, t3d, self.sets_type[2]))
        self.conn.execute("INSERT INTO TASKS(ID, TASK, SUBJECT, DATE, TAB) values
(?, ?, ?, ?, ?)", (self.id+4, t4n, t4s, t4d, self.sets_type[3]))

        # Remove the old records from the database
        self.conn.execute("DELETE from TASKS where ID='"+
str(self.sets_id[0])+"';")
        self.conn.execute("DELETE from TASKS where ID='"+
str(self.sets_id[1])+"';")
        self.conn.execute("DELETE from TASKS where ID='"+
str(self.sets_id[2])+"';")
        self.conn.execute("DELETE from TASKS where ID='"+
str(self.sets_id[3])+"';")
```

Once the new data has been written to the database and the old data removed, it is required that the page is updated. This makes use of a function that I had created previously when adding the delete function. The following function will commit any changes made to the database, close the connection to the database and finally call the setup_data function. This function reads all the tasks from the database and adds these to the pages, giving the effect that the changes have saved and the new data is now presented to the user.

```
        self.refresh_task_list_remove()
```

To use this function, it must be called upon the user's request. I introduced a new button into the program that will allow the user to save the modifications of the tasks on request. I chose to add this button in the lower right next to the add task button. I chose this location because the aesthetics would appear symmetrical with two circular buttons that control the page s to the left and two circular buttons to the right with the page counter in the centre. The code that I have written below will add the button with the new 'double tick' icon in the correct location on the page. The code fpor this button will be written inside the load_rows function.

```
button_save = Button(self, image=self.save, command=self.save_changes, anchor = W)
button_save.configure(border=0, relief = FLAT, highlightthickness=0)
button_save_window = self.canvas.create_window(452, 315, anchor=NW,
window=button_save)
```

Test	Test Data	Expected Result	Actual Result / Evidence
Open the program	N/A	The program should show the overview tab with the new button in place.	 Student Organiser File Help Sort Overview Homework Coursework Exams No tasks to display
Add 4 tasks to the program	N/A	The program should show the three tasks on the page.	 Student Organiser File Help Sort Overview Homework Coursework Exams Task 2 Subject B 11/12/2016 Task 3 Subject C 26/12/2016 Task 1 Subject A 29/12/2016 Task 4 Subject D 23/01/2017 Page 1 of 1
Edit the Name of every task and press the save button	Task A Task B Task C Task D	All tasks will be rewritten to the database with their new names and should be presented to the user.	 Student Organiser File Help Sort Overview Homework Coursework Exams Task B Subject B 11/12/2016 Task C Subject C 26/12/2016 Task A Subject A 29/12/2016 Task D Subject D 23/01/2017 Page 1 of 1
Edit the Subject of every task and press the save button	Subject 1 Subject 2 Subject 3 Subject 4	All tasks will be rewritten to the database with their new subjects and should be presented to the user.	 Student Organiser File Help Sort Overview Homework Coursework Exams Task B Subject 2 11/12/2016 Task C Subject 3 26/12/2016 Task A Subject 1 29/12/2016 Task D Subject 4 23/01/2017 Page 1 of 1
Edit the date of every task and press the save button	01/01/2017 02/02/2017 03/03/2017 04/04/2017	All tasks will be rewritten to the database with their new dates and should be presented to the user.	 Student Organiser File Help Sort Overview Homework Coursework Exams Task A Subject 1 01/01/2017 Task B Subject 2 02/02/2017 Task C Subject 3 03/03/2017 Task D Subject 4 04/04/2017 Page 1 of 1

Development Review:

This iteration focused upon adding the option to modify tasks. Any task that has been added to the program must be able to be adjusted where appropriate as outlined in my success criteria.

My initial interpretation of this task was to use the main tasks page as an input too. I used the entry boxes available to allow modifications to the data presented. This worked by adding in a save button that once clicked would take any data in these entry boxes and replacing the tasks on the page with the new data.

After testing every field available, the tasks were updated successfully.

Information Section:

This program is now capable of storing, sorting, modifying and deleting tasks. As this is designed to replace a school planner, there must be more functions to this software. The next addition that I plan to make to this program will be adding in access to any important information for the students.

I have chosen to add an information tab to the menu bar along the top of the program as this can be accessed anywhere within the program. This button can drop down a list of other buttons which can be used to link to other sections of the program. In the Information menu, I would like to include links to a school map, contact details, useful websites and a notes page. Each of these features are available in a standard school planner and are essential features that should be included in this software.

Adding the Information Tab:

First, I will write the code that will display an extra button in the menu bar with all relevant sub menu options. Here, I have created all function names that they will call once clicked. I will later create these functions to perform the appropriate actions.

```
ddata = Menu(menu_bar, tearoff=0)
menu_bar.add_cascade(label="Information", menu=ddata)
menu_map = Menu(self)
menu_map.add_command(label="St George's Academy", command=self.map_sga)
menu_map.add_command(label="Carre's Grammar School", command=self.map_cgs)
ddata.add_cascade(label="School Maps", menu=menu_map)
ddata.add_command(label="Contact Details", command=self.contact)
ddata.add_command(label="Useful Websites", command=self.websites)
ddata.add_separator()
ddata.add_command(label="Notes", command=self.notes)
```

Once this has been created, I will now need to create the following functions:

map_sga	St George’s Academy Map
map_cgs	Carre’s Grammar School Map
contact	School contact details
websiites	Useful websites
notes	Notes

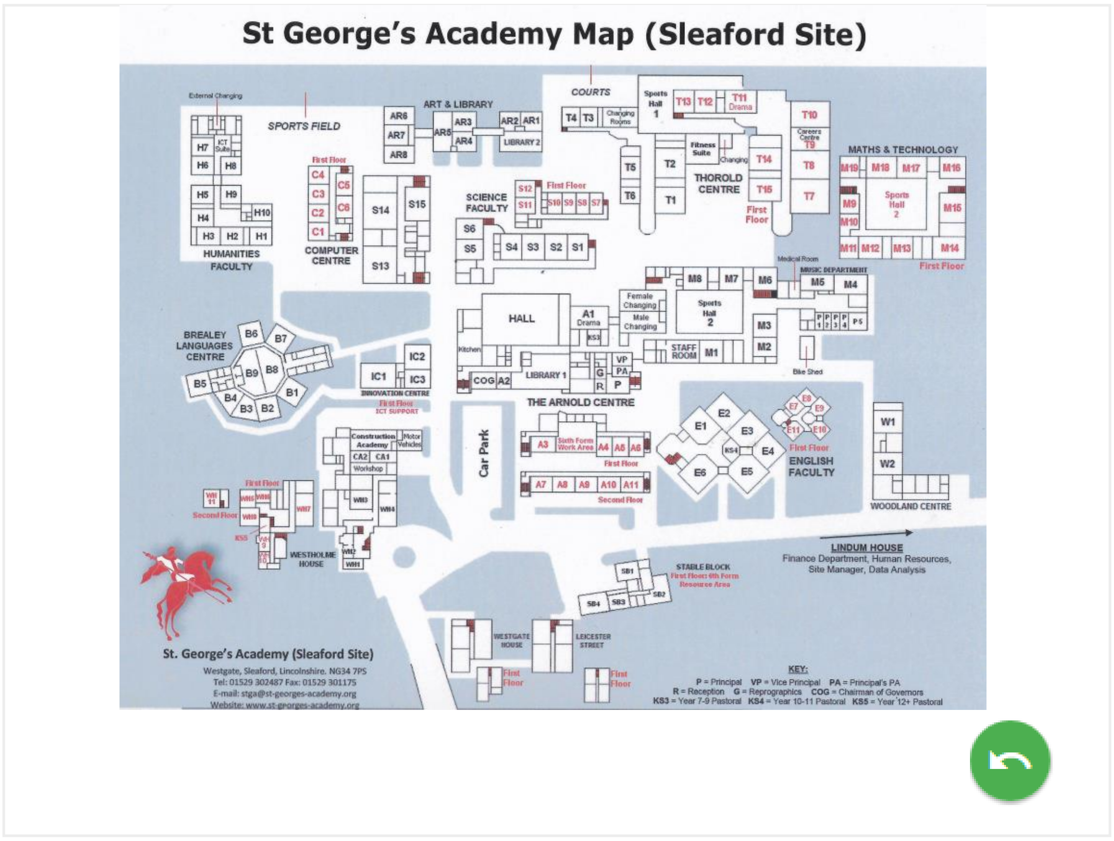
School Map:

I will be creating a school map page for two schools as the two that I attend are in a Joint Sixth Form. Each map for the school will have its own separate page. I will create a back button that will take the place similar to that of the add task button when tasks are displayed. This back button should return the user to the tasks page, the home screen. Each page will use an image of the school map, taking the majority of the space provided.

I have created four coloured back buttons that I will be using in these pages:

St George’s Academy Map:

A plan of how I would like this map to appear in the program.

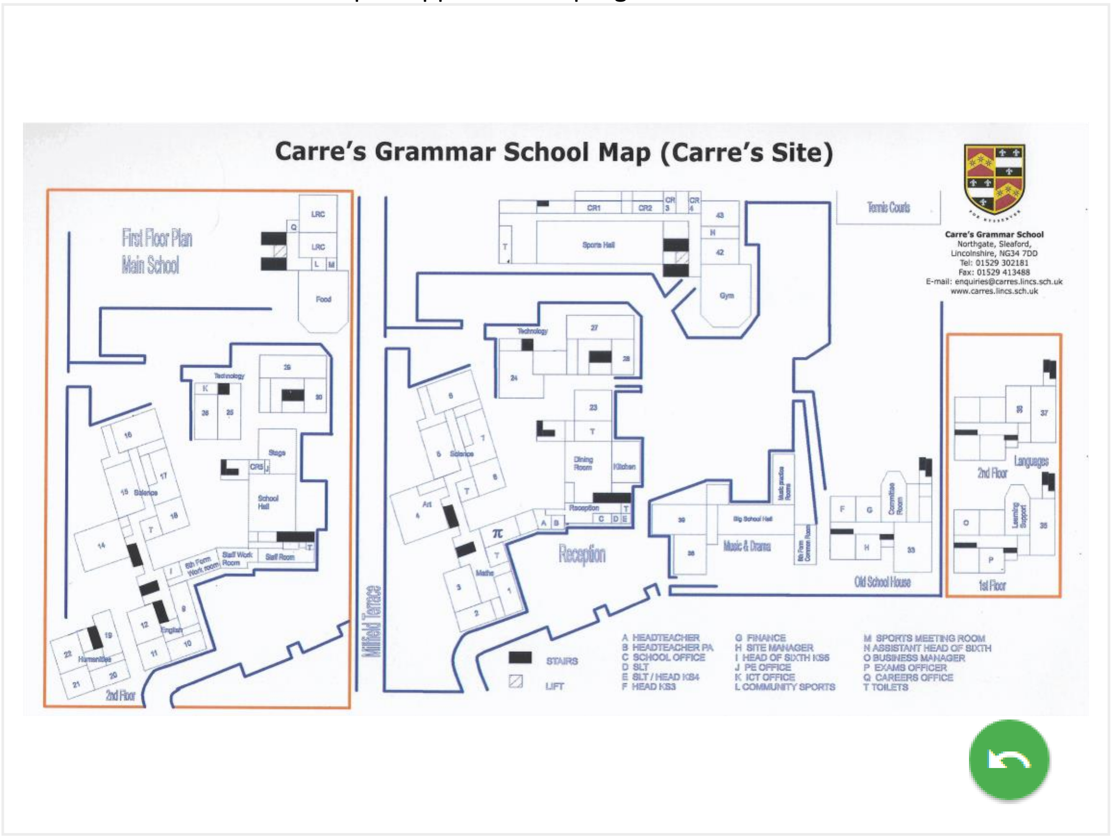


St George’s Academy Map (Sleaford Site)
SPORTS FIELD
ART & LIBRARY
COURTS
HUMANITIES FACULTY
COMPUTER CENTRE
SCIENCE FACULTY
THOROLD CENTRE
MATHS & TECHNOLOGY
MUSIC DEPARTMENT
HALL
BREALEY LANGUAGES CENTRE
INNOVATION CENTRE
THE ARNOLD CENTRE
Car Park
ENGLISH FACULTY
WOODLAND CENTRE
WESTHOLME HOUSE
STABLE BLOCK
First Floor: 6th Form Resource Area
LINDUM HOUSE
Finance Department, Human Resources, Site Manager, Data Analysis
WESTGATE HOUSE
LEICESTER STREET
St. George’s Academy (Sleaford Site)
Westgate, Sleaford, Lincolnshire. NG34 7PS
Tel: 01529 302487 Fax: 01529 301175
E-mail: stga@st-georges-academy.org
Website: www.st-georges-academy.org
KEY:
P = Principal VP = Vice Principal PA = Principal's PA
R = Reception G = Reprographics COG = Chairman of Governors
KS3 = Year 7-9 Pastoral KS4 = Year 10-11 Pastoral KS5 = Year 12+ Pastoral

```python
def map_sga(self):
    if self.tab == "Overview":
        self.back = self.greenBack
    if self.tab == "Homework":
        self.back = self.blueBack
    if self.tab == "Coursework":
        self.back = self.yellowBack
    if self.tab == "Exam":
        self.back = self.redBack

    self.canvas.delete(ALL)
    button_back = Button(self, image=self.back, command=self.return_tasks, anchor = W)
    button_back.configure(border=0, relief = FLAT, highlightthickness=0)
    button_back_window = self.canvas.create_window(502, 315, anchor=NW, window=button_back)

    self.map = PhotoImage(file="map_sga.gif")
    self.canvas.create_image(54, 0, image = self.map, anchor = NW)
```

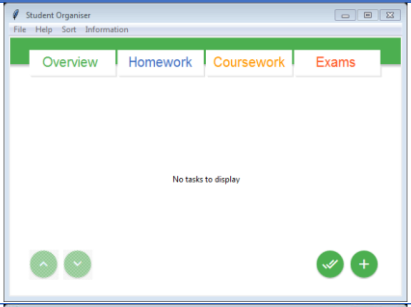
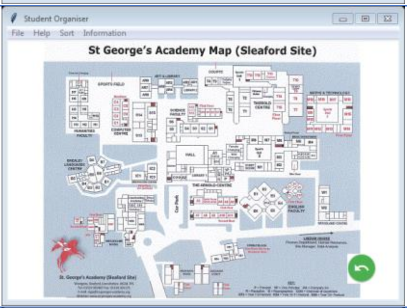
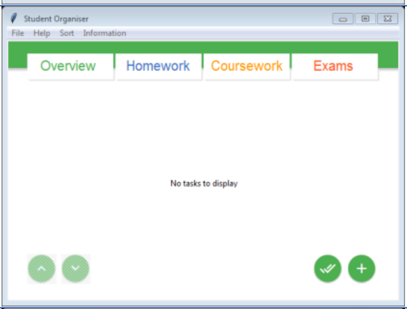
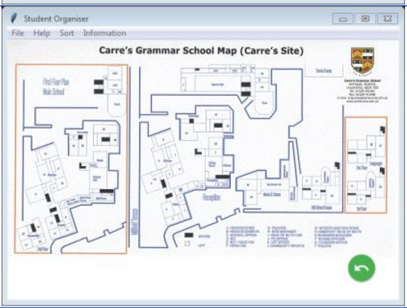
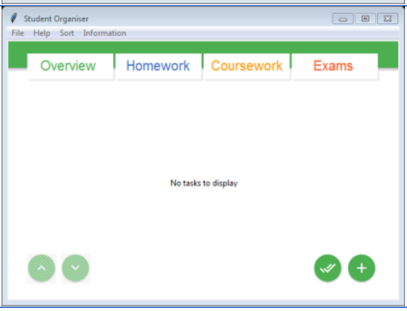
Carre’s Grammar School Map:

A plan of how I would like this map to appear in the program.



```python
def cgs_sga(self):
    if self.tab == "Overview":
        self.back = self.greenBack
    if self.tab == "Homework":
        self.back = self.blueBack
    if self.tab == "Coursework":
        self.back = self.yellowBack
    if self.tab == "Exam":
        self.back = self.redBack

    self.canvas.delete(ALL)
    button_back = Button(self, image=self.back, command=self.return_tasks, anchor = W)
    button_back.configure(border=0, relief = FLAT, highlightthickness=0)
    button_back_window = self.canvas.create_window(502, 315, anchor=NW, window=button_back)

    self.map = PhotoImage(file="map_sga.gif")
    self.canvas.create_image(0, 0, image = self.map, anchor = NW)
```

Each map had slightly different sizes; therefore, I needed to alter the coordinates so that they would both fit within the provided area.

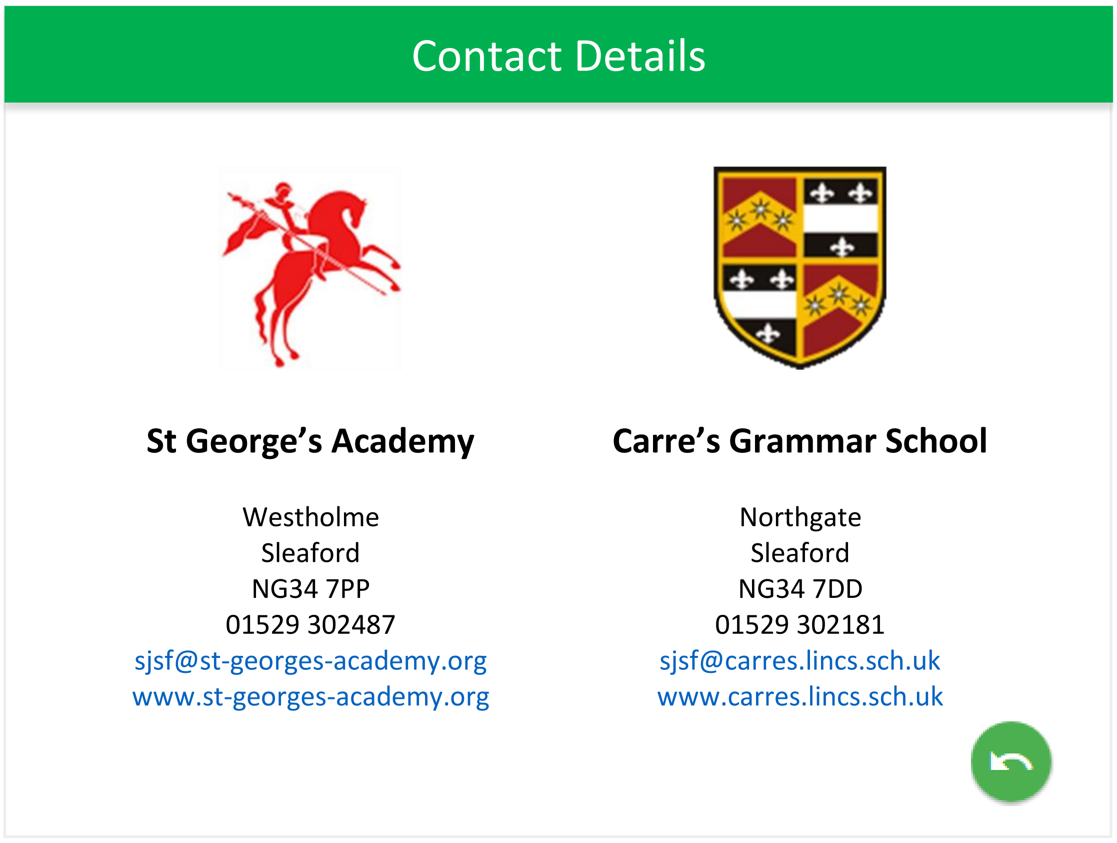
Test	Test Data	Expected Result	Actual Result / Evidence
Open the program	N/A	The program should show the overview tab any tasks written within the database.	 Student Organiser File Help Sort Information Overview Homework Coursework Exams No tasks to display
Use the menu bar to navigate to the St George's map	Information -> School Maps -> St George's Academy	The program should show a page without a title bar containing a large map of St George's Academy. There should also be a return button in the lower right hand corner.	 Student Organiser File Help Sort Information St George's Academy Map (Sleaford Site)
Click the return button	N/A	The program should show the overview tab any tasks written within the database.	 Student Organiser File Help Sort Information Overview Homework Coursework Exams No tasks to display
Use the menu bar to navigate to the Carre's Grammar map	Information -> School Maps -> Carre's Grammar School	The program should show a page without a title bar containing a large map of Carre's Grammar School. There should also be a return button in the lower right hand corner.	 Student Organiser File Help Sort Information Carre's Grammar School Map (Carre's Site)
Click the return button	N/A	The program should show the overview tab any tasks written within the database.	 Student Organiser File Help Sort Information Overview Homework Coursework Exams No tasks to display

The code created for the school maps page worked as expected so no further modifications are required.

Contact Details:

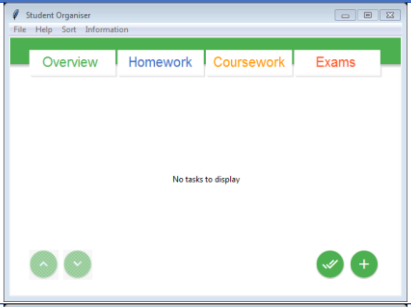
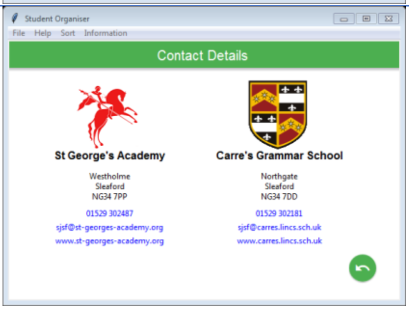
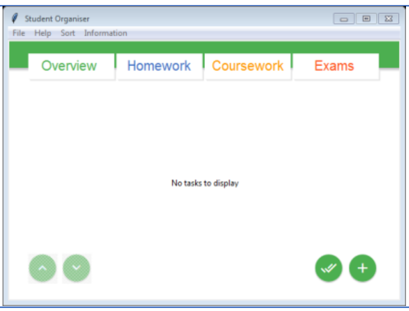
Here I will create a page that will show the school's contact details. I will use a title bar along the top that will be similar to that on the tasks page, with the exception that the tabs will not be drawn and instead, a white title will take their place. I will include the school's logo, name, address, phone number, email address and website. As this is a program, the phone number, email address and the website should be links. The phone number should open any external phone application installed on the computer with the correct number inserted, the email link should execute any email application installed with the 'to' address inserted correctly and finally, the website link should open the user's default browser and load the school's website.

A plan of how I would like this map to appear in the program.



Contact Details

St George's Academy	Carre's Grammar School
Westholme	Northgate
Sleaford	Sleaford
NG34 7PP	NG34 7DD
01529 302487	01529 302181
sjsf@st-georges-academy.org	sjsf@carres.lincs.sch.uk
www.st-georges-academy.org	www.carres.lincs.sch.uk

Test	Test Data	Expected Result	Actual Result / Evidence
Open the program	N/A	The program should show the overview tab any tasks written within the database.	 Student Organiser File Help Sort Information Overview Homework Coursework Exams No tasks to display
Use the menu bar to navigate to the contact details page	Information -> Contact Details	The program should show a page with a smaller title bar with the text, "Contact Details." The page should be split in half vertically containing the relevant contact details for SGA on the left and CGS on the right. There should also be a return button in the lower right hand corner.	 Student Organiser File Help Sort Information Contact Details St George's Academy Westholme Sleaford NG34 7PP 01529 302487 sjsf@st-georges-academy.org www.st-georges-academy.org Carre's Grammar School Northgate Sleaford NG34 7DD 01529 302181 sjsf@carres.lincs.sch.uk www.carres.lincs.sch.uk
Click the return button	N/A	The program should show the overview tab any tasks written within the database.	 Student Organiser File Help Sort Information Overview Homework Coursework Exams No tasks to display

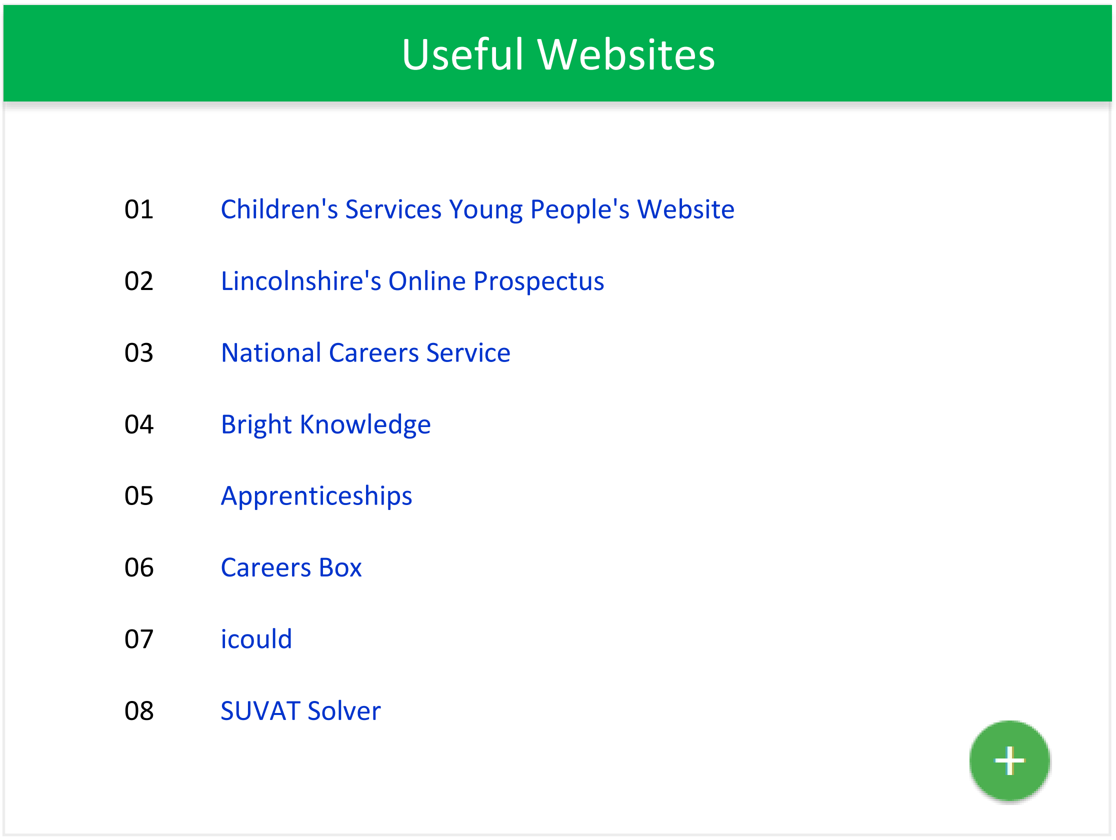
Useful Websites:

Using a similar theme to the contact details page, I will again clear the canvas, draw only the title bar and enter the title of, 'Useful Websites' for this page. I will create eight website links that the students for the selected year group should fund very useful. These will include; apprentices, carers advice, equation solving tools and the local prospectus. I will create a list of websites where they will be numbered from 1 to 8 and will be coloured in blue making it apparent that they are links that once clicked will open in their browser.

Each link should load the respective link in the user's default browser. This will work by using events. When the text is clicked, it will call a function with an event that will load the website. This function used the 'webbrowser.open_new' command that opens a given URL within the user's browser. To use this command, I must first import the module.

```python
import webbrowser
```

A plan of how I would like this map to appear in the program.



Useful Websites

01 Children's Services Young People's Website

02 Lincolnshire's Online Prospectus

03 National Careers Service

04 Bright Knowledge

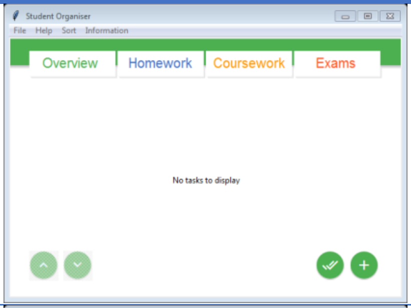
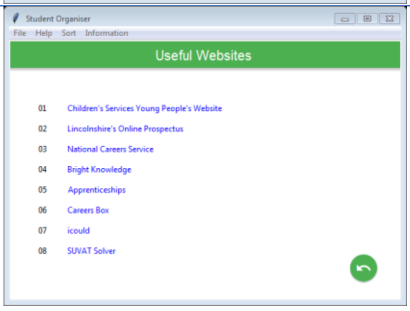
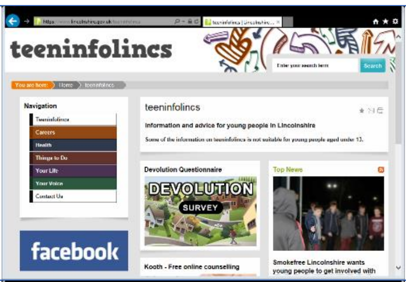
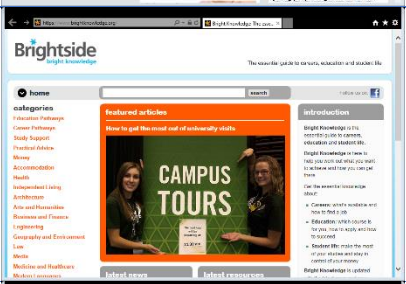
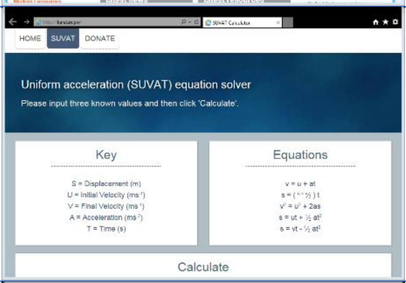
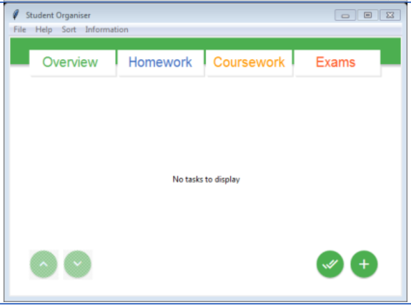
05 Apprenticeships

06 Careers Box

07 icould

08 SUVAT Solver

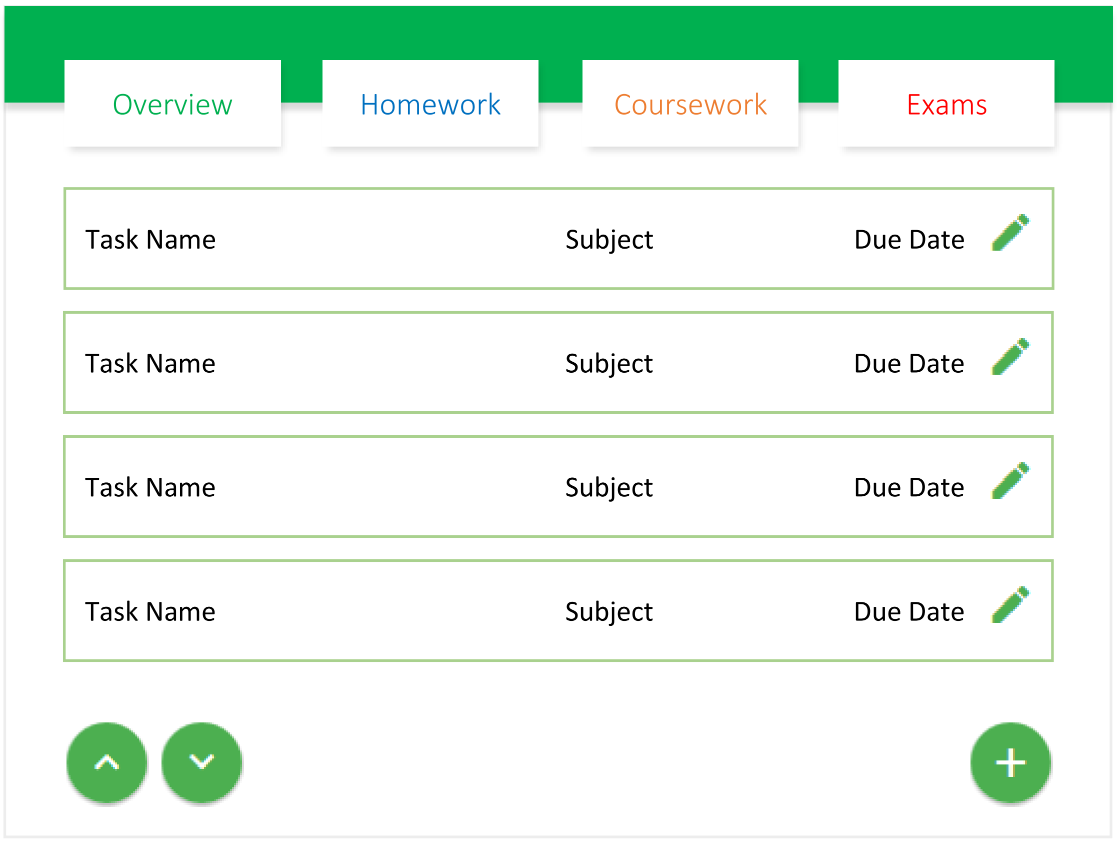
```python
def websites(self):
    if self.tab == "Overview":
        self.back = self.greenBack
    if self.tab == "Homework":
        self.back = self.blueBack
    if self.tab == "Coursework":
        self.back = self.yellowBack
    if self.tab == "Exam":
        self.back = self.redBack

    self.canvas.delete(ALL)
    self.draw_title_bar()
    self.canvas.create_rectangle(0, 40, 600, 0, fill=self.colour, outline=self.colour)
    self.canvas.create_text(287, 22, text="Useful Websites", font=self.tab_font, fill="white")

    button_back = Button(self, image=self.back, command=self.return_tasks, anchor = W)
    button_back.configure(border=0, relief = FLAT, highlightthickness=0)
    button_back_window = self.canvas.create_window(502, 315, anchor=NW, window=button_back)

    web1 = self.canvas.create_text(50, 100, text="01", fill="black")
    web1 = self.canvas.create_text(200, 100, text="Children's Services Young People's Website", fill="blue")
    self.canvas.tag_bind(web1, '<ButtonPress-1>', self.open_web1)

    web2 = self.canvas.create_text(50, 130, text="02", fill="black")
    web2 = self.canvas.create_text(171, 130, text="Lincolnshire's Online Prospectus", fill="blue")
    self.canvas.tag_bind(web2, '<ButtonPress-1>', self.open_web2)

    web3 = self.canvas.create_text(50, 160, text="03", fill="black")
    web3 = self.canvas.create_text(149, 160, text="National Careers Service", fill="blue")
    self.canvas.tag_bind(web3, '<ButtonPress-1>', self.open_web3)

    web4 = self.canvas.create_text(50, 190, text="04", fill="black")
    web4 = self.canvas.create_text(133, 190, text="Bright Knowledge", fill="blue")
    self.canvas.tag_bind(web4, '<ButtonPress-1>', self.open_web4)

    web5 = self.canvas.create_text(50, 220, text="05", fill="black")
    web5 = self.canvas.create_text(129, 220, text="Apprenticeships", fill="blue")
    self.canvas.tag_bind(web5, '<ButtonPress-1>', self.open_web5)

    web6 = self.canvas.create_text(50, 250, text="06", fill="black")
    web6 = self.canvas.create_text(116, 250, text="Careers Box", fill="blue")
    self.canvas.tag_bind(web6, '<ButtonPress-1>', self.open_web6)

    web7 = self.canvas.create_text(50, 280, text="07", fill="black")
    web7 = self.canvas.create_text(102, 280, text="icould", fill="blue")
    self.canvas.tag_bind(web7, '<ButtonPress-1>', self.open_web7)

    web8 = self.canvas.create_text(50, 310, text="08", fill="black")
    web8 = self.canvas.create_text(121, 310, text="SUVAT Solver", fill="blue")
    self.canvas.tag_bind(web8, '<ButtonPress-1>', self.open_web8)

def open_web1(self, event):
    webbrowser.open_new(r"http://www.teeninfolincs.co.uk")
def open_web2(self, event):
    webbrowser.open_new(r"http://www.14-19.info")
def open_web3(self, event):
    webbrowser.open_new(r"https://nationalcareersservice.direct.gov.uk")
def open_web4(self, event):
    webbrowser.open_new(r"http://www.brightknowledge.org")
def open_web5(self, event):
    webbrowser.open_new(r"http://www.apprenticeships.org.uk")
def open_web6(self, event):
    webbrowser.open_new(r"http://www.careersbox.co.uk")
def open_web7(self, event):
    webbrowser.open_new(r"http://www.icould.com")
def open_web8(self, event):
    webbrowser.open_new(r"http://www.karsten.pw")
```

Test	Test Data	Expected Result	Actual Result / Evidence
Open the program	N/A	The program should show the overview tab any tasks written within the database.	 Student Organiser File Help Sort Information Overview Homework Coursework Exams No tasks to display
Use the menu bar to navigate to the useful websites page	Information -> Useful Websites	The program should show a page with a smaller title bar with the text, "Useful Websites." The page should contain eight numbered websites in a list. Each website should open when clicked. There should also be a return button in the lower right hand corner.	 Student Organiser File Help Sort Information Useful Websites 01 Children's Services Young People's Website 02 Lincolnshire's Online Prospectus 03 National Careers Service 04 Bright Knowledge 05 Apprenticeships 06 Careers Box 07 icould 08 SUVAT Solver
Click the first website	N/A	The default browser should load with the Children's Services Young People's website.	 teeninfolincs Navigation teeninfolincs Information and advice for young people in Lincolnshire Devolution Questionnaire Top News facebook Kooth - Free online counselling Smokefree Lincolnshire wants young people to get involved with
Click the fourth website	N/A	The default browser should load with the Bright Knowledge website.	 Brightside bright knowledge home categories featured articles How to get the most out of university visits introduction
Click the last website	N/A	The default browser should load with the SUVAT equation solver website.	 HOME SUVAT DONATE Uniform acceleration (SUVAT) equation solver Please input three known values and then click 'Calculate'. Key S = Displacement (m) U = Initial Velocity (ms⁻¹) V = Final Velocity (ms⁻¹) A = Acceleration (ms⁻²) T = Time (s) Equations $v = u + at$ $s = (\frac{u+v}{2})t$ $v^2 = u^2 + 2as$ $s = ut + \frac{1}{2}at^2$ $s = vt - \frac{1}{2}at^2$ Calculate
Click the return button	N/A	The program should show the overview tab any tasks written within the database.	 Student Organiser File Help Sort Information Overview Homework Coursework Exams No tasks to display

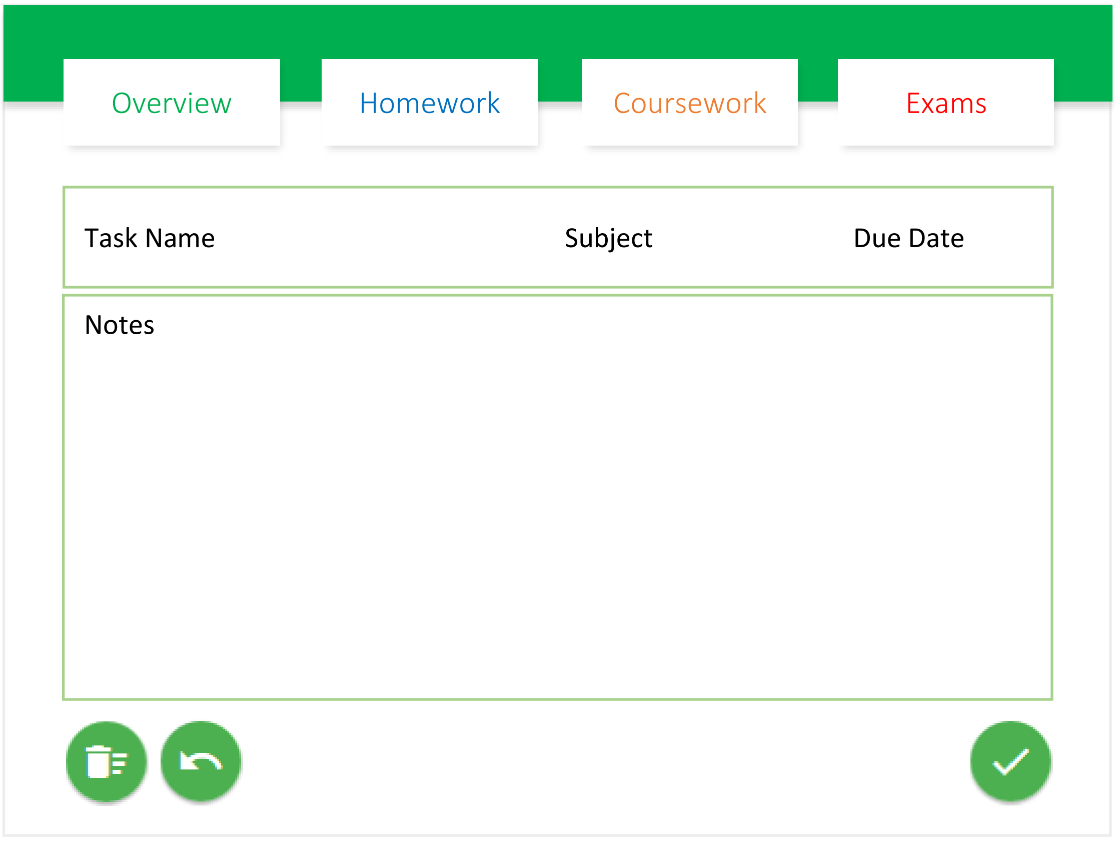
Client Meeting:

This is the fourth meeting with the client. My client has suggested that note should no longer be accessed through the menu bar. Instead, the user would prefer the option to add notes to their tasks. They would prefer a solution in which they can select any task on the tasks pages and a new page will appear with the task information and a large area to insert any of their notes. As well as this, the user had also requested that the tasks page should no longer be used to modify any of the tasks data and the delete button should be switched to a modify or view button which will load all the data regarding the relevant task.

Below is a plan of how the program should appear with these adjustments.



The above shows how the delete buttons have been replaced with icons representing modifications. Once clicked, the relevant task will be displayed with a large text area used for storing the user's notes. Below is a plan of how this page should appear once I create it.



The task is displayed in the format used in the tasks pages. Below this is a text box that the user can use to write in any note, it can be left empty if they do not need to use this. Three are three buttons that I have added to this page; the delete button will remove the task from the database and load the tasks page, the back button will return to the tasks page and any change made to the task will not be saved and the save button will save any information that the user has changed and save this to the database whilst loading the tasks page.

Adjustments:

As the tasks page will not be used to modify the tasks, I deleted all the code for the save_changes function as this is no longer required. I deleted the following code from the load_rows function as this is used to place the save button into the page. This button is not needed as I have previously deleted the function that it has been assigned.

```
button_save = Button(self, image=self.save, command=self.save_changes, anchor = W)
button_save.configure(border=0, relief = FLAT, highlightthickness=0)
button_save_window = self.canvas.create_window(452, 315, anchor=NW,
window=button_save)
```

I updated the ‘if statements’ at the beginning of the load_rows function so that the new buttons for modifying the tasks will be added in the correct colour and the redundant save button is removed.

```python
if self.tab == "Overview":
    self.page = self.ovpage
    self.up = self.greenUp
    self.down = self.greenDown
    self.back = self.greenBack
    self.edit = self.greenEdit
if self.tab == "Homework":
    self.page = self.hwpage
    self.up = self.blueUp
    self.down = self.blueDown
    self.back = self.blueBack
    self.edit = self.blueEdit
if self.tab == "Coursework":
    self.page = self.cwpage
    self.up = self.yellowUp
    self.down = self.yellowDown
    self.back = self.yellowBack
    self.edit = self.yellowEdit
if self.tab == "Exam":
    self.page = self.expage
    self.up = self.redUp
    self.down = self.redDown
    self.back = self.redBack
    self.edit = self.redEdit
```

These variables can now be used to use the correct button depending on the tab. I then Imported these images for the modify buttons within the init function.

```python
self.greenEdit = PhotoImage(file="buttons/green_edit.gif")
self.blueEdit = PhotoImage(file="buttons/blue_edit.gif")
self.yellowEdit = PhotoImage(file="buttons/yellow_edit.gif")
self.redEdit = PhotoImage(file="buttons/red_edit.gif")
```

The final stage of creating this button is to insert it into the correct position on the page. As they will be replacing the delete buttons, I can use the existing code and just change the image and command. Before, the delete buttons were inserted into the page with this code:

```python
button_remove_1 = Button(self, image=self.remove, command=lambda:
self.delete_row(0), anchor = W)
button_remove_1.configure(border=0, relief = FLAT, highlightthickness=0)
button_remove_1_window = self.canvas.create_window(520, 94, anchor=NW,
window=button_remove_1, tags="rowBox")
```

I have modified these to use the modify icon, a crayon, and to call a load_notes function as opposed to the delete_row function.

```python
button_modify_1 = Button(self, image=self.edit, command=lambda: self.load_notes(0),
anchor = W)
button_modify_1.configure(border=0, relief = FLAT, highlightthickness=0)
button_modify_1_window = self.canvas.create_window(520, 94, anchor=NW,
window=button_modify_1, tags="rowBox")
```

Now every button used to load the modify task page has been created and all redundant code from previously used buttons has been removed and updated. The next part of this task is to create the load_notes function for when the user clicks this button.

As I know that I will be using three buttons, delete, return and save, I can assign variables to these so that the correct colour button will load depending on the current tab that the user had selected.

```
def load_notes(self, task_id):
    if self.tab == "Overview":
        self.done = self.greenDone
        self.delete = self.greenDelete
        self.back = self.greenBack
    if self.tab == "Homework":
        self.done = self.blueDone
        self.delete = self.blueDelete
        self.back = self.blueBack
    if self.tab == "Coursework":
        self.done = self.yellowDone
        self.delete = self.yellowDelete
        self.back = self.yellowBack
    if self.tab == "Exam":
        self.done = self.redDone
        self.delete = self.redDelete
        self.back = self.redBack
```

The function will take the variable, task_id, as this will be passed from the previous page. This value will contain the ID of the selected task within the database so that I can use this to collect all the relevant information from the database in this page. It will also allow me to delete or modify the task too. As this is loading a new page, I need to clear the canvas and add in the tabs as usual.

```
    self.canvas.delete(ALL)
    self.draw_title_bar()
    self.canvas.create_rectangle(0, 40, 600, 0, fill=self.colour, outline=self.colour)
    self.draw_tabs()
```

I then added the row in which the task will be displayed. This will be the same as the tasks page with the coloured box containing the task name, subject and the due date. I copied the code from the load_rows function to create the container for the selected task.

```
    self.canvas.create_rectangle(30, 132, 548, 81, fill="#f9f9f9", outline="#f9f9f9", tags="rowBox")
    self.canvas.create_rectangle(30, 130, 546, 80, fill="white", outline=self.rowColour, width=2, tags="rowBox")

    self.task_name = Entry(self,width=45)
    self.task_name.configure(border=0, relief=FLAT, bg="white")
    task_name_window = self.canvas.create_window(40, 97, anchor=NW, window=self.task_name)
    self.task_subject = Entry(self,width=20)
    self.task_subject.configure(border=0, relief=FLAT, bg="white")
    task_subject_window = self.canvas.create_window(320, 97, anchor=NW, window=self.task_subject)
    self.task_date = Entry(self,width=15)
    self.task_date.configure(border=0, relief=FLAT, bg="white")
    task_date_window = self.canvas.create_window(450, 97, anchor=NW, window=self.task_date)
```

The positions of these entry boxes and the outer rectangle is identical to that of the first row in the page of tasks. Therefore, I did not need to adjust the coordinates of these widgets. To add the notes textbox, I needed to research how to use a scrolling text field in Python. I found the following which implements a scrollable text field into a frame in Python2.7.

This is the code that I had found on the website:

```python
from Tkinter import *
class scrollTxtArea:
    def __init__(self,root):
        frame=Frame(root)
        frame.pack()
        self.textPad(frame)
        return

    def textPad(self,frame):
        #add a frame and put a text area into it
        textPad=Frame(frame)
        self.text=Text(textPad,height=50,width=90)

        # add a vertical scroll bar to the text area
        scroll=Scrollbar(textPad)
        self.text.configure(yscrollcommand=scroll.set)

        #pack everything
        self.text.pack(side=LEFT)
        scroll.pack(side=RIGHT,fill=Y)
        textPad.pack(side=TOP)
        return
def main():
    root = Tk()
    foo=scrollTxtArea(root)
    root.title('TextPad With a Vertical Scroll Bar')
    root.mainloop()
main()
```

I adjusted this example to work within my code as I am using a canvas to place the widgets on.

```python
    self.tb = Text(self, width=64, height=11)
    self.tb.configure(border=1, bg="white")
    self.tb_window = self.canvas.create_window(30, 132, anchor=NW, window=self.tb)
```

I have manually altered the coordinates so that this text box will be place a few pixels below the task details and is also aligned to the left margin. To align the right side of this text box with the task details above, I set the width to 11.

I then inserted the code to place the three buttons on the page. I am using the same coordinates as the tasks pages so that there is a common theme and therefore I will not need to alter the coordinates for this page. I also decided upon the order so that the dele button would be positioned in the lower right and the return button should be placed to the right of this. This matches the two buttons to the left and the remaining save button the right.

```
    button_delete = Button(self, image=self.delete, command=lambda: self.delete_notes_task(task_id), anchor = W)
    button_delete.configure(border=0, relief = FLAT, highlightthickness=0)
    button_delete_window = self.canvas.create_window(30, 315, anchor=NW, window=button_delete)

    button_back = Button(self, image=self.back, command=self.return_tasks, anchor = W)
    button_back.configure(border=0, relief = FLAT, highlightthickness=0)
    button_back_window = self.canvas.create_window(80, 315, anchor=NW, window=button_back)

    button_done = Button(self, image=self.done, command=self.check_add_name, anchor = W)
    button_done.configure(border=0, relief = FLAT, highlightthickness=0)
    button_done_window = self.canvas.create_window(502, 315, anchor=NW, window=button_done)
```

The delete button will call the delete_notes_task function, the back button will call the return_tasks function and the save button will call the check_add_name function.

Then, I placed the data from the selected task into the page. I achieved this by using the list of data for the task and appending each piece of data within that list to a variable. Once completed, I could insert this data into the respective entry box. However, if the due date is earlier than the current date, the text colour of the task should be red.

```
    self.task_id = self.set[0][4]
    self.task_type = self.set[0][3]
    self.task_name.insert(INSERT, self.set[task_id][0])
    self.task_subject.insert(INSERT, self.set[task_id][1])
    self.task_date.insert(INSERT, self.set[task_id][2])
    self.tb.insert(INSERT, self.set[task_id][5])

    todayDate = (time.strftime("%d/%m/%Y"))
    if todayDate > self.task_date.get():
        self.task_name.configure(fg="#F44336")
        self.task_subject.configure(fg="#F44336")
        self.task_date.configure(fg="#F44336")
```

To allow me to reuse previous functions, I have added in a variable called editPage. This will be used when validating the data before the task is written to the database.

```
    self.editPage = True
```

Deleting a task:

I have changed the function and method for deleting the tasks from the database. The user will now no longer be able to remove data from the tasks pages, instead, they must select a task to modify and then use the on screen delete button. I have written a new function that will delete this task. I used a lambda function, as this will allow me to pass a variable to the function. The variable that I chose to pass was the task ID that is very useful as I can easily delete that task with the ID given, there will be no need to have multiple functions.

```
def save_notes_task(self):
    t1d=self.task_date.get()
    date1=t1d.split('/')
    t1d=date1[2]+"/"+date1[1]+"/"+date1[0]

    self.conn = sqlite3.connect('tasks.db')
    self.conn.execute("DELETE from TASKS where ID='"+ str(self.task_id)+"';")
    lastRow = self.conn.execute("SELECT MAX(id) FROM TASKS")
    for row in lastRow:
        self.id = row[0]
    if self.id == None:
        self.id = 0
    self.conn.execute("INSERT INTO TASKS(ID, TASK, SUBJECT, DATE, TAB, NOTES) values (?, ?, ?, ?, ?, ?)", (self.id+1, self.task_name.get(), self.task_subject.get(), t1d, self.task_type, str(self.tb.get("1.0", END))))
    self.conn.commit()
    self.refresh_task_list_remove()
```

The data required for me to write a task to the database includes; the task ID, the task name, subject, due date, the tab and the notes (if any.) First, I collected the date from the entry box designated to the date input. I then split this up at every '/' and constructed a string with this data to form a date in the order YYYY/MM/DD as this is the format that is required by SQLite. Then I connected to the database and executed a command to find the greatest ID of any task within the database. IF there are no tasks, it should be assumed that the ID to use is zero. I then created an SQL statement that will add a record into the database with all the information being taken from the previous screen, the modify page.

Once the task has been written, I call the refresh_task_list function that I have previously written for efficiency. This commits the changes, closes the connection to the database and final reloads the tasks page so that the user can see the changes in place with the correct data loaded in. Below is a copy of the function that I used.

```
def refresh_task_list_remove(self):
    self.conn.commit()
    self.conn.close()
    self.setup_data()
```

Returning to the Tasks Page:

The return button should return the user to the previous page of tasks without any changes being submitted to the database.

Saving a Task:

Saving a task will update the task in the database with the details entered in this page. To save a task, it must first be validated. This will ensure that the program will not crash with invalid inputs by the user. I will be using the validation functions that I have previously created for efficiency. Therefore, I will change these functions so that they will still work for validating when adding a task and when modifying and applying the changes for a task.

Validating Task Name:

The first piece of data to be validated is the task name. As the name assigned for the entry box of the task name is the same in the modify and add page, there was no need to modify this function.

```
def check_add_name(self):
    if self.task_name.get() == "":
        messagebox.showinfo("Invalid Entry", "Task name cannot be left blank.")
    else:
        self.check_add_subject()
```

Validating Task Subject:

If the task name check has passed, then the subject field must be checked. As the subject entry box uses the same name for the add task page and modify task page, this function did not need changing.

```
def check_add_subject(self):
    if self.task_subject.get() == "":
        messagebox.showinfo("Invalid Entry", "Subject cannot be left blank.")
    else:
        self.check_add_date()
```

Validating Task Date:

If the subject check has passed, then the date should be checked. The date is entered differently for the add task page and the modify task page. For the add task page, there are three drop down menus, the first containing integers from 1 to 31, the second containing a string for each month and the last contains 5 integers for each year from the current. However, in the modify task page, the date is inserted in a single entry box with the format of DD/MM/YYYY. Therefore, I will split the next function into two parts, the first will run if the page validated is the modify tasks page, the second will run if the add task page is being validated.

If self.editPage is true, then the string from the date entry box is collected in the form of DD/MM/YYYY. This is split at every '/' which will create a list of three items; the day, the month and the year. If the length of this list is less than 3, then the message, "Incorrect date format, please use DD/MM/YYYY" will be presented to the user in the form of a pop up box. If the entry box is left empty, then the user will receive the following message, "Date cannot be left blank." If the length of the data entered in this box does not equal 10, then there are too many or too few characters and the message, "Incorrect date format, please use DD/MM/YYYY," is shown. To check that the date entered is in the form of INTEGER / INTEGER / INTEGER, I then write a test to check if these values are digits. If not, then there is an error in the date entered and the user will receive a pop up with the details. Finally, if the second and fifth character in the whole entry box is not a '/' then it does not follow the correct date format required so the verification will fail. This is the code for this function that affects the modify page:

```python
def check_add_date(self):
    if self.editPage:
        dateData = (self.task_date.get().strip().split('/'))
        if len(dateData) < 3:
            messagebox.showinfo("Invalid Entry", "Date cannot be left blank.")
        self.taskDay = dateData[0]
        self.monthNumber = dateData[1]
        self.taskYear = dateData[2]

        if self.task_date.get() == "":
            messagebox.showinfo("Invalid Entry", "Date cannot be left blank.")
        elif len(self.task_date.get()) != 10:
            messagebox.showinfo("Invalid Entry", "Incorrect date format, please use DD/MM/YYYY.")
        elif self.taskDay.isdigit() == False or self.monthNumber.isdigit() == False or self.taskYear.isdigit() == False:
            messagebox.showinfo("Invalid Entry", "Incorrect date format, please use DD/MM/YYYY.")
        elif (self.task_date.get()[2]) != "/" or (self.task_date.get()[5]) != "/":
            messagebox.showinfo("Invalid Entry", "Incorrect date format, please use DD/MM/YYYY.")
        else:
            self.check_add_valid_date()
```

If the page being validated is the add task page, then a different set of checks are performed. As the user is restricted in terms of entering the date, there will be less verification methods to perform. The only checks that should be used in this function is to validate that the user has selected an option for the three entry boxes. This is the same as the old function before adding the above code, just with the additional 'else' statement.

```python
    else:
        if self.task_day.get() == "":
            messagebox.showinfo("Invalid Entry", "Day cannot be left blank.")
        else:
            if self.task_month.get() == "":
                messagebox.showinfo("Invalid Entry", "Month cannot be left blank.")
            else:
                if self.task_year.get() == "":
                    messagebox.showinfo("Invalid Entry", "Year cannot be left blank.")
                else:
                    self.check_add_valid_date()
```

Once the user inputted date has been verified in both cases, I can use these inputs to check if the date is an actual date. This function will involve converting months given as integers to their name and vice versa. To do this, I created a dictionary as this can be used to find a value by the ID or the ID by the value, making it a very useful tool in this process.

```python
def check_add_valid_date(self):
    months = {"January": 1,
      "February": 2,
      "March": 3,
      "April": 4,
      "May": 5,
      "June": 6,
      "July": 7,
      "August": 8,
      "September": 9,
      "October": 10,
      "November": 11,
      "December": 12}
```

To reduce the amount of code required, I decided to use four variables that will be created in the modify page or add new page part of this function. These will be

Variable	Data Type	Example
$self.taskDay$	Integer	29
$self.monthNumber$	Integer	12
$self.monthName$	String	December
$self.taskYear$	Integer	2016

If this function is run from the add task page, then the task day, month name and year are all assigned to the variables self.taskDay, self.monthName and self.taskYear. As the month name has been given as an input, the number should be found in order to validate the date. I look up every value in the dictionary until the month name selected matches that in the dictionary. Once complete, the ID of this item is assigned to the self.monthNumber variable. At this point, all four of these variables have been assigned, ready for the next stage.

```python
    if self.editPage == False:
        self.taskDay = self.task_day.get()
        self.monthName = self.task_month.get()
        self.taskYear = self.task_year.get()

        for name, number in months.items():
            if self.monthName == str(name):
                self.monthNumber = number
```

If the function is called through the modify task page, then the date is given through a single entry box in the format of DD/MM/YYYY. This is again, split up at every '/' to form a list of three items for each part of the date. These three values make up the variables, self.taskDay, self.monthNumber and self.taskYear. To find the name of the month, I first removed the proceeding 0 from the month number if present. This gave a single or double digit integer that will now correspond to the ID's of the values in the dictionary. Using this new value, I searched through the dictionary, comparing this value with every ID present until they match. This match will correlate the month ID with the month's name. This name is required for use in the error message.

```
    else:
        dateData = (self.task_date.get().strip().split('/'))
        self.taskDay = dateData[0]
        self.monthNumber=dateData[1]
        self.task_year = dateData[2]

        monthIntegers=[]
        for n in self.monthNumber:
            monthIntegers.append(n)
        if  monthIntegers[0]=="0":
            self.monthNumber=monthIntegers[1]

        for name, number in months.items():
            if self.monthNumber == str(number):
                self.monthName = name
```

All four variables have been assigned and therefore, I can validate the date for both types of input pages together. I used the arrays that I created in this function earlier that store the months containing 30 and 31 days. I continued to reuse the previous code that will show an error message if the day selected is greater than the maximum allowed for the selected month.

```
    # The maximum number of days for each month is determined
    # 2 arrays are created
    # The first contains every month that can have up to 30 days
    # The second contains every month that can have up to 31 days
    days_30 = (4,6,9,11)
    days_31 = (1,3,5,7,8,10,12)

    # If the user enters a day value greater than 30 for a 30 day month, a message will be shown
    # If the user enters a day value greater than 31 for a 31 day month, a message will be shown
    if int(self.monthNumber) in days_30 and int(self.taskDay) > 30:
        messagebox.showinfo("Invalid Entry", str(self.monthName)+" can only have up to 30 days.")

    elif int(self.monthNumber) in days_31 and int(self.taskDay) > 31:
        messagebox.showinfo("Invalid Entry", str(self.monthName)+" can only have up to 31 days.")
    else:
        self.check_leap_year()
```

The final check for the date is to verify that the date is valid for leap years. This will only affect dates where the user has selected February as the due date. I reused this function from when I created this earlier. I had only altered the variable names so that I can use the variables created in the previous function so that it will apply for both types of entry page.

```python
def check_leap_year(self):
    # Check if the year selected is a leap year
    # A variable 'leap_year' is created an is assigned as false
    # If the selected year can be divided by 4, it is considered a leap year
    # But, if the selected year can be divided by 100, it is no longer a leap year
    # If the selected year can be divided by 400, it is a leap year
    leap_year=False
    if (int(self.taskYear)%4) == 0:
        leap_year=True
        if (int(self.taskYear)%100) == 0:
            leap_year=False
        elif (int(self.taskYear)%400) == 0:
            leap_year=True

    # If the user has selected February and a leap_year is true, the day cannot be greater than 29
    # If the user has selected February and a leap_year is false, the day cannot be greater than 28
    # If the day is valid, the write_task function is called


    if int(self.monthNumber) == 2 and leap_year and int(self.taskDay) > 29:
        messagebox.showinfo("Invalid Entry", "On a leap year, "+str(self.monthName)+" can only have up to 29 days.")
    elif int(self.monthNumber) == 2 and not leap_year and int(self.taskDay) > 28:
        messagebox.showinfo("Invalid Entry", "Not on a leap year, "+str(self.monthName)+" can only have up to 28 days.")
    else:
        if self.editPage == False:
            self.write_task()
        else:
            self.editPage=False
            self.save_notes_task()
```

Write Task:

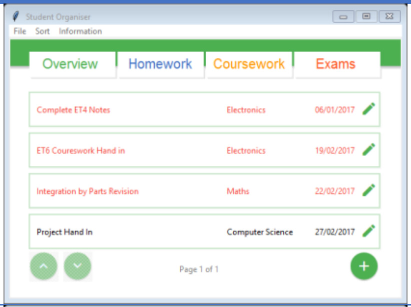
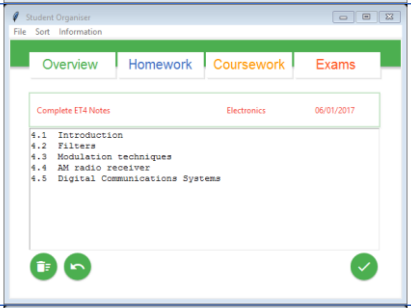
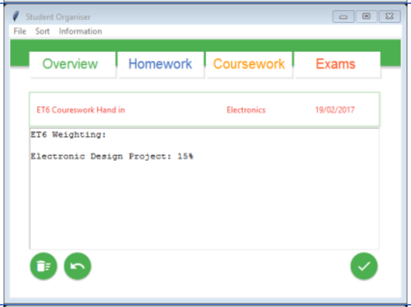
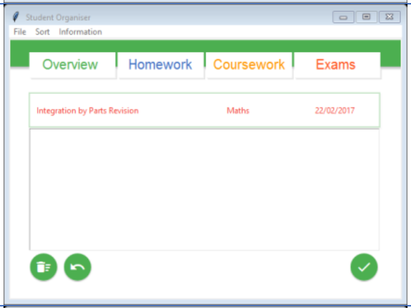
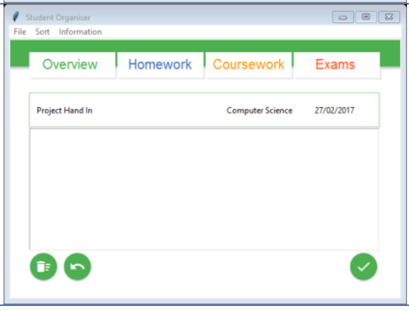
If the user adds a task through the add task page and all data entered passes the validation, then the write_task function loads. This is the same function as before.

If the user modifies the task, the save_notes_task function loads. This will update the record in the database with the new data entered in this page. This works by deleting the selected task from the database. Once removed, it will create a new record with data that is present in the modify tasks page as this is the new data that the user has submitted.

```python
def save_notes_task(self):
    t1d=self.task_date.get()
    date1=t1d.split('/')
    t1d=date1[2]+"/"+date1[1]+"/"+date1[0]

    self.conn = sqlite3.connect('tasks.db')
    self.conn.execute("DELETE from TASKS where ID='"+ str(self.task_id)+"';")
    lastRow = self.conn.execute("SELECT MAX(id) FROM TASKS")
    for row in lastRow:
        self.id = row[0]
    if self.id == None:
        self.id = 0
    self.conn.execute("INSERT INTO TASKS(ID, TASK, SUBJECT, DATE, TAB, NOTES) values (?, ?, ?, ?, ?, ?)", (self.id+1, self.task_name.get(), self.task_subject.get(), t1d, self.task_type, str(self.tb.get("1.0", END))
))
    self.conn.commit()
    self.refresh_task_list_remove()
```

Testing of the Amendments:

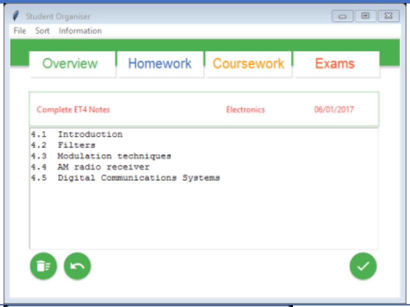
Testing the Edit Page:

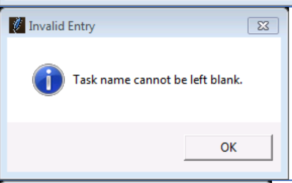
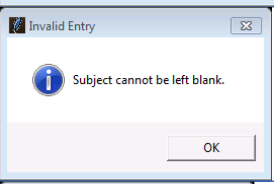
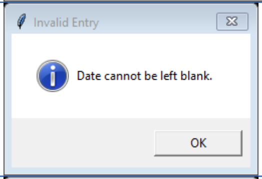
Test	Test Data	Expected Result	Actual Result / Evidence
Load the program	N/A	The program should show every task on the overview page.	 Student Organiser File Sort Information Overview Homework Coursework Exams Complete ET4 Notes Electronics 06/01/2017 ET6 Coureswork Hand in Electronics 19/02/2017 Integration by Parts Revision Maths 22/02/2017 Project Hand In Computer Science 27/02/2017 Page 1 of 1
Edit task 1 (click the crayon button beside the task)	N/A	The program should show the first task's data with a large text area for notes.	 Student Organiser File Sort Information Overview Homework Coursework Exams Complete ET4 Notes Electronics 06/01/2017 4.1 Introduction 4.2 Filters 4.3 Modulation techniques 4.4 AM radio receiver 4.5 Digital Communications Systems
Edit task 2 (click the crayon button beside the task)	N/A	The program should show the second task's data with a large text area for notes.	 Student Organiser File Sort Information Overview Homework Coursework Exams ET6 Coureswork Hand in Electronics 19/02/2017 ET6 Weighting: Electronic Design Project: 15%
Edit task 3 (click the crayon button beside the task)	N/A	The program should show the third task's data with a large text area for notes.	 Student Organiser File Sort Information Overview Homework Coursework Exams Integration by Parts Revision Maths 22/02/2017
Edit task 4 (click the crayon button beside the task)	N/A	The program should show the fourth task's data with a large text area for notes.	 Student Organiser File Sort Information Overview Homework Coursework Exams Project Hand In Computer Science 27/02/2017

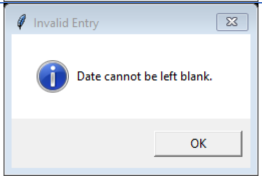
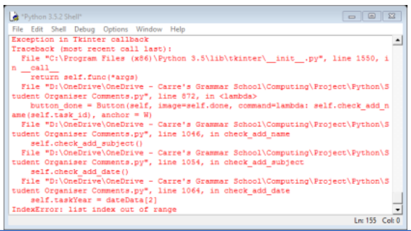
The edit page worked as expected and loaded the correct task's data depending on the button that was used. The notes box contained any notes that had been previously entered too.

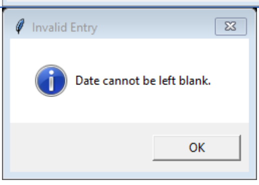
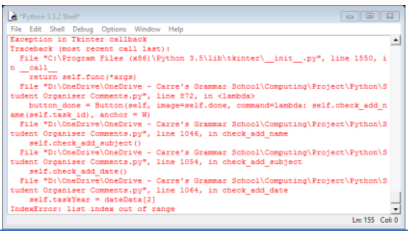
Testing the Input Validation:

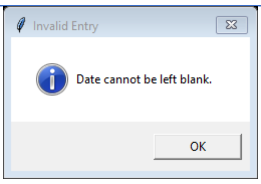
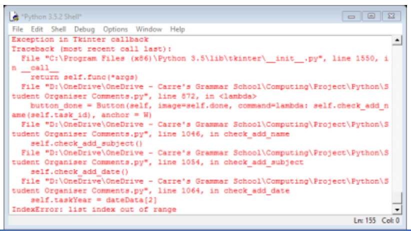
Test	Test Data	Expected Result	Actual Result / Evidence

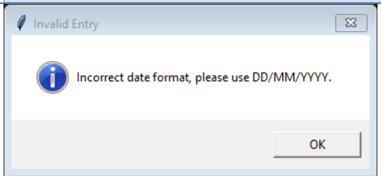
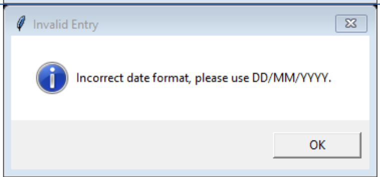
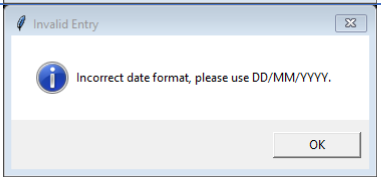
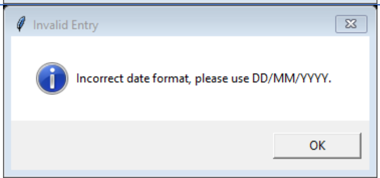
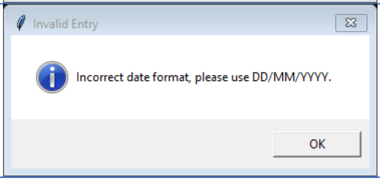
Edit task 1 (click the crayon button beside the task)	N/A	The program should show the first task's data with a large text area for notes.	 Student Organiser
			File Sort Information
			Overview Homework Coursework Exams
			Complete ET4 Notes Electronics 06/01/2017
			`4.1 Introduction`
			`4.2 Filters`
			`4.3 Modulation techniques`
			`4.4 AM radio receiver`
			`4.5 Digital Communications Systems`

Clear the task name and save	NAME: ‘’	An error message appears with the text, ‘Task name cannot be left blank.’	 Invalid Entry Task name cannot be left blank. OK
Clear the subject and save	SUBJECT: ‘’	An error message appears with the text, ‘Subject cannot be left blank.’	 Invalid Entry Subject cannot be left blank. OK
Clear the date and save	DATE: ‘’	An error message appears with the text, ‘Date cannot be left blank.’	 Invalid Entry Date cannot be left blank. OK

Change the third character from a ‘/’	DATE: ’06c01/2017’	An error message appears with the text, ‘Incorrect date format, please use DD/MM/YYYY.’	 Invalid Entry
			Date cannot be left blank.
			OK
			 *Python 3.5.2 Shell*
			File Edit Shell Debug Options Window Help
			`Exception in Tkinter callback`
			`Traceback (most recent call last):`
			`  File "C:\Program Files (x86)\Python 3.5\lib\tkinter\__init__.py", line 1550, in __call__`
			`    return self.func(*args)`
			`  File "D:\OneDrive\OneDrive - Carre's Grammar School\Computing\Project\Python\Student Organiser Comments.py", line 872, in <lambda>`
			`    button_done = Button(self, image=self.done, command=lambda: self.check_add_name(self.task_id), anchor = W)`
			`  File "D:\OneDrive\OneDrive - Carre's Grammar School\Computing\Project\Python\Student Organiser Comments.py", line 1046, in check_add_name`
			`    self.check_add_subject()`
			`  File "D:\OneDrive\OneDrive - Carre's Grammar School\Computing\Project\Python\Student Organiser Comments.py", line 1054, in check_add_subject`
			`    self.check_add_date()`
			`  File "D:\OneDrive\OneDrive - Carre's Grammar School\Computing\Project\Python\Student Organiser Comments.py", line 1064, in check_add_date`
			`    self.taskYear = dateData[2]`
			`IndexError: list index out of range`
			Ln: 155 Col: 0

Change the sixth character from a ‘/’	DATE: ’06/01-2017’	An error message appears with the text, ‘Incorrect date format, please use DD/MM/YYYY.’	 Invalid Entry
			Date cannot be left blank.
			OK
			 *Python 3.5.2 Shell*
			File Edit Shell Debug Options Window Help
			`Exception in Tkinter callback`
			`Traceback (most recent call last):`
			`  File "C:\Program Files (x86)\Python 3.5\lib\tkinter\__init__.py", line 1550, in __call__`
			`    return self.func(*args)`
			`  File "D:\OneDrive\OneDrive - Carre's Grammar School\Computing\Project\Python\Student Organiser Comments.py", line 872, in <lambda>`
			`    button_done = Button(self, image=self.done, command=lambda: self.check_add_name(self.task_id), anchor = W)`
			`  File "D:\OneDrive\OneDrive - Carre's Grammar School\Computing\Project\Python\Student Organiser Comments.py", line 1046, in check_add_name`
			`    self.check_add_subject()`
			`  File "D:\OneDrive\OneDrive - Carre's Grammar School\Computing\Project\Python\Student Organiser Comments.py", line 1054, in check_add_subject`
			`    self.check_add_date()`
			`  File "D:\OneDrive\OneDrive - Carre's Grammar School\Computing\Project\Python\Student Organiser Comments.py", line 1064, in check_add_date`
			`    self.taskYear = dateData[2]`
			`IndexError: list index out of range`
			Ln: 155 Col: 0

Change the third & sixth characters from a '/'	DATE: '06201,2017'	An error message appears with the text, 'Incorrect date format, please use DD/MM/YYYY.'	 Invalid Entry
			Date cannot be left blank.
			OK
			 `Exception in Tkinter callback`
			`Traceback (most recent call last):`
			`  File "C:\Program Files (x86)\Python 3.5\lib\tkinter\__init__.py", line 1550, in __call__`
			`    return self.func(*args)`
			`  File "D:\OneDrive\OneDrive - Carre's Grammar School\Computing\Project\Python\Student Organiser Comments.py", line 872, in <lambda>`
			`    button_done = Button(self, image=self.done, command=lambda: self.check_add_name(self.task_id), anchor = W)`
			`  File "D:\OneDrive\OneDrive - Carre's Grammar School\Computing\Project\Python\Student Organiser Comments.py", line 1046, in check_add_name`
			`    self.check_add_subject()`
			`  File "D:\OneDrive\OneDrive - Carre's Grammar School\Computing\Project\Python\Student Organiser Comments.py", line 1054, in check_add_subject`
			`    self.check_add_date()`
			`  File "D:\OneDrive\OneDrive - Carre's Grammar School\Computing\Project\Python\Student Organiser Comments.py", line 1064, in check_add_date`
			`    self.taskYear = dateData[2]`
			`IndexError: list index out of range`

Add more than 10 characters	DATE: '06/01/20117'	An error message appears with the text, 'Incorrect date format, please use DD/MM/YYYY.'	 Invalid Entry Incorrect date format, please use DD/MM/YYYY. OK
Add less than 10 characters	DATE: '06/01/207'	An error message appears with the text, 'Incorrect date format, please use DD/MM/YYYY.'	 Invalid Entry Incorrect date format, please use DD/MM/YYYY. OK
Add non numeric characters to the day	DATE: 'cl/01/2017'	An error message appears with the text, 'Incorrect date format, please use DD/MM/YYYY.'	 Invalid Entry Incorrect date format, please use DD/MM/YYYY. OK
Add non numeric characters to the month	DATE: '06/@1/2017'	An error message appears with the text, 'Incorrect date format, please use DD/MM/YYYY.'	 Invalid Entry Incorrect date format, please use DD/MM/YYYY. OK
Add non numeric characters to the year	DATE: '06/01/#¬p7'	An error message appears with the text, 'Incorrect date format, please use DD/MM/YYYY.'	 Invalid Entry Incorrect date format, please use DD/MM/YYYY. OK

Most of the input validation worked as expected. However, there was an issue when changing the third and sixth character in the entry box. The expected outcome for these of these events was that an error message appears with the text, 'Incorrect date format, please use DD/MM/YYYY.' The function would then pass, allowing the user to alter their inputs and the program should not crash as it did during the test.

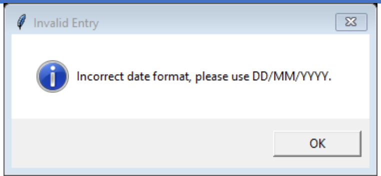
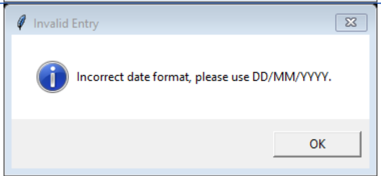
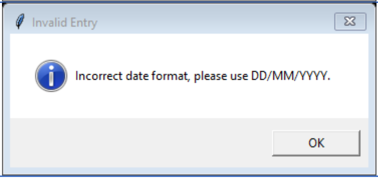
The problem was the order in which I validated the date. Currently, I split the date into the three main parts, the day, month and year. If the day was less than 3 in length, an error message would be shown to the user. Then I check if it is blank, followed by checking the third and sixth character. The issue with this is that the date is split at every '/' and as there would be a single '/' in the input, this fails the first check. There are now only two pieces of the date found, not 3, which is less than the expected 3.

The code that failed this test:

```python
def check_add_date(self):
    if self.editPage:
        dateData = (self.task_date.get().strip().split('/'))
        if len(dateData) < 3:
            messagebox.showinfo("Invalid Entry", "Date cannot be left blank.")
        self.taskDay = dateData[0]
        self.monthNumber = dateData[1]
        self.taskYear = dateData[2]

        if self.task_date.get() == "":
            messagebox.showinfo("Invalid Entry", "Date cannot be left blank.")
        elif len(self.task_date.get()) != 10:
            messagebox.showinfo("Invalid Entry", "Incorrect date format, please use DD/MM/YYYY.")
        elif self.taskDay.isdigit() == False or self.monthNumber.isdigit() == False or self.taskYear.isdigit() == False:
            messagebox.showinfo("Invalid Entry", "Incorrect date format, please use DD/MM/YYYY.")
        elif (self.task_date.get()[2]) != "/" or (self.task_date.get()[5]) != "/":
            messagebox.showinfo("Invalid Entry", "Incorrect date format, please use DD/MM/YYYY.")
        else:
            self.check_add_valid_date()
```

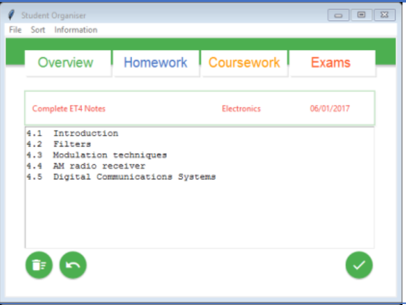
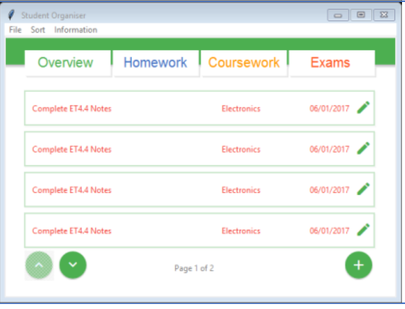
I changed the order of which these tests were completed. I will first check if the user's entry is empty. Followed by this, I will check if the user has not entered 10 characters. Once these has passed, I will then verify if the third and sixth characters are not forward slashes. This order of these steps are very important in this process as there would be an out of range error if the user entered less than 10 characters. This is due to the searching that the will do when finding the third and fifth character in the entry. If there are less than 5 characters, then this error will occur. However, if I check that there are exactly 10 characters first, then this error should never occur. Finally, if these pass, I then split the task up at every forward slash. At this point, there will definitely be two forward slashes in the correct position allowing the split process to work as intended. Checks are performed to verify these three pieces of data are only digits, no non-numerical characters.

```python
def check_add_date(self):
    if self.editPage:
        # VALIDATION: if date is blank, return message
        if self.task_date.get() == "":
            messagebox.showinfo("Invalid Entry", "Date cannot be left blank.")
            pass
        # VALIDATION: if date is shorter than 10 characters, return message
        elif len(self.task_date.get()) != 10:
            messagebox.showinfo("Invalid Entry", "Incorrect date format, please use DD/MM/YYYY.")
            pass
        # VALIDATION: if date does not include a '/' in the relevant positions, return message
        elif (self.task_date.get()[2]) != "/" or (self.task_date.get()[5]) != "/":
            messagebox.showinfo("Invalid Entry", "Incorrect date format, please use DD/MM/YYYY.")
            pass
        # VALIDATION: if date contains non number characters, return message
        else:
            dateData = (self.task_date.get().strip().split('/'))
            self.taskDay = dateData[0]
            self.monthNumber = dateData[1]
            self.taskYear = dateData[2]
            if self.taskDay.isdigit() == False or self.monthNumber.isdigit() == False or self.taskYear.isdigit() == False:
                messagebox.showinfo("Invalid Entry", "Incorrect date format, please use DD/MM/YYYY.")
                pass

            else:
                self.check_add_valid_date()
```

Test	Test Data	Expected Result	Actual Result / Evidence
Change the third character from a '/'	DATE: '06c01/2017'	An error message appears with the text, 'Incorrect date format, please use DD/MM/YYYY.'	 Invalid Entry — Incorrect date format, please use DD/MM/YYYY. OK
Change the sixth character from a '/'	DATE: '06/01-2017'	An error message appears with the text, 'Incorrect date format, please use DD/MM/YYYY.'	 Invalid Entry — Incorrect date format, please use DD/MM/YYYY. OK
Change the third & sixth characters from a '/'	DATE: '06201,2017'	An error message appears with the text, 'Incorrect date format, please use DD/MM/YYYY.'	 Invalid Entry — Incorrect date format, please use DD/MM/YYYY. OK

The adjustments fixed this issue.

Testing the Update Function:

Test	Test Data	Expected Result	Actual Result / Evidence
Edit task 1 (click the crayon button beside the task)	N/A	The program should show the first task's data with a large text area for notes.	 Student Organiser File Sort Information Overview Homework Coursework Exams Complete ET4 Notes Electronics 06/01/2017 4.1 Introduction 4.2 Filters 4.3 Modulation techniques 4.4 AM radio receiver 4.5 Digital Communications Systems
Change the task name	NAME: 'Complete ET4.4 Notes'	The program updates the task's name and shows the page of tasks that the user was previously on.	 Student Organiser File Sort Information Overview Homework Coursework Exams Complete ET4.4 Notes Electronics 06/01/2017 Complete ET4.4 Notes Electronics 06/01/2017 Complete ET4.4 Notes Electronics 06/01/2017 Complete ET4.4 Notes Electronics 06/01/2017 Page 1 of 2

I stopped testing at this point as I had found there was an error in saving the new data to the database. The data that the user had written in this page should only update the task selected. However, the data is written to every task in the database, replacing all their tasks to identical copies of the task being modified. This is a large bug in the program that must be addressed before I continue testing the other entry methods.

There were a few errors with this code. Sometimes, when the task was selected to be modified and then saved, it would sometimes select the wrong task. In addition to this, it would save the changes to every task in the database too, making every task identical. The third issue that I found during this testing process was selecting a task to modify after changing the order of the page with the sorting options.

First, I found that the task ID that was taken by the save_notes_task function was incorrect. Instead of passing the task ID as it is in the database, it would send a value from 0 to 3, depending on the row in the page. To fix this issue, I assigned the 'self.task_id' variable differently within the load_notes function. I changed the code from:

```
self.task_id = self.set[0][4]
```

To the following:

```
self.task_id = (self.set[task_id][4])
```

This new line will select the task's data from the tasks array. Within this selected data set, the fourth item is selected, the task's ID.

Now that the function will operate on the correct task, I still have a few bugs to fix. The next is that the method in which I update the task is very inefficient and would also corrupt the order if the user were to sort by task added. I will use the SQL update command rather than a delete and insert. This command will select any task where a selected column matches a given string. Once the record has been selected, any column can be updated to new values. I will use the task ID to select the task as this is the unique identifier for each task. Then, I will change the following columns: TASK, SUBJECT, DATE, TAB and NOTES. This is the data that is entered by the user and are available for the user to modify.

The function before I changed the SQL query:

```
def save_notes_task(self):
    t1d=self.task_date.get()
    date1=t1d.split('/')
    t1d=date1[2]+"/"+date1[1]+"/"+date1[0]

    self.conn = sqlite3.connect('tasks.db')
        self.conn.execute("DELETE from TASKS where ID='"+ str(self.task_id)+"';")
        lastRow = self.conn.execute("SELECT MAX(id) FROM TASKS")
        for row in lastRow:
            self.id = row[0]
        if self.id == None:
            self.id = 0
        self.conn.execute("INSERT INTO TASKS(ID, TASK, SUBJECT, DATE, TAB, NOTES) values (?, ?, ?, ?, ?, ?)", (self.id+1, self.task_name.get(), self.task_subject.get(), t1d, self.task_type, str(self.tb.get("1.0", END))
))

    self.conn.commit()
    self.conn.execute("DELETE from TASKS where ID='"+ str(self.sets_id[0])+"';")
    self.refresh_task_list_remove()
```

The function with the update SQL statement:

```
def save_notes_task(self):
    t1d=self.task_date.get()
    date1=t1d.split('/')
    t1d=date1[2]+"/"+date1[1]+"/"+date1[0]

    self.conn = sqlite3.connect('tasks.db')
    sql = ''' UPDATE TASKS
              SET TASK = ? ,
                  SUBJECT = ? ,
                  DATE = ?,
                  TAB = ?,
                  NOTES = ?
              WHERE ID = ?'''
    cur = self.conn.cursor()
    cur.execute(sql, (self.task_name.get(), self.task_subject.get(), t1d, self.task_type, str(self.tb.get("1.0", END)), str(self.task_id)))

    self.conn.commit()
    self.refresh_task_list_remove()
```

When a task is modified now, the correct task will be selected and the data entered in the page will be saved to the task in the database. The ID of the task will not be changed so sorting by task added will no longer show an incorrect order.

The final bug to correct is caused by changing the order of the tasks before selecting a task to be modified. The error that I had received was, “IndexError: list index out of range.” This error occurred when the load_notes function was called. The error means that some data could not be inserted into the page correctly as some was missing. The list index was out of range, meaning that there was insufficient data within the array. There are only two places in this program where the data from the databases is appended to the tasks array. The first is whilst the program first loads during the setup_data function. The error must be in the second place as that function is used once a sorting option has been applied to a page.

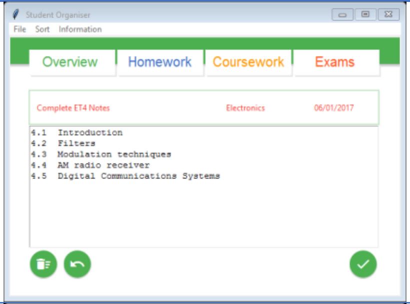
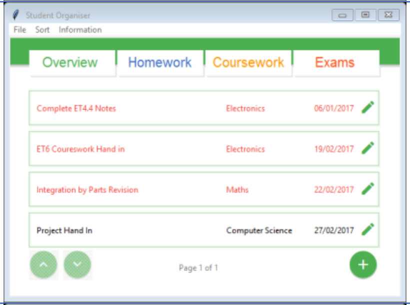
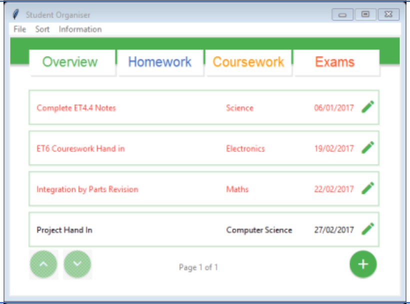
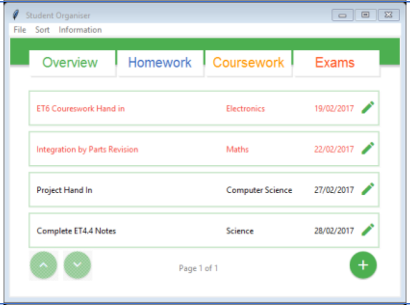
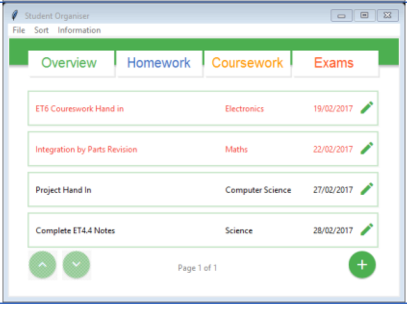
I found that I haven’t updated the refresh_task_list function which uses the data read from the database in the selected order. When appending this data to the tasks array, the notes were not added. This was because I had forgotten to update this function after I had added in the notes feature into the program, I had only included this at the start of the program.

The function before I had appended the notes data to the tasks array.

```
1.  def refresh_task_list(self):
2.      for row in self.cursor:
3.          single = []                 #Create an array for each task
4.          single.append(row[1])       #Add name
5.          single.append(row[2])       #Add subject
6.          # Reverse the date format (YYYY/MMM/DD => DD/MM/YYYY)
7.          dateData = (row[3].strip().split('/'))
8.          dateDay = dateData[2]
9.          dateMonth = dateData[1]
10.         dateYear = dateData[0]
11.         dateNew = dateDay + "/" + dateMonth + "/" + dateYear
12.         single.append(dateNew)      #Add date
13.         single.append(row[4])       #Add type
14.         single.append(row[0])       #Add ID
15.         self.tasks.append(single)   #Create an array of each task's array
16.     self.conn.close()
17.     self.load_rows()
```

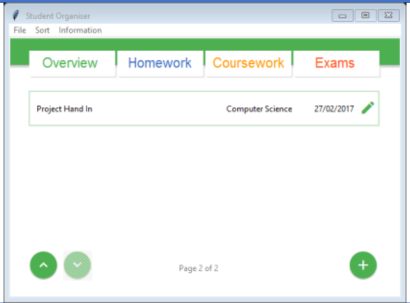
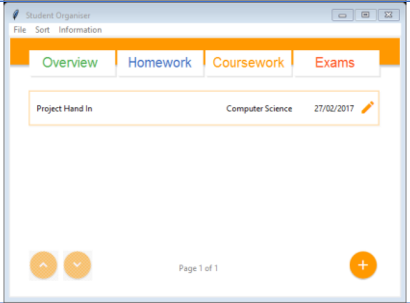
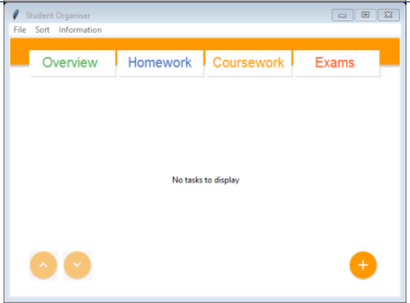
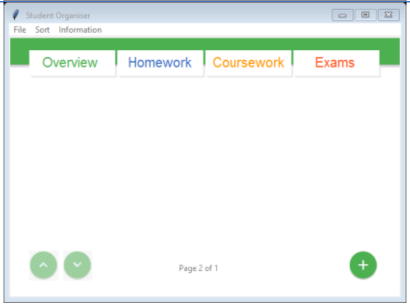
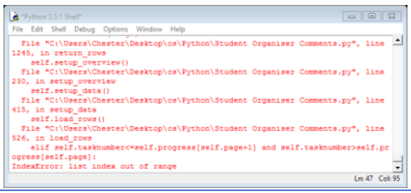
The amendment can be found on line 15, where the fifth piece of task data is added to the tasks array. This data contains the notes of the task.

```
1.  def refresh_task_list(self):
2.      for row in self.cursor:
3.          single = []                 #Create an array for each task
4.          single.append(row[1])       #Add name
5.          single.append(row[2])       #Add subject
6.          # Reverse the date format (YYYY/MMM/DD => DD/MM/YYYY)
7.          dateData = (row[3].strip().split('/'))
8.          dateDay = dateData[2]
9.          dateMonth = dateData[1]
10.         dateYear = dateData[0]
11.         dateNew = dateDay + "/" + dateMonth + "/" + dateYear
12.         single.append(dateNew)      #Add date
13.         single.append(row[4])       #Add type
14.         single.append(row[0])       #Add ID
15.         single.append(row[5])       #Add notes
16.         self.tasks.append(single)   #Create an array of each task's array
17.     self.conn.close()
18.     self.load_rows()
```

Test	Test Data	Expected Result	Actual Result / Evidence
Edit task 1 (click the crayon button beside the task)	N/A	The program should show the first task's data with a large text area for notes.	 Student Organiser File Sort Information Overview Homework Coursework Exams Complete ET4 Notes Electronics 06/01/2017 4.1 Introduction 4.2 Filters 4.3 Modulation techniques 4.4 AM radio receiver 4.5 Digital Communications Systems
Change the task name	NAME: 'Complete ET4.4 Notes'	The program updates the task's name and shows the page of tasks that the user was previously on.	 Student Organiser File Sort Information Overview Homework Coursework Exams Complete ET4.4 Notes Electronics 06/01/2017 ET6 Coureswork Hand in Electronics 19/02/2017 Integration by Parts Revision Maths 22/02/2017 Project Hand In Computer Science 27/02/2017 Page 1 of 1
Change the subject	SUBJECT: 'Science'	The program updates the subject of this task and shows the page of tasks that the user was previously on.	 Student Organiser File Sort Information Overview Homework Coursework Exams Complete ET4.4 Notes Science 06/01/2017 ET6 Coureswork Hand in Electronics 19/02/2017 Integration by Parts Revision Maths 22/02/2017 Project Hand In Computer Science 27/02/2017 Page 1 of 1
Change the due date to after the current date	DATE: '28/02/2017'	The program updates the date of this task and shows the page of tasks that the user was previously on. The task should be shown in black text, not red.	 Student Organiser File Sort Information Overview Homework Coursework Exams ET6 Coureswork Hand in Electronics 19/02/2017 Integration by Parts Revision Maths 22/02/2017 Project Hand In Computer Science 27/02/2017 Complete ET4.4 Notes Science 28/02/2017 Page 1 of 1
Delete the notes	NOTES: ''	The program updates the notes for this task and shows the page of tasks that the user was previously on. This is the only field that is allowed to be saved with no content.	 Student Organiser File Sort Information Overview Homework Coursework Exams ET6 Coureswork Hand in Electronics 19/02/2017 Integration by Parts Revision Maths 22/02/2017 Project Hand In Computer Science 27/02/2017 Complete ET4.4 Notes Science 28/02/2017 Page 1 of 1

When changing each piece of data of the task, the database updated with these new values successfully. As a result of these five tests, I can conclude that any modifications made to any of the tasks data will be saved. Upon saving, the previous tasks page will be loaded.

Testing the Delete Function:

Test	Test Data	Expected Result	Actual Result / Evidence
Scroll to a page with a single task remaining	N/A	The program should show the last task on the tasks page.	 Student Organiser File Sort Information Overview Homework Coursework Exams Page 2 of 2 Project Hand In Computer Science 27/02/2017
Select the tab which this task is saved as	N/A	The program will show this task on the task's type page.	 Student Organiser File Sort Information Overview Homework Coursework Exams Page 1 of 1 Project Hand In Computer Science 27/02/2017
Delete the task	N/A	The task page will be empty and the message, 'No tasks to display' will be displayed.	 Student Organiser File Sort Information Overview Homework Coursework Exams No tasks to display
Select the overview tab	N/A	The task page will scroll to the previous page or it would be empty and the message, 'No tasks to display' will be displayed.	 Student Organiser File Sort Information Overview Homework Coursework Exams Page 2 of 1  *Python 3.5.1 Shell* File Edit Shell Debug Options Window Help Ln: 47 Col: 95 File "C:\Users\Chester\Desktop\cs\Python\Student Organiser Comments.py", line 1245, in return_rows self.setup_overview() File "C:\Users\Chester\Desktop\cs\Python\Student Organiser Comments.py", line 230, in setup_overview self.setup_data() File "C:\Users\Chester\Desktop\cs\Python\Student Organiser Comments.py", line 415, in setup_data self.load_rows() File "C:\Users\Chester\Desktop\cs\Python\Student Organiser Comments.py", line 526, in load_rows elif self.tasknumber<=self.progress[self.page+1] and self.tasknumber>self.progress[self.page]: IndexError: list index out of range

The reason for this sequence of tests was to test that the delete button would remove the task from the database entirely. By switching pages, this would show that the task had been completely removed as it wouldn't appear anywhere in the program where it had previously.

This test had highlighted an issue with the program. The overview page caused an error because it was showing a page with no tasks that was not page one. Usually, if the last task on a page was to be deleted, the page counter would reduce by two, hence the previous page would be displayed. However, if the current page is the first page, then the message, 'No tasks to display' will be shown.

As this task was removed from another page, the overview page counter had not been adjusted to this change. To fix this issue I changed certain page counters to '0' upon loading the setup_data function. I will only change three of these page counters where the one that will not be adjusted is controlling the current tab. By doing so, when switching a tab, the tab will begin at the first page every time.

The part of the function that I have updated to fix this bug:

```
def setup_data(self):
    if self.tab == "Overview":
        self.colour = "#4caf50"
        self.rowColour = "#c8e6c9"
        self.add = self.greenAdd
        self.delete = self.greenDelete
        self.remove = self.greenRemove
        self.hwpage=0
        self.cwpage=0
        self.expage=0

    if self.tab == "Homework":
        self.colour = "#4472C4"
        self.rowColour = "#bbdefb"
        self.add = self.blueAdd
        self.delete = self.blueDelete
        self.remove = self.blueRemove
        self.ovpage=0
        self.cwpage=0
        self.expage=0

    if self.tab == "Coursework":
        self.colour = "#ff9800"
        self.rowColour = "#ffe0b2"
        self.add = self.yellowAdd
        self.delete = self.yellowDelete
        self.remove = self.yellowRemove
        self.ovpage=0
        self.hwpage=0
        self.expage=0

    if self.tab == "Exam":
        self.colour = "#ff5722"
        self.rowColour = "#ffcdd2"
        self.add = self.redAdd
        self.delete = self.redDelete
        self.remove = self.redRemove
        self.ovpage=0
        self.hwpage=0
        self.cwpage=0
```

Development Review:

This is the last development cycle before final testing. In this previous cycle, I have added the information section where the school's contact details, map and useful websites can be accessed.

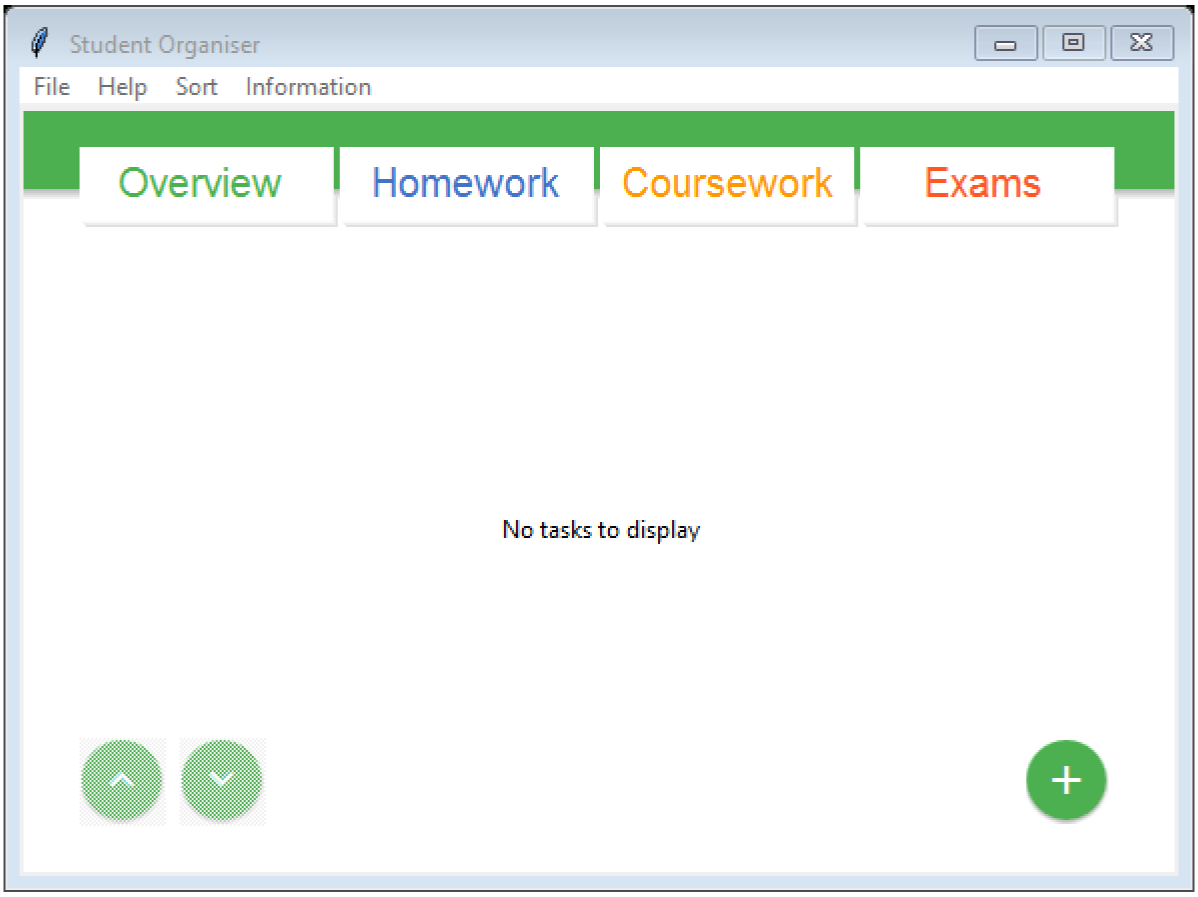
The last development was focused upon adding in this information about the school and any academic content that student's may find useful. I have not fully met his criteria as it sates, "There should be school information available to access (including: a school map, term dates, contact details and website links)." I have not included any term dates in this software and this will be included in future development, however, all other requirements in this specification has been met thus far.

In addition to this, I had adjusted the program to user feedback where a task should be able to be modified in a new page. Where any task selected can be loaded onto its own, separate page with all of the data presented and an area to add notes about the task. Due to this request, the notes section that would be accessed through the menu bar will no longer be developed.

3.4 - EVALUATION

3.4.1 - Testing to Inform Evaluation:

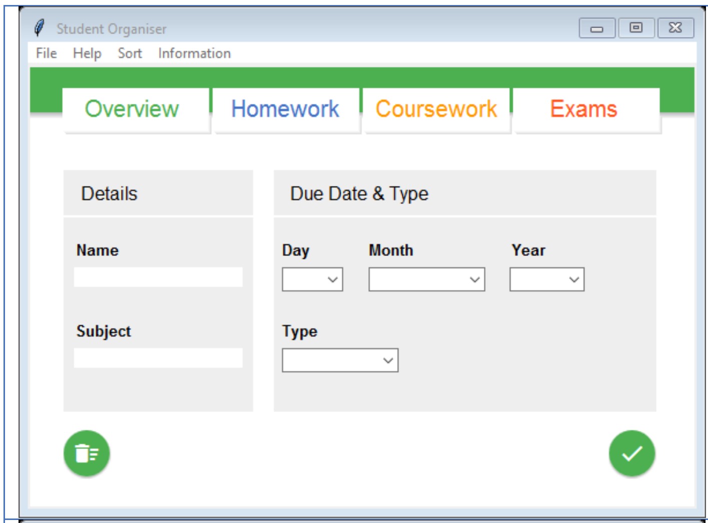
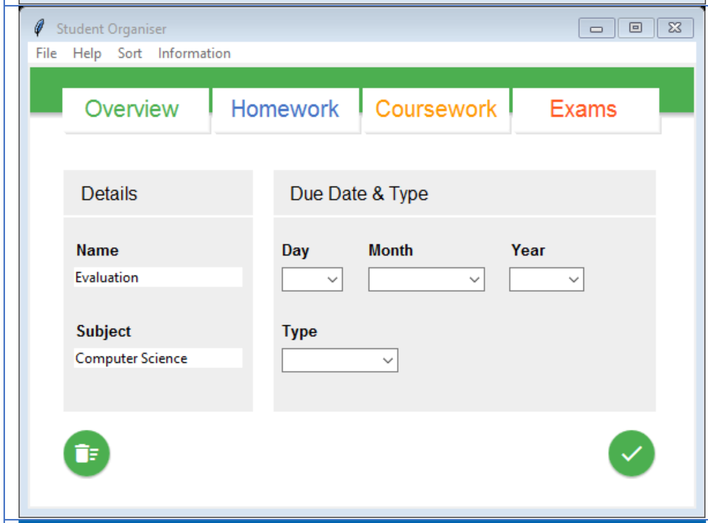
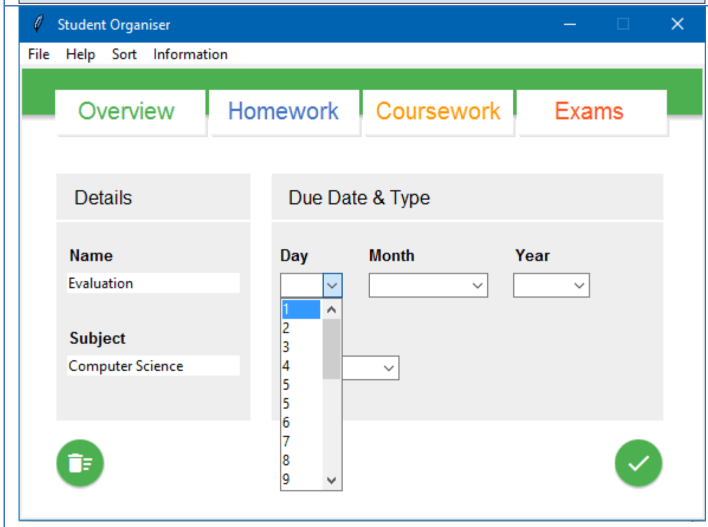
Testing the Solution:

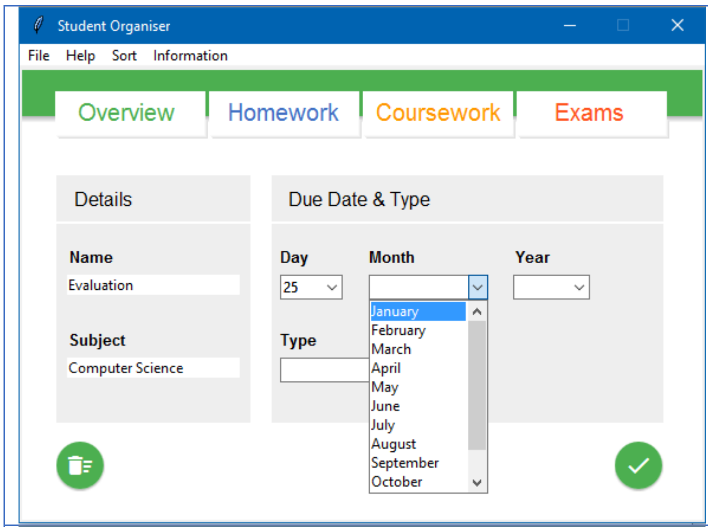


No tasks to display

When the program first loads, this page will be shown. This is the overview page that shows every task that has been saved in this program in ascending order sorted by the task date. As there are no tasks written in this program, the message, “No tasks to display” is shown. There are also less than four tasks to display on this page which causes the page up and page down buttons to both be disabled. In addition, the page counter will also be hidden as there are zero pages available. All of these features have worked as expected as seen in the screenshot above.

Adding a Task:

 Student Organiser · File Help Sort Information · Overview Homework Coursework Exams · Details: Name, Subject · Due Date & Type: Day, Month, Year, Type	When the add task button has been clicked, this page is loaded. There are six entry methods available, all compulsory. The page has been constructed correctly and all the correct entry methods have been loaded with their respective heading.
 Student Organiser · File Help Sort Information · Overview Homework Coursework Exams · Details: Name: Evaluation, Subject: Computer Science · Due Date & Type: Day, Month, Year, Type	The task name and subject are entered through the two entry boxes in the left box. Any value can be entered into these but they must not be left empty. I have filled in the task name and the subject appropriately. The input which I have just written should be allowed by the validation functions so that the data is written to the database.
 Student Organiser · File Help Sort Information · Overview Homework Coursework Exams · Details: Name: Evaluation, Subject: Computer Science · Due Date & Type: Day (1, 2, 3, 4, 5, 5, 6, 7, 8, 9), Month, Year	When setting the due date of the task, the user must select three options from each dropdown menu. The first is the day. To select the day, the user should click the dropdown menu and a list of every available day should appear. The list of integers begins at '1' and continues up to and including '31.' The options here are independent from the other two variables that make up the date. I had decided that this would be the best method in case the user was to change their selection for any of the other two values and accidently reset any other previously selected values.

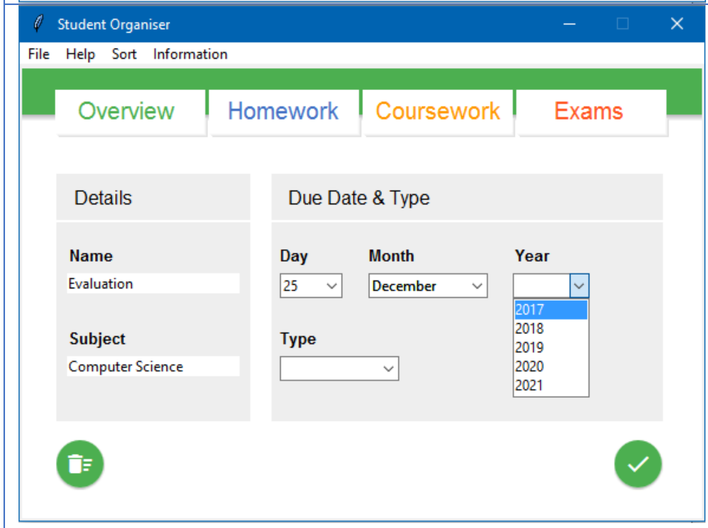


Name: Evaluation
Subject: Computer Science
Day: 25
Month:
January
February
March
April
May
June
July
August
September
October
Year:
Type:

The month can be selected in a similar method.

The only difference is that the user will be presented with the names of each month in chronological order which allows for a more user friendly interface. Each month is listed and the user should simply select the month that applies.

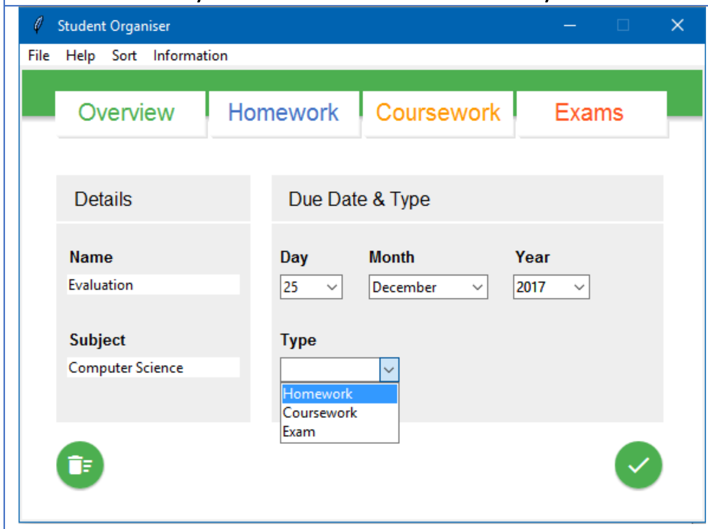
Here, every month had displayed correctly and in order.



Name: Evaluation
Subject: Computer Science
Day: 25
Month: December
Year:
2017
2018
2019
2020
2021
Type:

The years displayed within this dropdown are dynamic. These values will vary depending upon the year in which the user adds a task. To ensure that the user wouldn't have to scroll a long way or struggle to find a year, I decided that the program will output the soonest year as the current year with the following four consecutive years.

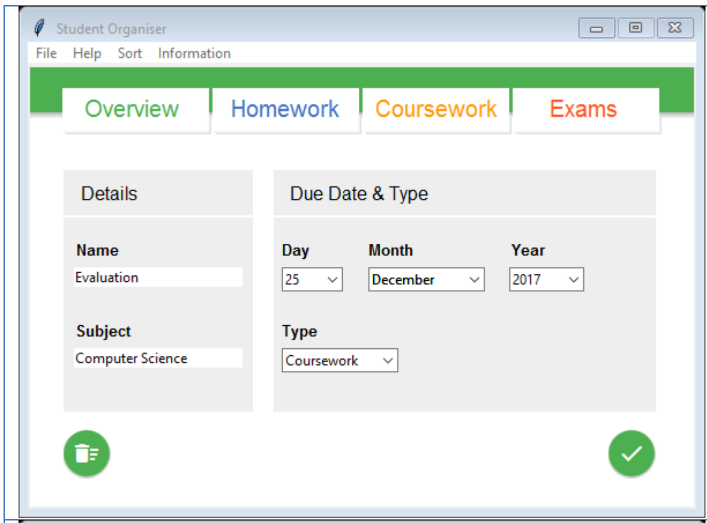
I had chosen these five years as the purpose of this program is to act as a student organizer and it is unlikely that a student would have any tasks due outside of this five-year duration. As the current year is 2017, it has successfully inserted the five choices of year.



Name: Evaluation
Subject: Computer Science
Day: 25
Month: December
Year: 2017
Type:
Homework
Coursework
Exam

The final detail that must be given is the task type. This will be used when displaying the tasks on different pages. There are four tabs but three options. The first tab, 'overview' will show every task regardless of the type so that the user can clearly see an overview of every upcoming task without any need to keep switching between the three pages.

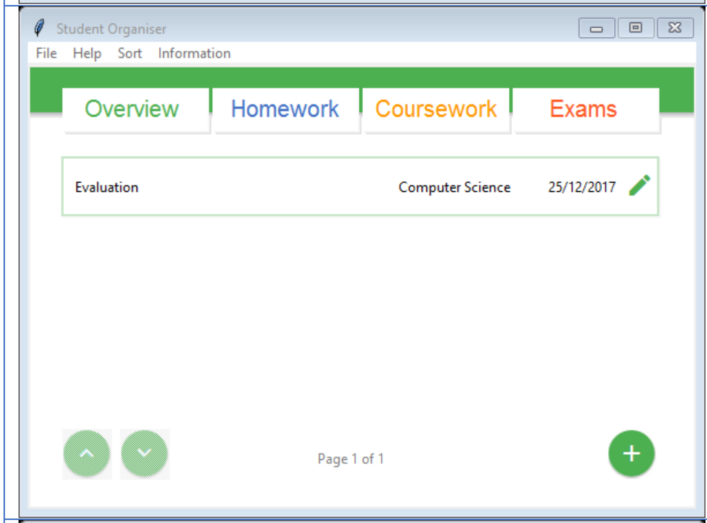
The three available tabs have been made available in the dropdown menu and also in order of the tab along the top of this program.



Name: Evaluation
Subject: Computer Science
Day: 25
Month: December
Year: 2017
Type: Coursework

Now that all the relevant information has been inserted, the user can now click the tick button to submit this into the database.

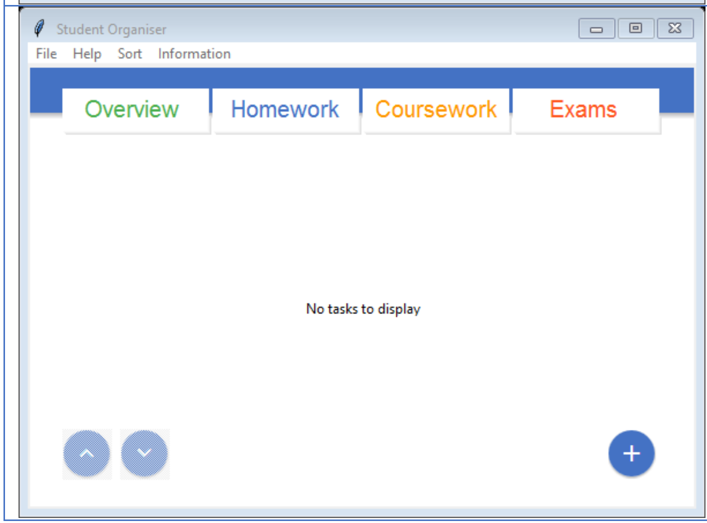
There is also the option to delete all of this data and return to the previous page by using the delete button.



Evaluation Computer Science 25/12/2017

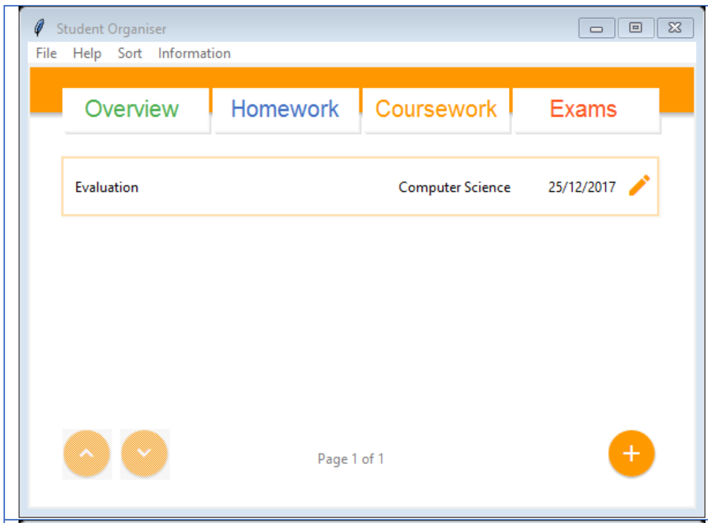
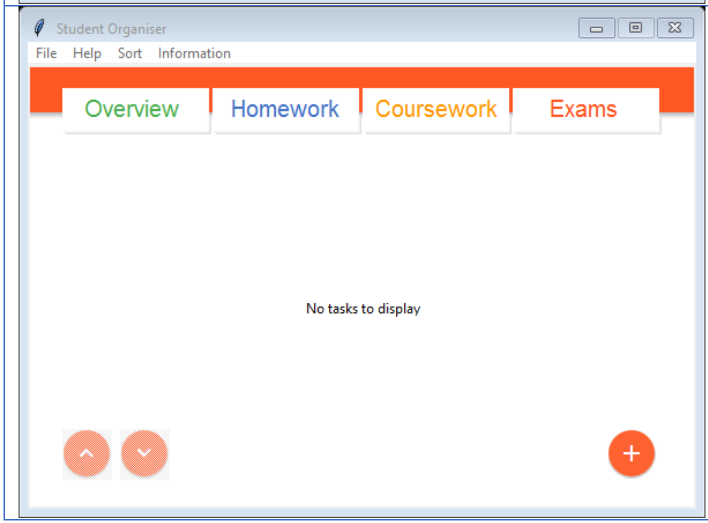
Page 1 of 1

As the data that I had entered was valid, the task had been successfully written to the database. There were no dialogue boxes displayed informing the user of invalid data either. The overview tasks page is loaded as a result. There is only one task shown on this page as I have only created this task.

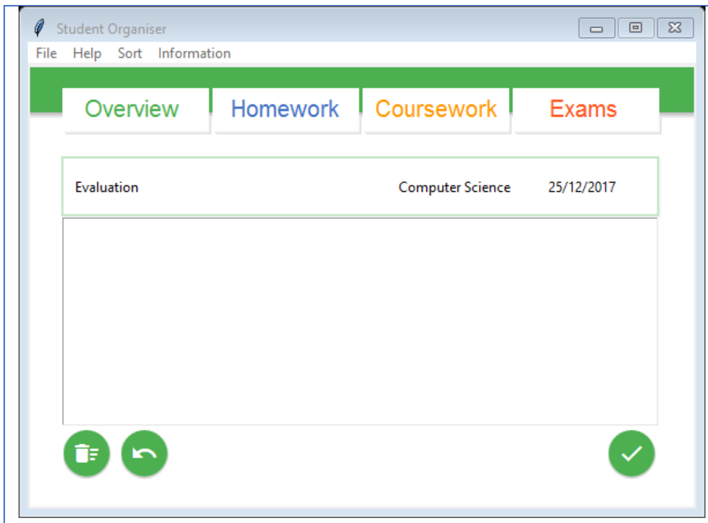


No tasks to display

If the user switches to the homework tab, no tasks have been shown. This is because there are no tasks with the task type set to homework.

 Student Organiser File Help Sort Information Overview Homework Coursework Exams Evaluation Computer Science 25/12/2017	However, on the coursework tab, this task is displayed. This has worked as expected and the task that I had previously created is positioned on the correct page.
 Student Organiser File Help Sort Information Overview Homework Coursework Exams No tasks to display	On the exams page, there are no tasks available. This is the correct outcome as no tasks have the exam task type selected.

Modifying a Task:



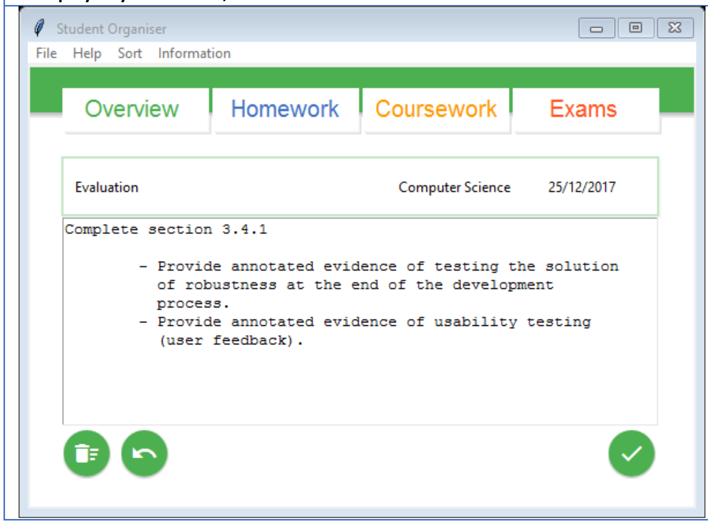
Evaluation Computer Science 25/12/2017

If the user were to click the crayon icon next to the task, this page will be shown. Here the task is loaded into a new screen where the user has the ability to make any changes to the task's data.

The name and the subject can be altered where the same validation checks will run when adding the task once the user clicks the save button.

The date can also be modified. Instead of selecting an integer day, followed by a string month and an integer year, all values entered are integers. The format has changed so that the user will now need to enter a date with the following format, 'DD/MM/YYYY.' The date entered will be validated to ensure that it is a real date before saving to the database.

A large text area can be found below. This can be used for the user to insert any notes for this specific task. This field can be left blank too as it should not be compulsory for the student to set notes on their tasks if they feel they do not need them. When the task is first created, this field is empty by default, as shown in the screenshot above.

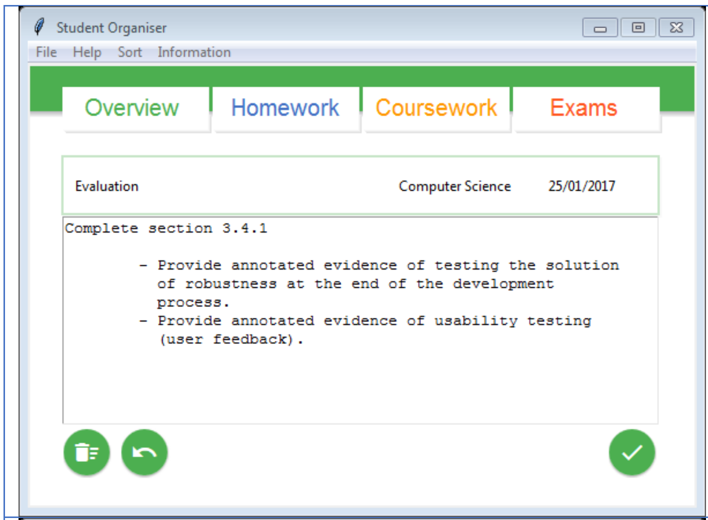
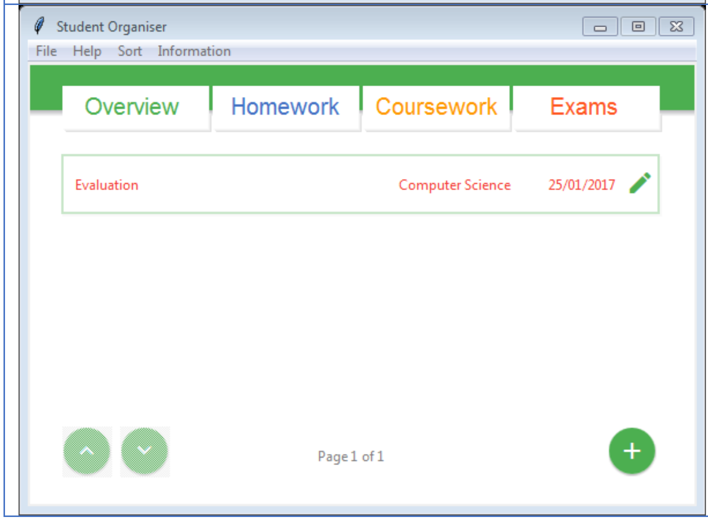


Evaluation Computer Science 25/12/2017

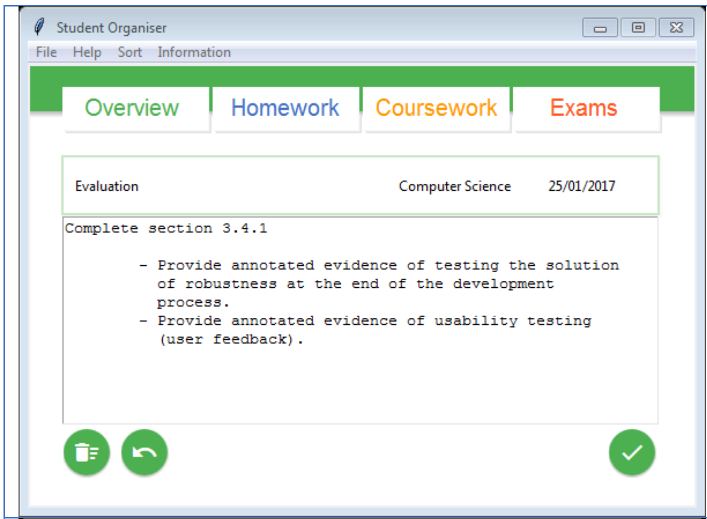
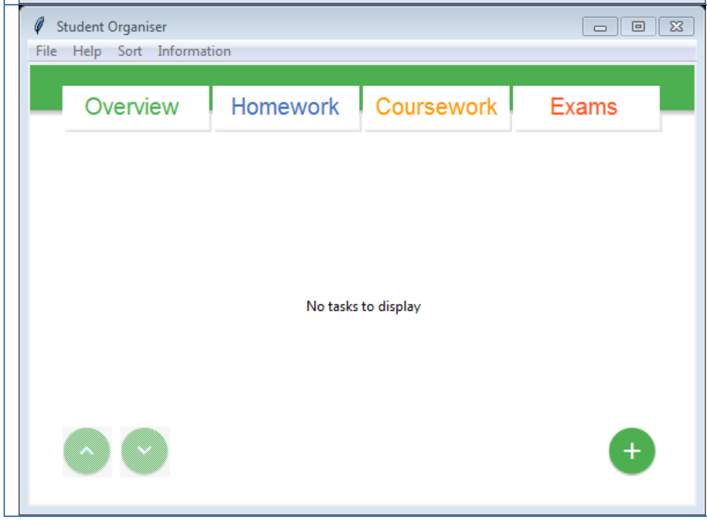
```
Complete section 3.4.1

        - Provide annotated evidence of testing the solution
          of robustness at the end of the development
          process.
        - Provide annotated evidence of usability testing
          (user feedback).
```

If the user were to add these notes to the task, it should save this data to the database. When the user wants to view these notes, they can simply click the crayon icon and view this screen again.

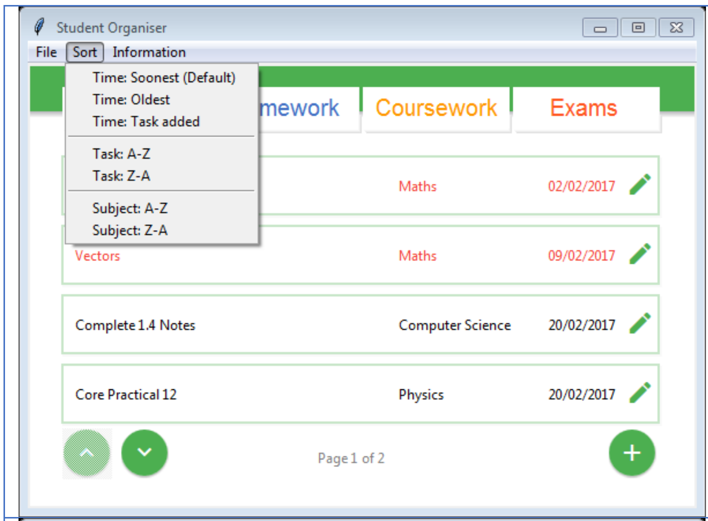
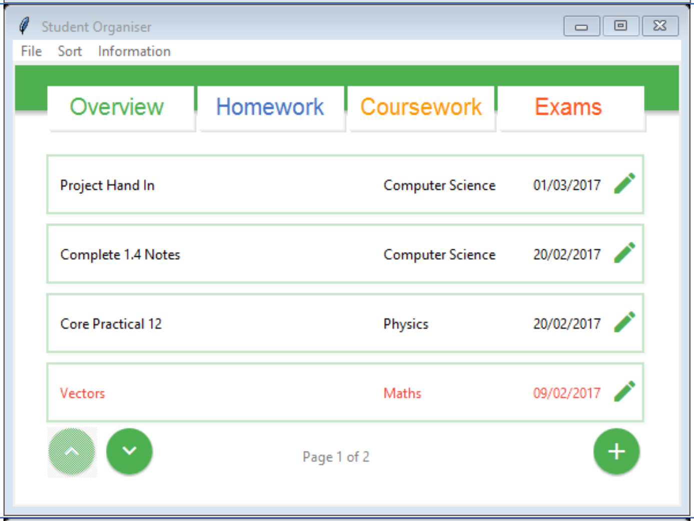
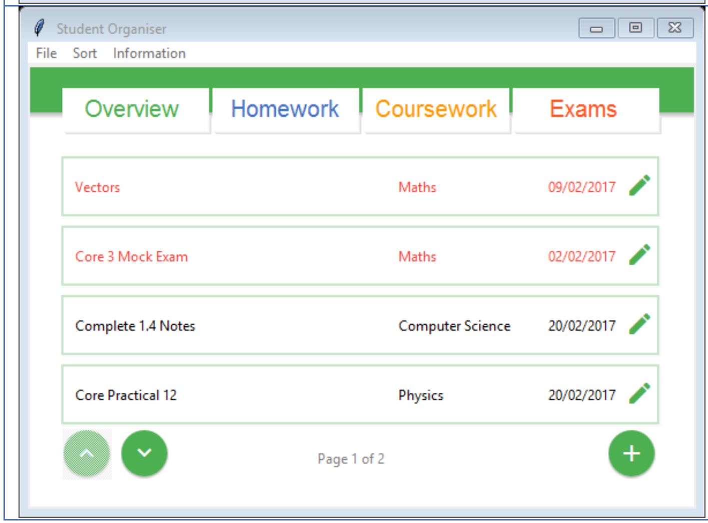
If they do not want to save any changes to the data, they can simply click the back arrow which will return the user to their previous page. Alternatively, if they would like to delete this task, they can click the bin icon to permanently remove the task's data from the database.

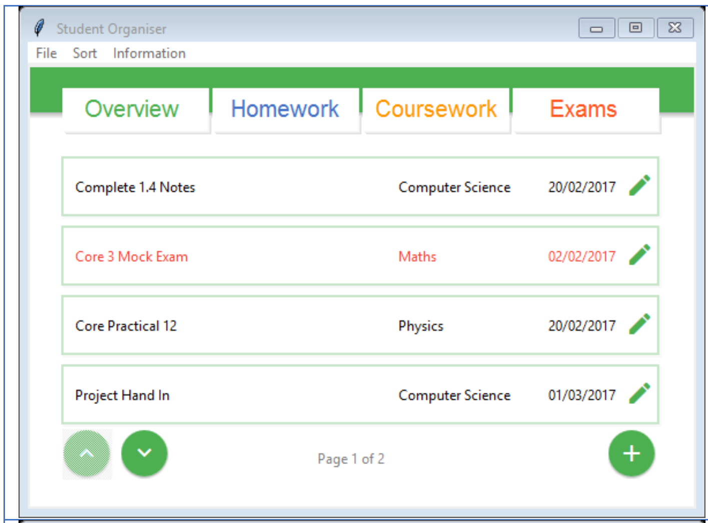
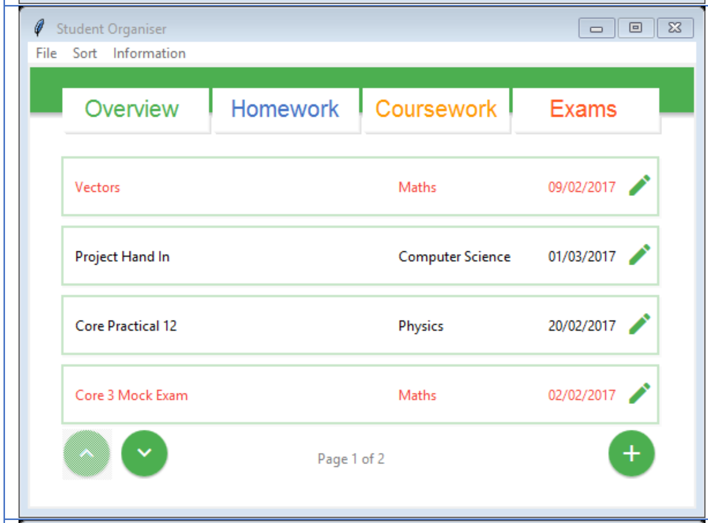
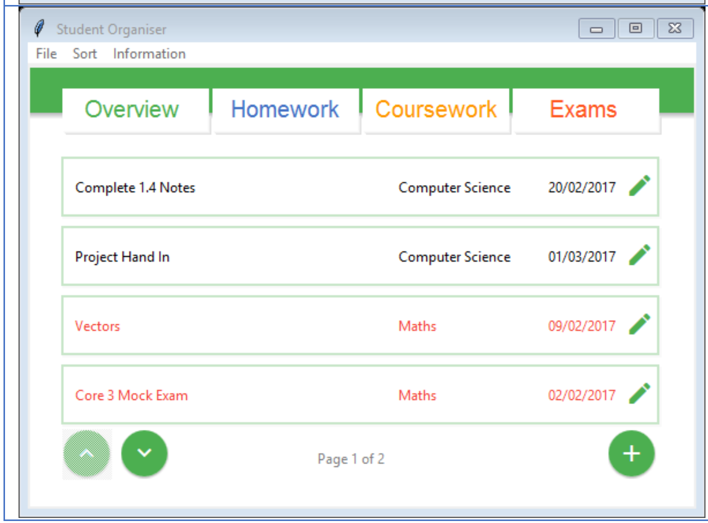
 Student Organiser File Help Sort Information Overview Homework Coursework Exams Evaluation Computer Science 25/01/2017 Complete section 3.4.1 - Provide annotated evidence of testing the solution of robustness at the end of the development process. - Provide annotated evidence of usability testing (user feedback).	Here I have changed the date so that it will now be due for 25/01/2017. By clicking the save button, the program will return to the previous page (overview) showing the adjustments.
 Student Organiser File Help Sort Information Overview Homework Coursework Exams Page 1 of 1 Evaluation Computer Science 25/01/2017	The date has been updated and the task has been highlighted in red. This is because the current date is 06/02/2017 and this task has been classed as overdue.

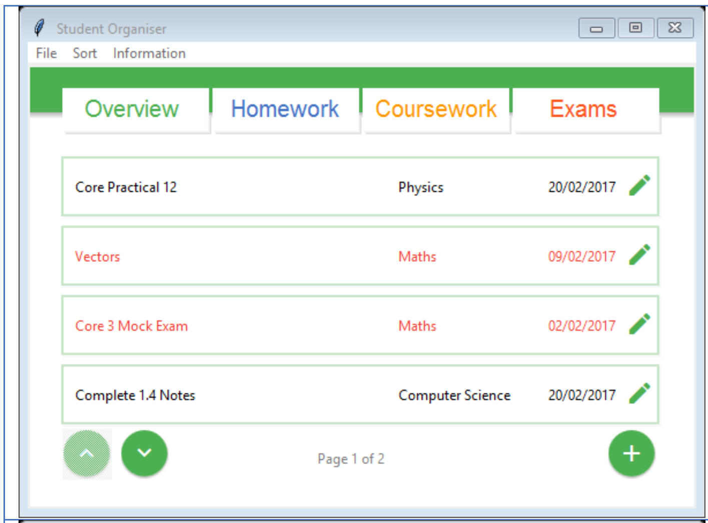
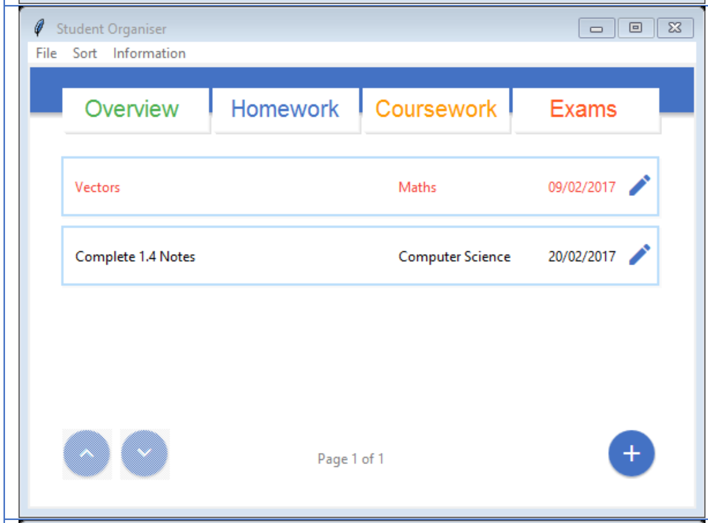
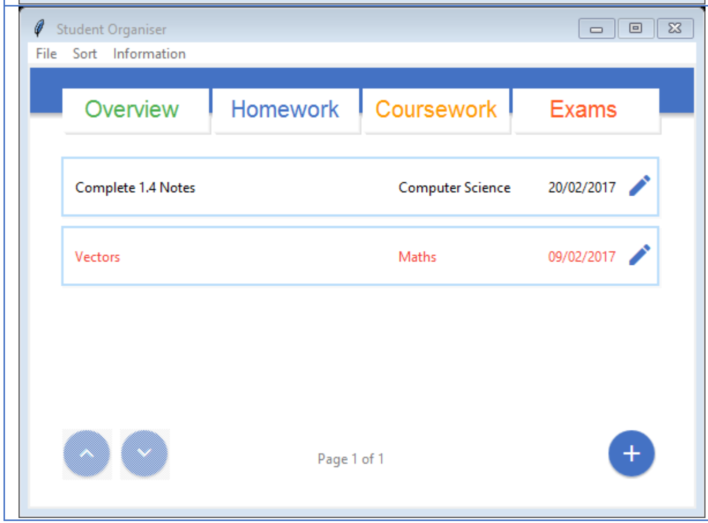
Deleting a Task:

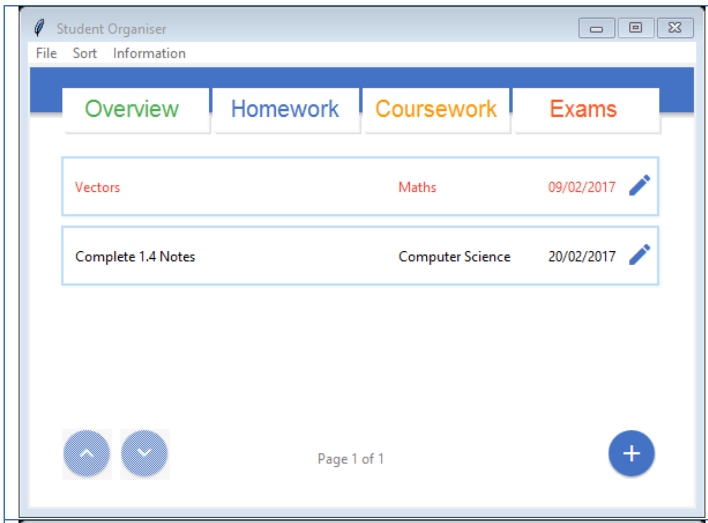
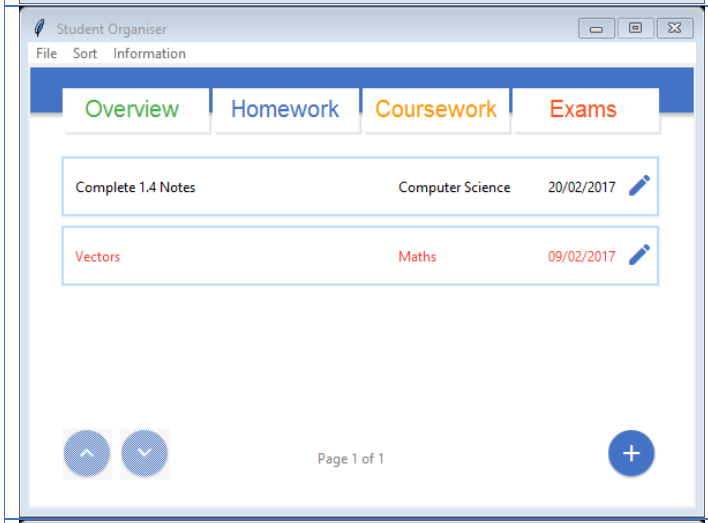
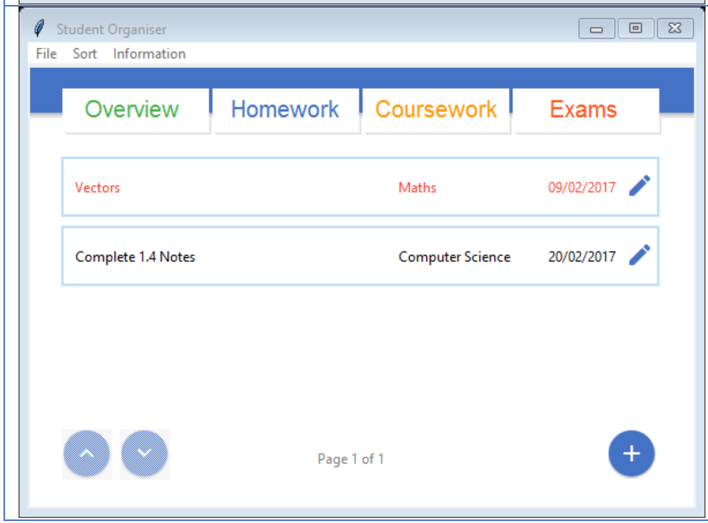
 Student Organiser File Help Sort Information Overview Homework Coursework Exams Evaluation Computer Science 25/01/2017 Complete section 3.4.1 - Provide annotated evidence of testing the solution of robustness at the end of the development process. - Provide annotated evidence of usability testing (user feedback).	To delete a task, first it must be opened for modifications by clicking the crayon icon next to the task to be removed. The task's data is then loaded into a new screen where the user has the ability to make any changes to the task's data.
 Student Organiser File Help Sort Information Overview Homework Coursework Exams No tasks to display	Once the delete button has been pressed, the task's record in the databases is removed. The previous page is then shown and as there are no tasks available, the following message is displayed, "No tasks to display." Additionally, the page counter is also hidden and the page scrolling buttons are both disabled.

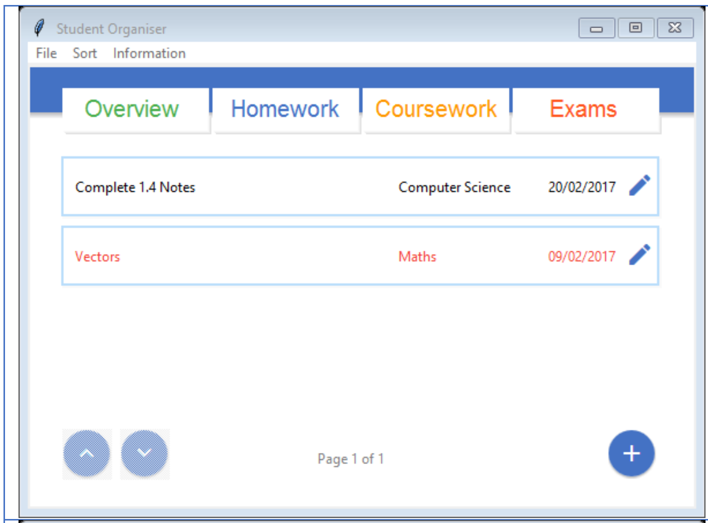
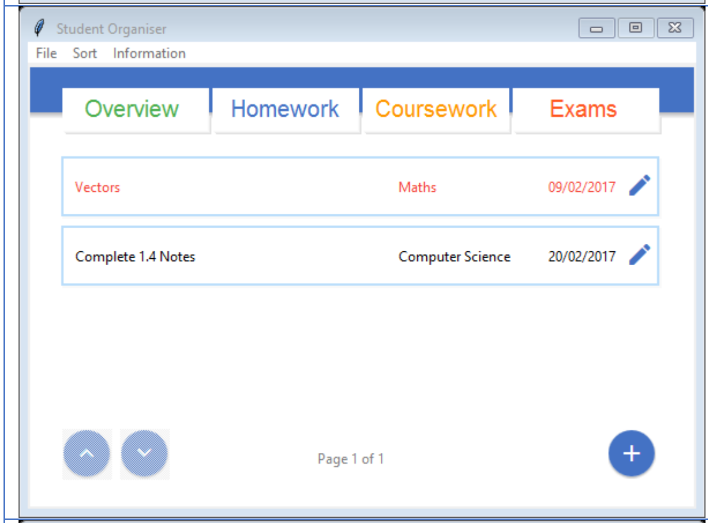
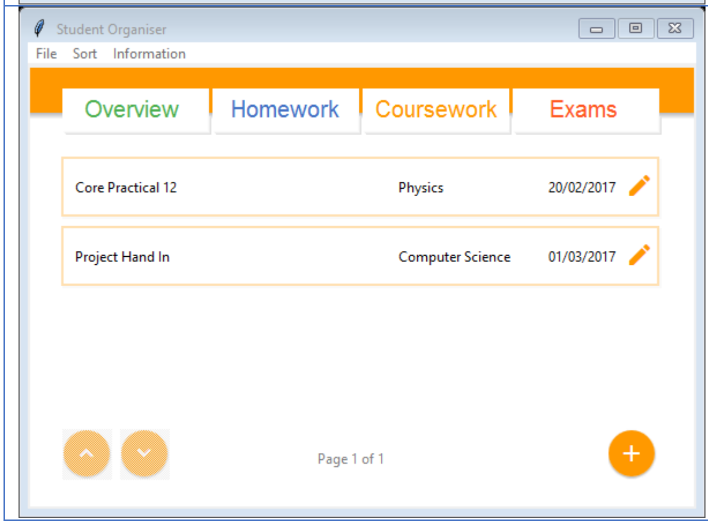
Sorting Tasks:

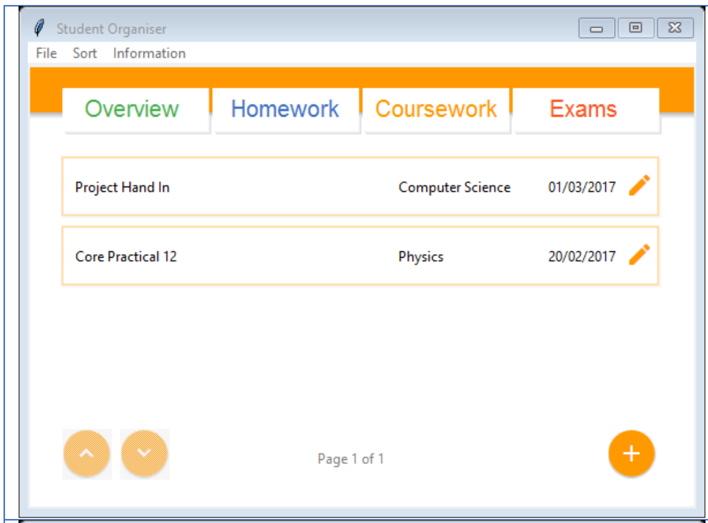
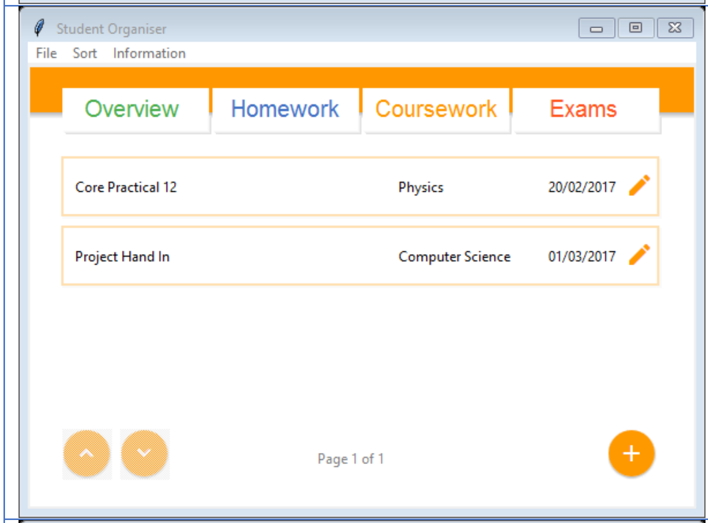
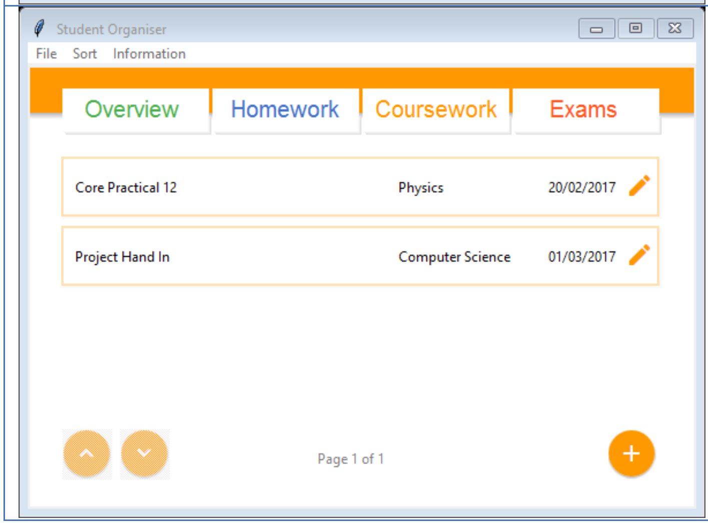
 Student Organiser File Sort Information Time: Soonest (Default) Time: Oldest Time: Task added Task: A-Z Task: Z-A Subject: A-Z Subject: Z-A mework Coursework Exams Maths 02/02/2017 Vectors Maths 09/02/2017 Complete 1.4 Notes Computer Science 20/02/2017 Core Practical 12 Physics 20/02/2017 Page 1 of 2	Here is the program with five sample tasks written in. Each task is different so that the sorting algorithms can be clearly demonstrated. The sort button in the menu bar will list all methods of sorting the tasks. By clicking any of these methods, the tasks below will rearrange to the selected option.
 Student Organiser File Sort Information Overview Homework Coursework Exams Project Hand In Computer Science 01/03/2017 Complete 1.4 Notes Computer Science 20/02/2017 Core Practical 12 Physics 20/02/2017 Vectors Maths 09/02/2017 Page 1 of 2	Sort by time oldest: The tasks changed order so that the task due soonest will appear last.
 Student Organiser File Sort Information Overview Homework Coursework Exams Vectors Maths 09/02/2017 Core 3 Mock Exam Maths 02/02/2017 Complete 1.4 Notes Computer Science 20/02/2017 Core Practical 12 Physics 20/02/2017 Page 1 of 2	Sort by time of task added: The tasks changed order so that the task that I had added first will appear soonest.

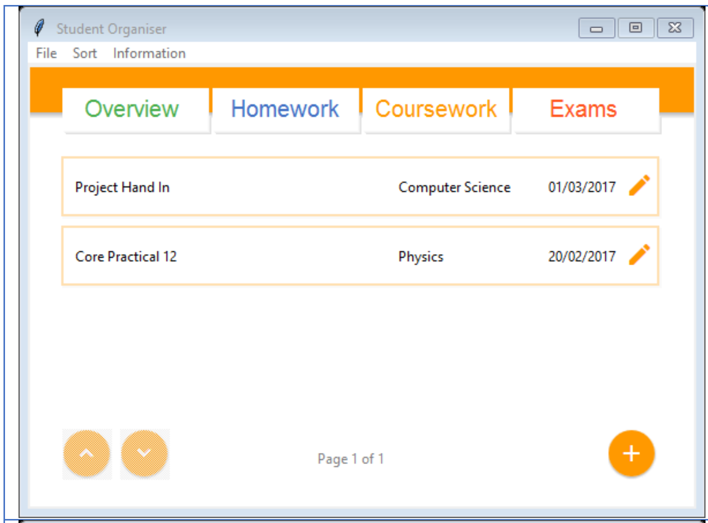
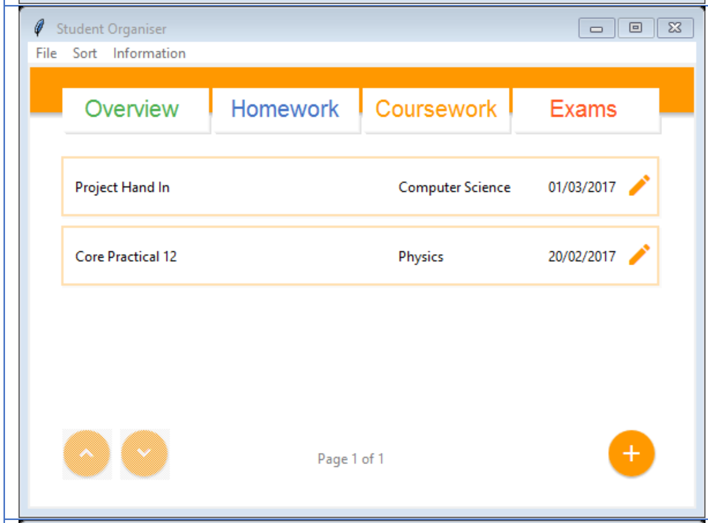
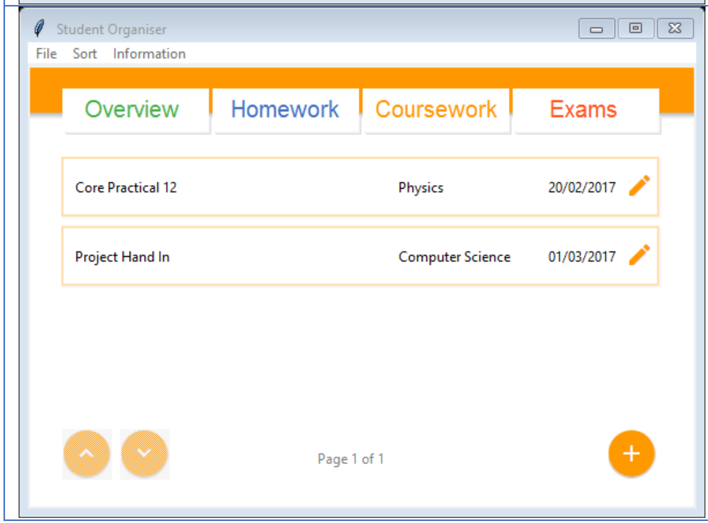
 Student Organiser File Sort Information Overview Homework Coursework Exams Complete 1.4 Notes — Computer Science — 20/02/2017 Core 3 Mock Exam — Maths — 02/02/2017 Core Practical 12 — Physics — 20/02/2017 Project Hand In — Computer Science — 01/03/2017 Page 1 of 2	Sort by task A-Z: The tasks changed order so that the tasks appear in ascending alphabetical order by their name.
 Student Organiser File Sort Information Overview Homework Coursework Exams Vectors — Maths — 09/02/2017 Project Hand In — Computer Science — 01/03/2017 Core Practical 12 — Physics — 20/02/2017 Core 3 Mock Exam — Maths — 02/02/2017 Page 1 of 2	Sort by task Z-A: The tasks changed order so that the tasks appear in descending alphabetical order by their name.
 Student Organiser File Sort Information Overview Homework Coursework Exams Complete 1.4 Notes — Computer Science — 20/02/2017 Project Hand In — Computer Science — 01/03/2017 Vectors — Maths — 09/02/2017 Core 3 Mock Exam — Maths — 02/02/2017 Page 1 of 2	Sort by subject A-Z: The tasks changed order so that the tasks appear in ascending alphabetical order by their subject.

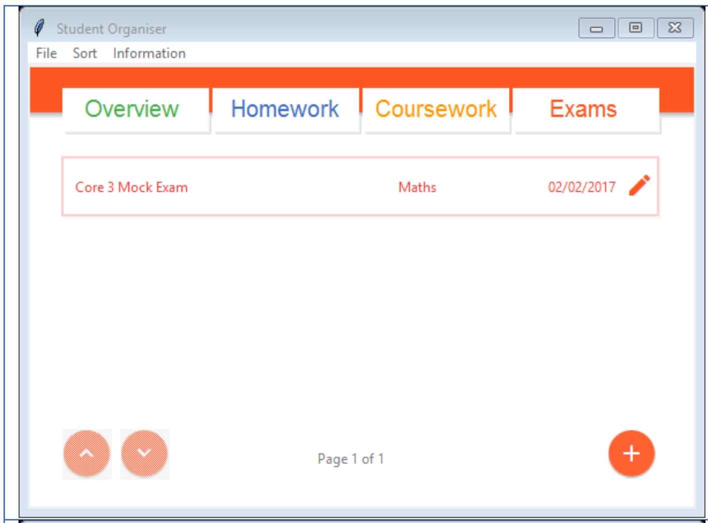
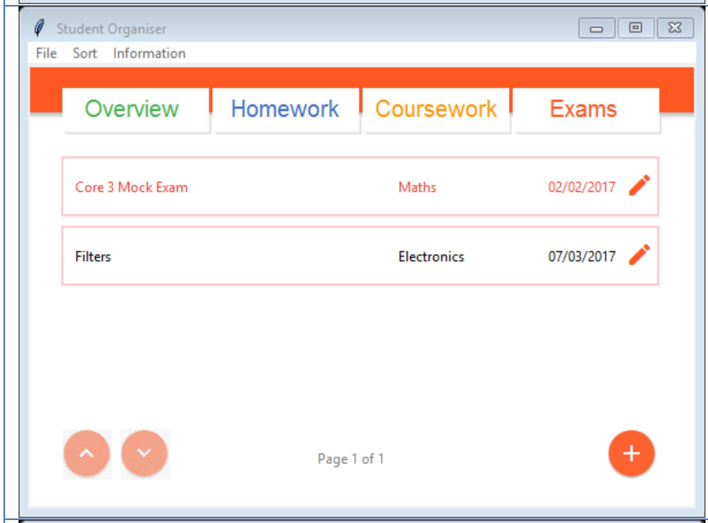
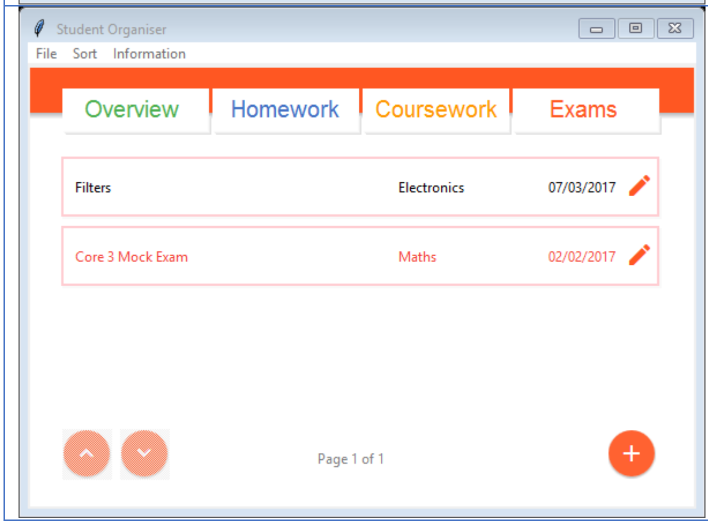
 Student Organiser File Sort Information Overview Homework Coursework Exams Core Practical 12 Physics 20/02/2017 Vectors Maths 09/02/2017 Core 3 Mock Exam Maths 02/02/2017 Complete 1.4 Notes Computer Science 20/02/2017 Page 1 of 2	Sort by subject Z-A: The tasks changed order so that the tasks appear in descending alphabetical order by their subject.
 Student Organiser File Sort Information Overview Homework Coursework Exams Vectors Maths 09/02/2017 Complete 1.4 Notes Computer Science 20/02/2017 Page 1 of 1	Sorting the tasks should work on all four tabs within the program, individually. Upon loading the homework tab, each task is sorted by time due soonest by default.
 Student Organiser File Sort Information Overview Homework Coursework Exams Complete 1.4 Notes Computer Science 20/02/2017 Vectors Maths 09/02/2017 Page 1 of 1	Sort by time oldest: The tasks changed order so that the task due soonest will appear last.

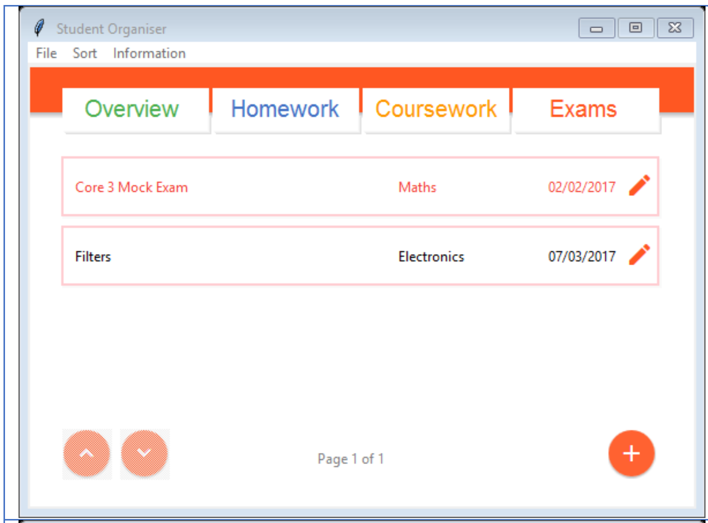
 Student Organiser File Sort Information Overview Homework Coursework Exams Vectors Maths 09/02/2017 Complete 1.4 Notes Computer Science 20/02/2017 Page 1 of 1	Sort by time of task added: The tasks changed order so that the task that I had added first will appear soonest.
 Student Organiser File Sort Information Overview Homework Coursework Exams Complete 1.4 Notes Computer Science 20/02/2017 Vectors Maths 09/02/2017 Page 1 of 1	Sort by task A-Z: The tasks changed order so that the tasks appear in ascending alphabetical order by their name.
 Student Organiser File Sort Information Overview Homework Coursework Exams Vectors Maths 09/02/2017 Complete 1.4 Notes Computer Science 20/02/2017 Page 1 of 1	Sort by task Z-A: The tasks changed order so that the tasks appear in descending alphabetical order by their name.

 Student Organiser File Sort Information Overview Homework Coursework Exams Complete 1.4 Notes Computer Science 20/02/2017 Vectors Maths 09/02/2017 	Sort by subject A-Z: The tasks changed order so that the tasks appear in ascending alphabetical order by their subject.
 Student Organiser File Sort Information Overview Homework Coursework Exams Vectors Maths 09/02/2017 Complete 1.4 Notes Computer Science 20/02/2017 	Sort by subject Z-A: The tasks changed order so that the tasks appear in descending alphabetical order by their subject.
 Student Organiser File Sort Information Overview Homework Coursework Exams Core Practical 12 Physics 20/02/2017 Project Hand In Computer Science 01/03/2017 	Upon loading the coursework tab, each task is sorted by time due soonest by default.

 Student Organiser File Sort Information Overview Homework Coursework Exams Project Hand In — Computer Science — 01/03/2017 Core Practical 12 — Physics — 20/02/2017 Page 1 of 1	Sort by time oldest: The tasks changed order so that the task due soonest will appear last.
 Student Organiser File Sort Information Overview Homework Coursework Exams Core Practical 12 — Physics — 20/02/2017 Project Hand In — Computer Science — 01/03/2017 Page 1 of 1	Sort by time of task added: The tasks changed order so that the task that I had added first will appear soonest.
 Student Organiser File Sort Information Overview Homework Coursework Exams Core Practical 12 — Physics — 20/02/2017 Project Hand In — Computer Science — 01/03/2017 Page 1 of 1	Sort by task A-Z: The tasks changed order so that the tasks appear in ascending alphabetical order by their name.

 Student Organiser File Sort Information Overview Homework Coursework Exams Page 1 of 1 Project Hand In Computer Science 01/03/2017 Core Practical 12 Physics 20/02/2017	Sort by task Z-A: The tasks changed order so that the tasks appear in descending alphabetical order by their name.
 Student Organiser File Sort Information Overview Homework Coursework Exams Page 1 of 1 Project Hand In Computer Science 01/03/2017 Core Practical 12 Physics 20/02/2017	Sort by subject A-Z: The tasks changed order so that the tasks appear in ascending alphabetical order by their subject.
 Student Organiser File Sort Information Overview Homework Coursework Exams Page 1 of 1 Core Practical 12 Physics 20/02/2017 Project Hand In Computer Science 01/03/2017	Sort by subject Z-A: The tasks changed order so that the tasks appear in descending alphabetical order by their subject.

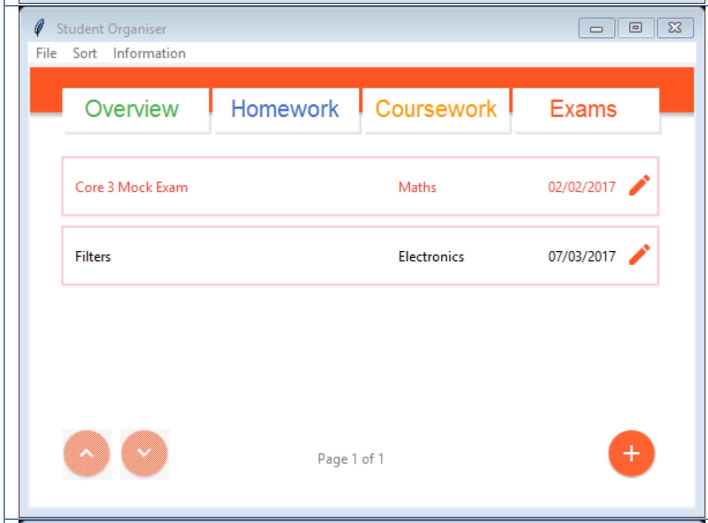
 Student Organiser File Sort Information Overview Homework Coursework Exams Core 3 Mock Exam Maths 02/02/2017 Page 1 of 1	Upon loading the exams tab, each task is sorted by time due soonest by default. There is only a single task present on this page. For demonstration purposes, I will add the following task: Filters, Electronics, 07/03/2017, Exam
 Student Organiser File Sort Information Overview Homework Coursework Exams Core 3 Mock Exam Maths 02/02/2017 Filters Electronics 07/03/2017 Page 1 of 1	Now that there are two tasks present, I can now evidence the result of each sorting method on the exams page.
 Student Organiser File Sort Information Overview Homework Coursework Exams Filters Electronics 07/03/2017 Core 3 Mock Exam Maths 02/02/2017 Page 1 of 1	Sort by time oldest: The tasks changed order so that the task due soonest will appear last.



Core 3 Mock Exam	Maths	02/02/2017
Filters	Electronics	07/03/2017

Sort by time of task added:

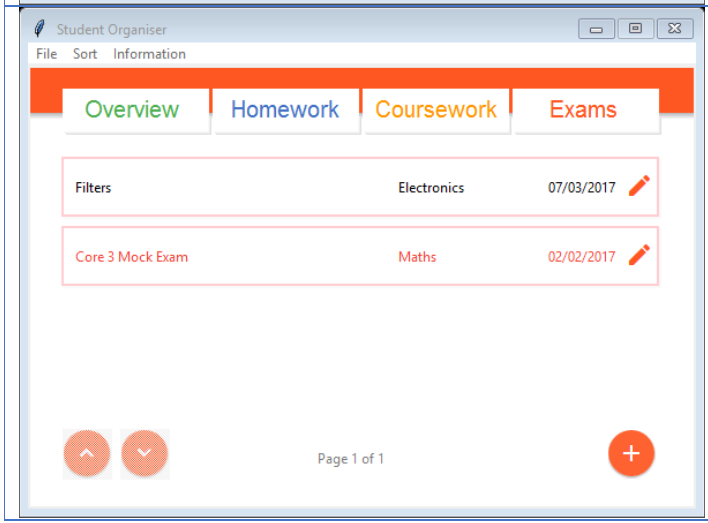
The tasks changed order so that the task that I had added first will appear soonest.



Core 3 Mock Exam	Maths	02/02/2017
Filters	Electronics	07/03/2017

Sort by task A-Z:

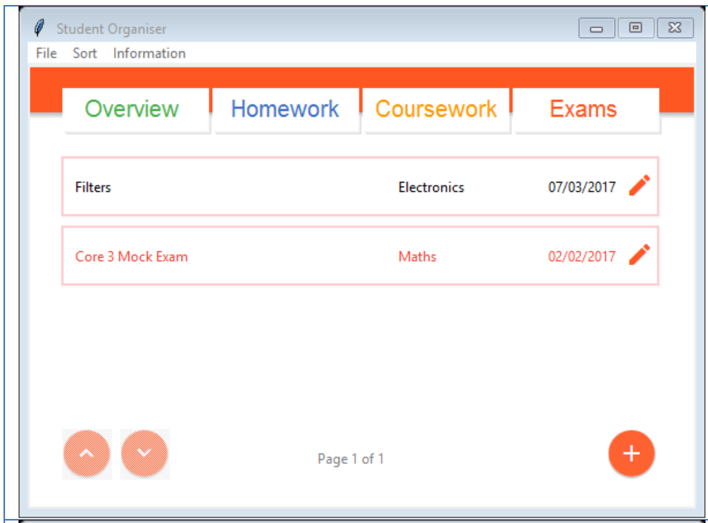
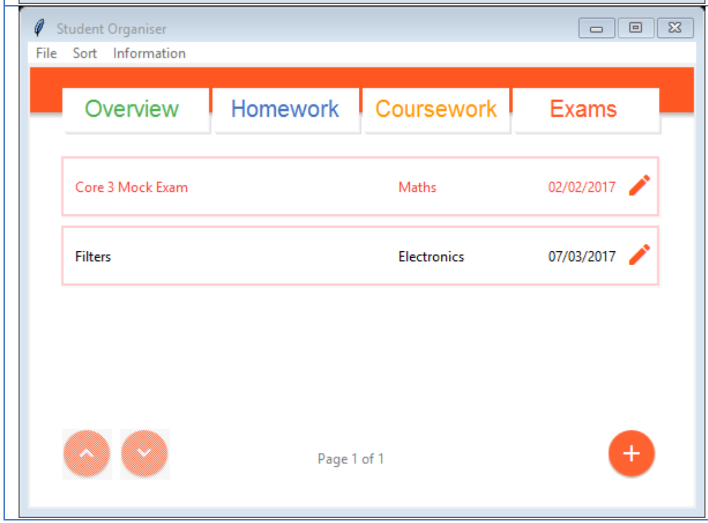
The tasks changed order so that the tasks appear in ascending alphabetical order by their name.



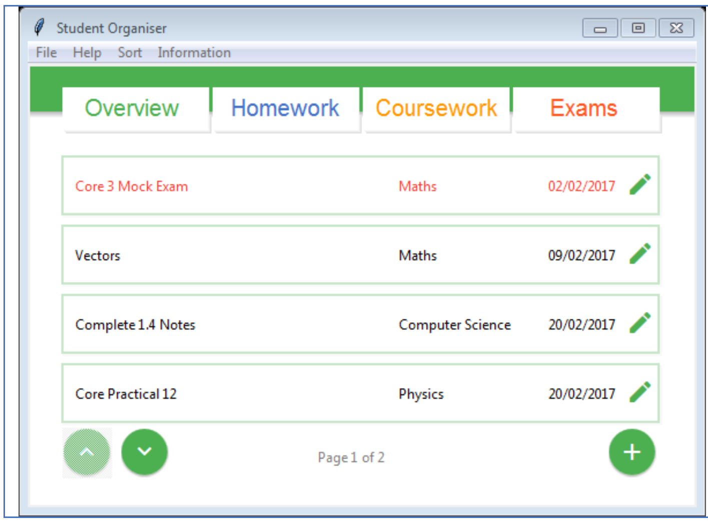
Filters	Electronics	07/03/2017
Core 3 Mock Exam	Maths	02/02/2017

Sort by task Z-A:

The tasks changed order so that the tasks appear in descending alphabetical order by their name.

 Student Organiser File Sort Information Overview Homework Coursework Exams Filters Electronics 07/03/2017 Core 3 Mock Exam Maths 02/02/2017 Page 1 of 1	Sort by subject A-Z: The tasks changed order so that the tasks appear in ascending alphabetical order by their subject.
 Student Organiser File Sort Information Overview Homework Coursework Exams Core 3 Mock Exam Maths 02/02/2017 Filters Electronics 07/03/2017 Page 1 of 1	Sort by subject Z-A: The tasks changed order so that the tasks appear in descending alphabetical order by their subject.

Overdue Tasks:



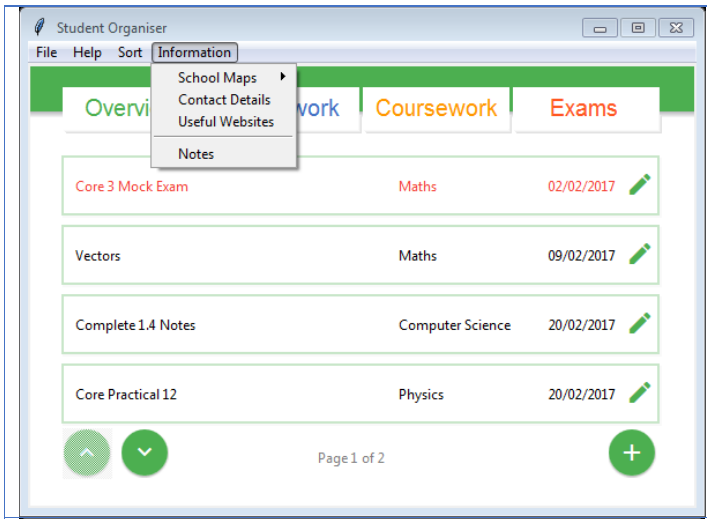
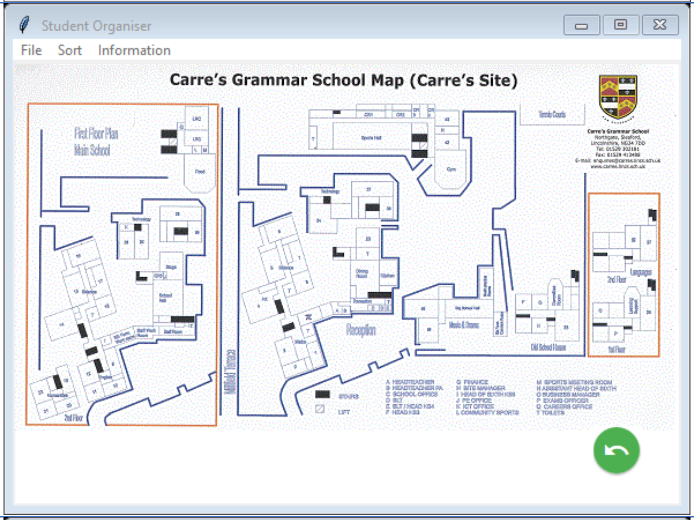
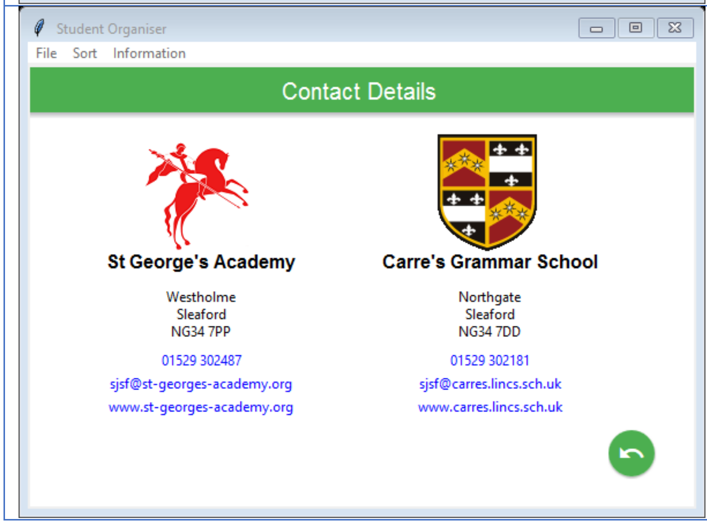
Core 3 Mock Exam	Maths	02/02/2017
Vectors	Maths	09/02/2017
Complete 1.4 Notes	Computer Science	20/02/2017
Core Practical 12	Physics	20/02/2017

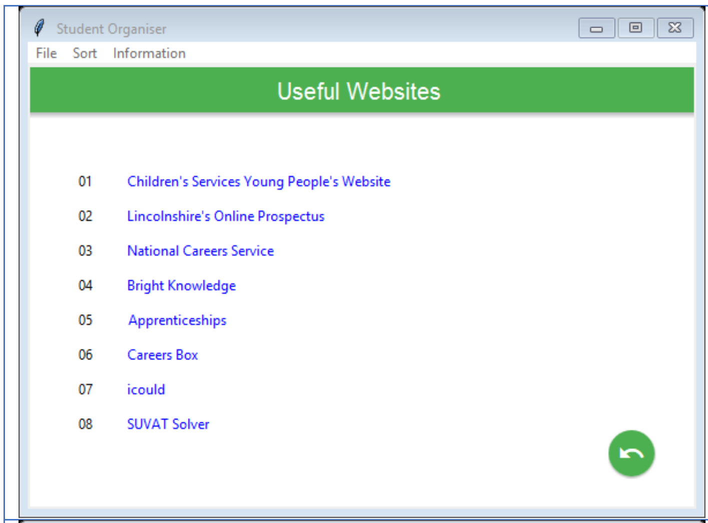
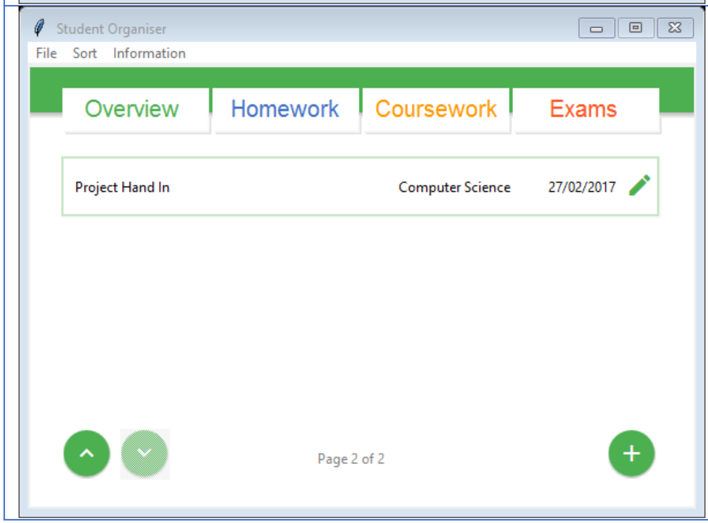
Page 1 of 2

Tasks that are overdue should have red text.

The current date is 08/02/2017, so the first task in the page is considered overdue, and therefore is red.

Databank:

 Student Organiser File Help Sort Information School Maps Contact Details Useful Websites Notes Overvi work Coursework Exams Core 3 Mock Exam — Maths — 02/02/2017 Vectors — Maths — 09/02/2017 Complete 1.4 Notes — Computer Science — 20/02/2017 Core Practical 12 — Physics — 20/02/2017 Page 1 of 2	In the dropdown menu, the user can select the 'Information' button to view data about their school. The list includes: • A school map • Contact details • Useful websites
 Student Organiser File Sort Information Carre's Grammar School Map (Carre's Site) First Floor Plan Main School Reception	If the user selects the school map, they will be given a choice of the two schools. Upon selecting a school, a map will be displayed in the main window as shown to the left.
 Contact Details **St George's Academy** Westholme Sleaford NG34 7PP 01529 302487 sjsf@st-georges-academy.org www.st-georges-academy.org **Carre's Grammar School** Northgate Sleaford NG34 7DD 01529 302181 sjsf@carres.lincs.sch.uk www.carres.lincs.sch.uk	If the user selects contact details, two columns will be displayed, one per school. The information given is the school name, their logo, a phone number, email address, the website and the address. The phone number, email and website can be clicked on to launch their respective clients (the website will open their browser, for example).

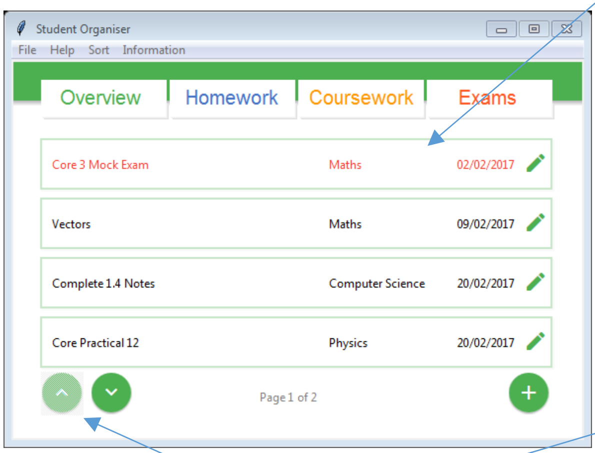
 Student Organiser File Sort Information Useful Websites 01 Children's Services Young People's Website 02 Lincolnshire's Online Prospectus 03 National Careers Service 04 Bright Knowledge 05 Apprenticeships 06 Careers Box 07 icould 08 SUVAT Solver	The final option is the list of useful websites. By clicking on a link, the default browser will open a new tab with the corresponding URL.
 Student Organiser File Sort Information Overview Homework Coursework Exams Project Hand In Computer Science 27/02/2017 Page 2 of 2	The back button will load tasks page for their current tab and page number.

Usability Testing:

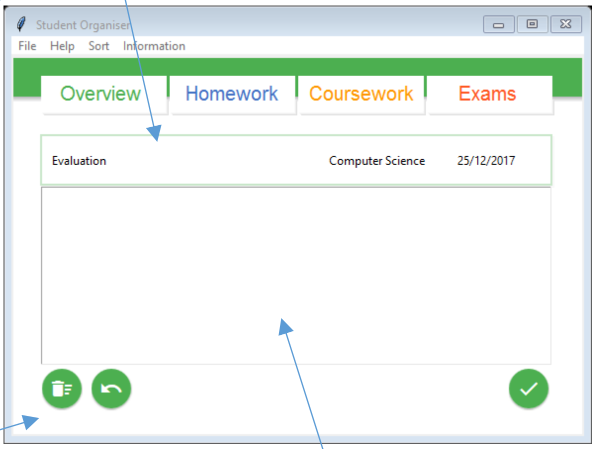
Throughout my previous tests, evidence of usability features can be found. The theme is consistent, allowing the user to easily understand the structure of the program as a whole. All pages only contain the essential objects to form the page leaving a clutter free design for ease of use. The navigation system I used where every button used follows the same theme. Each button matches its surrounding colours and they all use white, easy to understand icons to highlight their purpose. Where in use, there are only three main buttons positioned at the bottom of the page. These buttons are positioned in the same configuration with two to the left edge and a single button placed on the right-hand side. They are all placed at the same vertical level, where their horizontal position varies with three set positions.

The colours have been chosen carefully to complement each other. The background colour is always white so that the page content is always very easy to see. An exception to this can be found on the add task page where two lightly shaded grey rectangles were used to structure the page.

I have used the same style to display the task's data on the edit page. Allows for easier understanding as the data is presented the same way every time.



Core 3 Mock Exam	Maths	02/02/2017
Vectors	Maths	09/02/2017
Complete 1.4 Notes	Computer Science	20/02/2017
Core Practical 12	Physics	20/02/2017

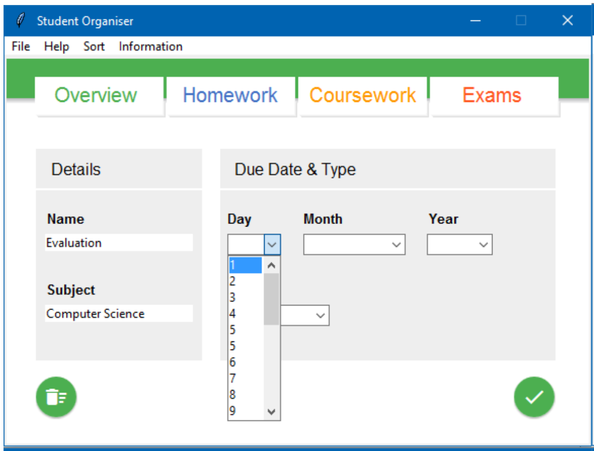


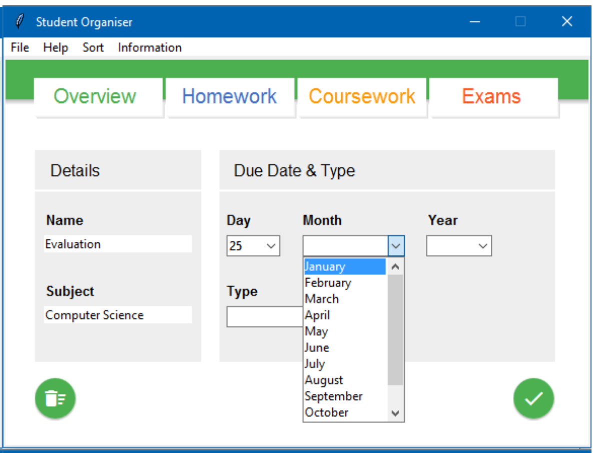
Evaluation	Computer Science	25/12/2017

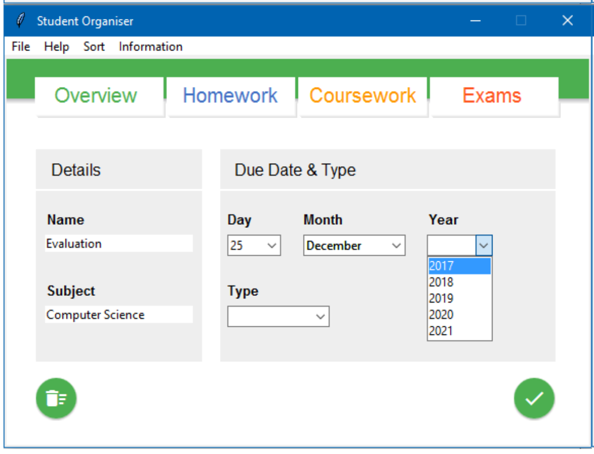
Each page contains buttons that follow the theme of the tab. The buttons are identical in colour, positioning, layout and size.

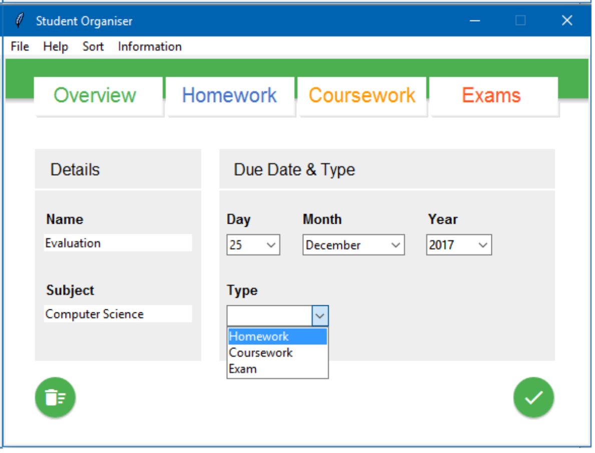
As per the user group feedback, I added a text area for the student to store additional notes for any of their tasks. I filled the remaining space available in the widow so that the notes can be easily seen without the need for constant scrolling to view the notes. This improves the ease of use of this pages.

The below screenshots show the usability features that I have implemented for the user's input. I have used the most appropriate input methods for each piece of data required to store a task to improve the ease of use. I used dropdown menus for the date and type as there can only be a set list of options available. For the day, I chose to list every number from 1 to 31 so that the user can quickly select the correct number. The month drop down menu lists every month in chronological order in terms of their name as this is easier to understand when selecting the month. The year will display a four-digit integer with only the current and next four years available. By reducing the number of options available, reduces the size of the menu and means that the user will not need to search for the correct year from a large list. The task type will only show 3 options as there are only 3 tabs with a unique set of tasks


Student Organiser
File Help Sort Information
Overview Homework Coursework Exams
Details
Due Date & Type
Name: Evaluation
Subject: Computer Science
Day: 1 2 3 4 5 5 6 7 8 9
Month:
Year:


Student Organiser
File Help Sort Information
Overview Homework Coursework Exams
Details
Due Date & Type
Name: Evaluation
Subject: Computer Science
Day: 25
Month: January February March April May June July August September October
Year:
Type:

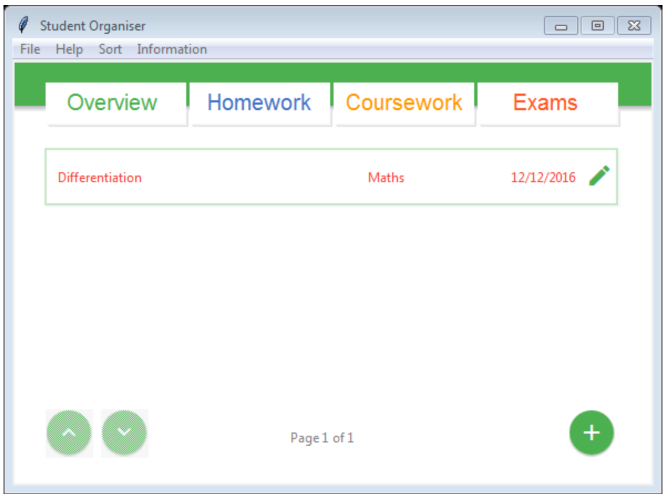

Student Organiser
File Help Sort Information
Overview Homework Coursework Exams
Details
Due Date & Type
Name: Evaluation
Subject: Computer Science
Day: 25
Month: December
Year: 2017 2018 2019 2020 2021
Type:


Student Organiser
File Help Sort Information
Overview Homework Coursework Exams
Details
Due Date & Type
Name: Evaluation
Subject: Computer Science
Day: 25
Month: December
Year: 2017
Type: Homework Coursework Exam

Justification with Evidence:

Clean, Simple GUI

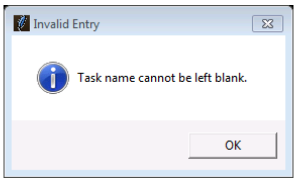
As this is an opinion based test, I asked this question in the student feedback form where all participants involved agreed with this statement. This is a success because the program is designed for these students and the criteria stated it must be clean and easy to use because of this. Since every student agreed that the GUI was well designed and was simple and easy to use, this is very successful.



Differentiation Maths 12/12/2016

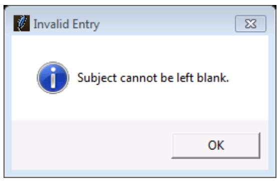
Input Validation:

All of my inputs required input validation to ensure that the program runs smoothly, without any errors. As well as checking if an input is present, it checked if dates were legitimate dates or if they were in the correct format too.


Invalid Entry

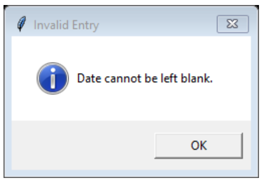
Task name cannot be left blank.

OK


Invalid Entry

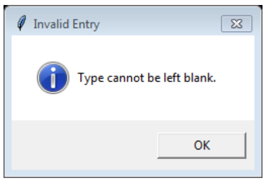
Subject cannot be left blank.

OK


Invalid Entry

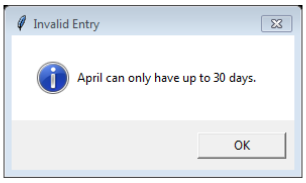
Date cannot be left blank.

OK


Invalid Entry

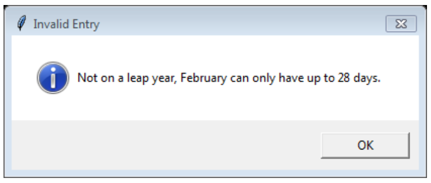
Type cannot be left blank.

OK


Invalid Entry

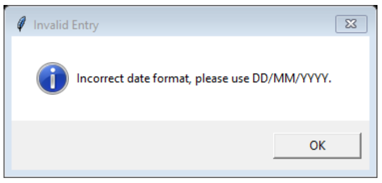
April can only have up to 30 days.

OK


Invalid Entry

Not on a leap year, February can only have up to 28 days.

OK

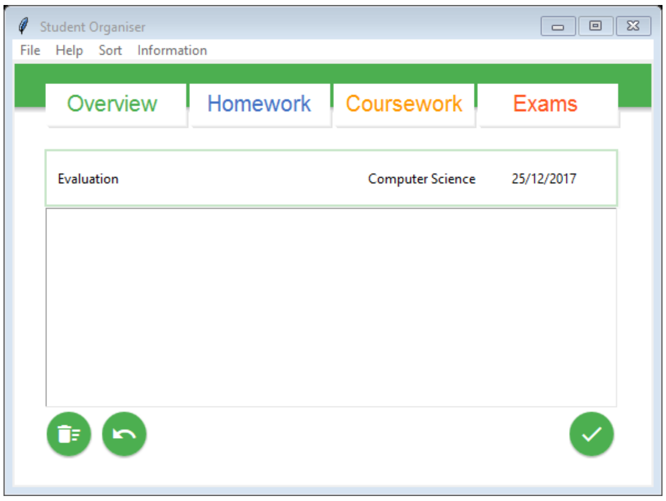

Invalid Entry

Incorrect date format, please use DD/MM/YYYY.

OK

Appropriate Input Methods

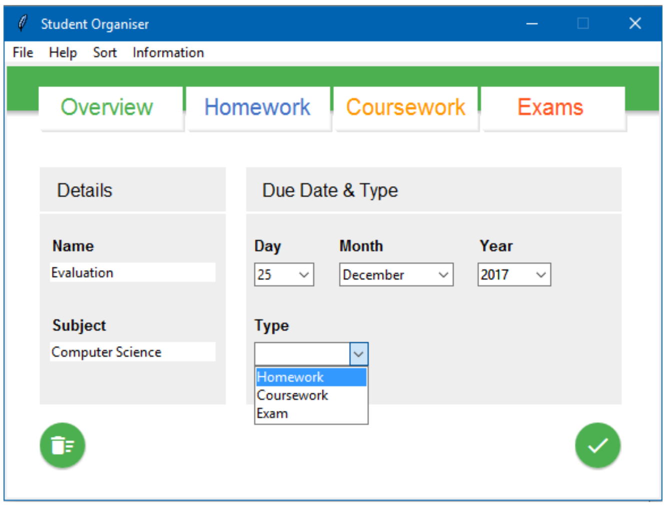
For any short piece of text to be entered, such as the task title or subject, I had used text entry boxes where I had adjusted their width to suite the purpose. This can be seen in the modify task page where the task name, subject and date have been set to specific widths so that they all fit in a single line where there is plenty of room to enter the details. Also on the modify tasks page, there is a notes section below. I had chosen to use a text area with a scrolling side bar because the user may need to use many lines to write notes about a task. The scrolling side bar would allow the user to navigate through their notes easily. Because of these reasons, the input methods that I have chosen here are relevant to the type of data required and improves ease of use.



Evaluation Computer Science 25/12/2017

On the insert tasks page, I had chosen to use four dropdown menus for the date and task type. For the day, I had chosen to list every number from 1 to 31. These are the only values required and will not allow the user to insert an invalid day value. For the month, I had listed the month's name for all twelve months so that it appears easier to read and work with. It prevents the user from having to convert the month's name to the number before selecting a month. The last part of the due date was another drop down menu for the year. I had listed the current year followed by the following four. This simplifies the options that the user can select as it removes any years that the user will not need to use at the current time of insertion.

The task type can only be one of three options and therefore it would be very appropriate to use a drop down menu to list them. Not only does it remove any chance of an invalid entry, it improves the functionality as the user will not need to write their own in or include a spelling mistake.



Name: Evaluation

Subject: Computer Science

Day: 25

Month: December

Year: 2017

Type:

Homework
Coursework
Exam

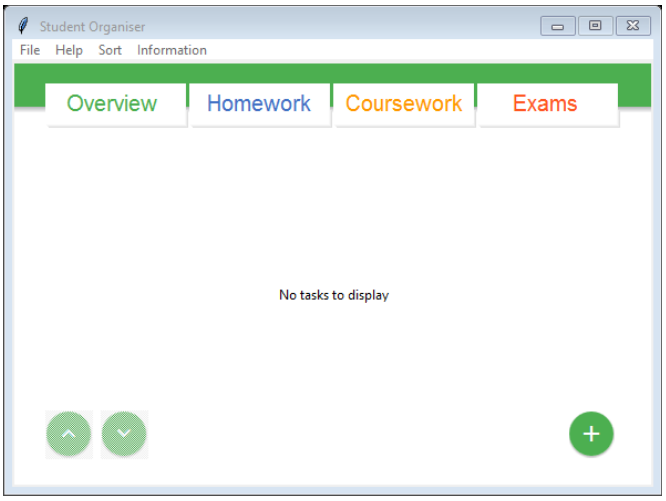
Clearly Labelled Buttons

Throughout the program, I have used a variety of buttons so that the user can access additional functions. I have taken care in choosing the design for these so that they were as easy to interpret as possible. I have used Google's Material Design theme with these floating action buttons. I selected four Material Design colours from their pallet for the background colour, before adding an icon in the foreground. The icons are simplistic and maintain a constant theme where they are white with a transparent background. I chose the icons that I thought would be best suited for the function that it controls.

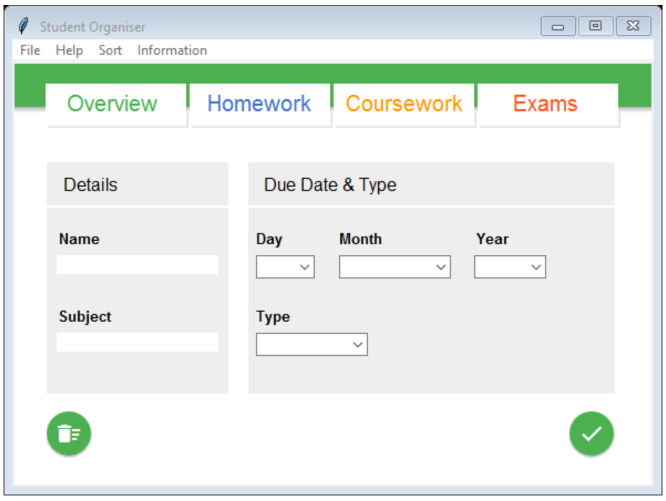
Below is a table of the icons that I have selected for each function.

Icon	Function
Bin	Delete
Up Arrow	Page Up
Down Arrow	Page Down
Plus	Add Task
Curved Left Arrow	Return

Below are two screenshots from two pages within my program. At the bottom of each page, there are a total of five buttins with the design which I have previously described. They clearly show the purpose that they serve, especially when given the page that they have been placed on.

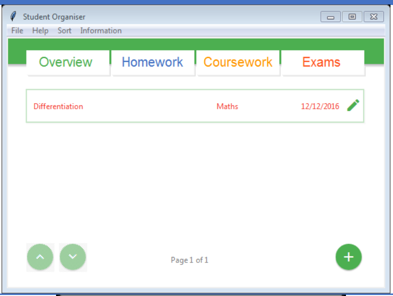
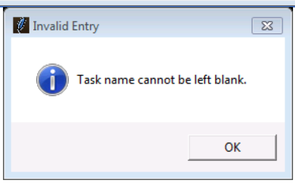
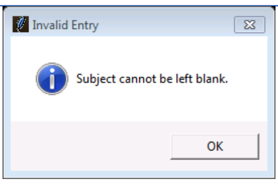
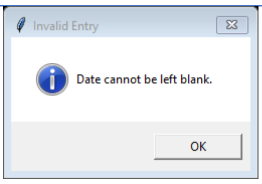


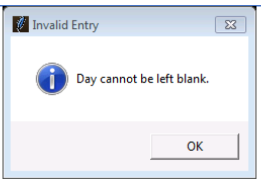
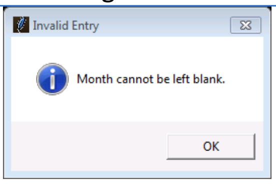
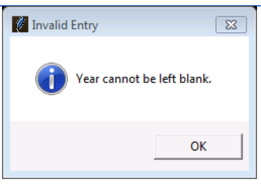
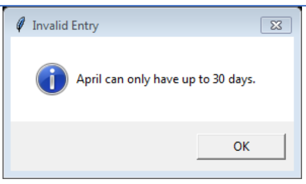
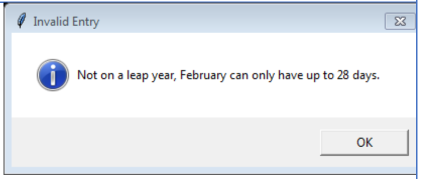
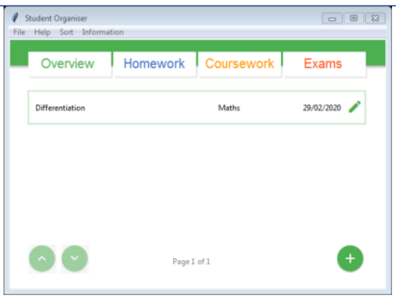
No tasks to display

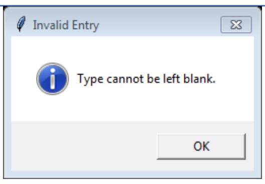
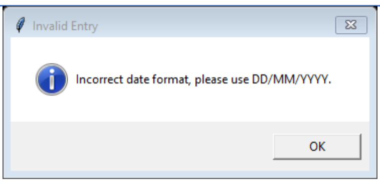
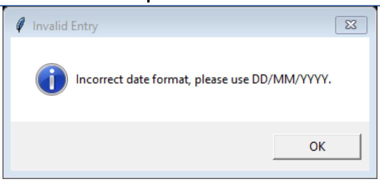
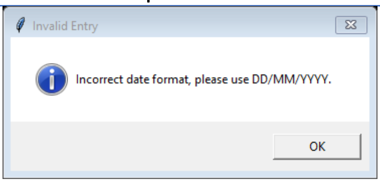
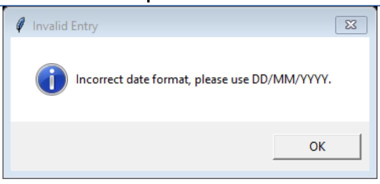


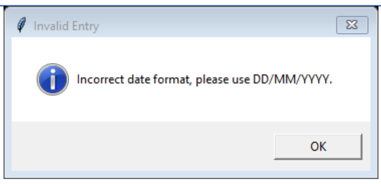
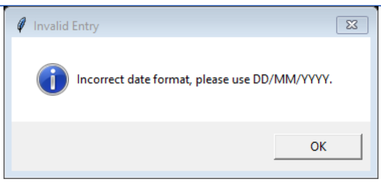
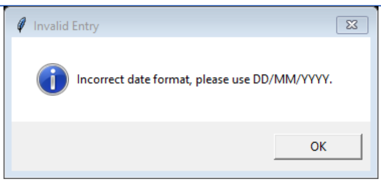
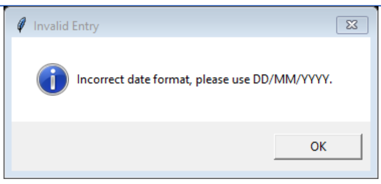
Testing Robustness:

I will perform some additional tests to verify the integrity of the solution. I will attempt to break the program with these tests as a failure would result in evidence supporting that the program is not robust. I will enter invalid inputs for every input method available in the program. Input methods are where data can be entered by the user and is then interpreted. If the data is invalid, it is not expected, errors could be caused.

Test	Test Data	Justification	Actual Result / Evidence
Add a task with valid details.	Differentiation Maths 12/12/2016 Homework	Any tasks with valid details must be accepted by the program. If they are not, the program is not functioning correctly. It must store any valid details given by the student as all tasks will be different.	 Student Organiser File Help Sort Information Overview Homework Coursework Exams Page 1 of 1 Differentiation Maths 12/12/2016
Add a task without a name.	Maths 12/12/2016 Homework	An empty filed should not be allowed as this would cause potential errors when reading the data later on. A message box should be displayed to the user with the message, "Task name cannot be left blank."	 Task name cannot be left blank. The message was presented to the user which shows that the validation had worked correctly, maintaining the integrity of the solution.
Add a task without a subject.	Differentiation 12/12/2016 Homework	This is a required entry and leaving it null would result in errors when reading the file or when sorting the tasks by subject. This should be prevented and the following message should be displayed to the user, "Subject cannot be left blank."	 Subject cannot be left blank. The message box had been displayed and the input was not accepted.
Add a task without a date.	Differentiation Maths Homework	The due date is a key piece of data for the functioning of the program. By default, all tasks are sorted by their due dates. By not selecting a due date, the program will not load the tasks correctly. If this rejects the given date correctly, a message box should appear with the text, "Date cannot be left blank."	 Date cannot be left blank. The message box was presented to the user and the input ws not accepted. The test had passed.

Add a task without a day selected.	Differentiation Maths /12/2016 Homework	The date cannot be submitted if pieces of the date are not present. If the day is null, a message box should appear with the text, “Day cannot be left blank.”	 Invalid Entry Day cannot be left blank. OK The task was not saved and the output message was correct.
Add a task without a month selected.	Differentiation Maths 12/ /2016 Homework	The date cannot be submitted if pieces of the date are not present. If the month is null, a message box should appear with the text, “Month cannot be left blank.”	 Invalid Entry Month cannot be left blank. OK The task was not saved and the output message was correct.
Add a task without a year selected.	Differentiation Maths 12/12/ Homework	The date cannot be submitted if pieces of the date are not present. If the month is null, a message box should appear with the text, “Year cannot be left blank.”	 Invalid Entry Year cannot be left blank. OK The task was not saved and the output message was correct.
Insert a task with an invalid date.	Differentiation Maths 31/04/2017 Homework	This will verify that the program shouldn’t allow dates where the day selected is greater than the maximum number of days in the month.	 Invalid Entry April can only have up to 30 days. OK It had stated that the month selected, April, can only have 30 days. This has succesfully validated the input and informed the user of the error.
Insert a task with an invalid date.	Differentiation Maths 29/02/2017 Homework	This will verify that the program shouldn’t allow a day value greater than 28 for February while it is not a leap year.	 Invalid Entry Not on a leap year, February can only have up to 28 days. OK It had stated that not on a leap year, February, can only have 28 days. This has succesfully validated the input and informed the user of the error.
Insert a task with an invalid date.	Differentiation Maths 29/02/2016 Homework	This will verify that the program should allow a day value greater of 29 or less for February on a leap year.	 Student Organiser File Help Sort Information Overview Homework Coursework Exams Differentiation Maths 29/02/2020 Page 1 of 1 It had allowed this date as it is valid. The changes were made to the task and it was saved.

Insert a task without a task type selected.	Differentiation Maths 25/04/2017	The test here will check that the program will not save a task if the type has not been selected. This is an important test as all tasks must be assigned a type. This is because the tabs along the top of the program relies on this field to correctly organise the user's tasks.	 Invalid Entry Type cannot be left blank. OK It had stated that the task type cannot be left blank so the user will now know where the mistake in this form is. This guides the user to correct the error so that the task can be saved and the incorrect entries are resolved.
Modify the third character of the date.	Differentiation Maths 06c01/2017 Homework	This will test the validation for the date entry box on the modify page. The format of the due date is strictly DD/MM/YYYY and any inaccurate entries must be discarded.	 Invalid Entry Incorrect date format, please use DD/MM/YYYY. OK The user has been alerted of the date format to use and the date was not accepted.
Modify the sixth character of the date.	Differentiation Maths 06/01@2017 Homework	This test will enter a date in an incorrect format to prompt the program for an incorrect format message.	 Invalid Entry Incorrect date format, please use DD/MM/YYYY. OK The user has been alerted of the date format to use and the date was not accepted.
Modify the third and sixth character of the date.	Differentiation Maths 06#01{2017 Homework	This test will enter a date in an incorrect format to prompt the program for an incorrect format message.	 Invalid Entry Incorrect date format, please use DD/MM/YYYY. OK The user has been alerted of the date format to use and the date was not accepted.
Modify the date to greater than 10 characters in length.	Differentiation Maths 06/01/20017 Homework	This test will enter a date in an incorrect format to prompt the program for an incorrect format message. The date can only accept 10 characters as this is all that is required. A different value means that the date has been given in an invalid format.	 Invalid Entry Incorrect date format, please use DD/MM/YYYY. OK The user has been alerted of the date format to use and the date was not accepted.

Modify the date to fewer than 10 characters in length.	Differentiation Maths 6/01/2017 Homework	This test will enter a date in an incorrect format to prompt the program for an incorrect format message. The date can only accept 10 characters as this is all that is required. A different value means that the date has been given in an invalid format.	 Invalid Entry Incorrect date format, please use DD/MM/YYYY. OK The user has been alerted of the date format to use and the date was not accepted.
Modify the date to include characters as the day value.	Differentiation Maths c6/01/2017 Homework	This test will enter a date in an incorrect format to prompt the program for an incorrect format message. The date can consist of numbers for the three parts. Any additional characters will be classed as an invalid entry where an invalid format has been given.	 Invalid Entry Incorrect date format, please use DD/MM/YYYY. OK The user has been alerted of the date format to use and the date was not accepted.
Modify the date to include characters as the month value.	Differentiation Maths 16/Kl/2017 Homework	This test will enter a date in an incorrect format to prompt the program for an incorrect format message. The date can consist of numbers for the three parts. Any additional characters will be classed as an invalid entry where an invalid format has been given.	 Invalid Entry Incorrect date format, please use DD/MM/YYYY. OK The user has been alerted of the date format to use and the date was not accepted.
Modify the date to include characters as the year value.	Differentiation Maths 16/01/2ol7 Homework	This test will enter a date in an incorrect format to prompt the program for an incorrect format message. The date can consist of numbers for the three parts. Any additional characters will be classed as an invalid entry where an invalid format has been given.	 Invalid Entry Incorrect date format, please use DD/MM/YYYY. OK The user has been alerted of the date format to use and the date was not accepted.

Overall, every test passed without failure. Therefore, I can conclude that this program is very robust in design. All invalid inputs were rejected when tested to ensure that all variables will be expected in terms of content, length, format and type. Where an error message occurred, the user could read the information given to learn what the issue was with their entry. This is a useful feature as the user knows exactly where the issue is and con resolve it promptly to save the task.

3.4.2 - Success of the Solution:

The solution that I have created has worked very well.

The process of adding a task to this program functions as intended without fault. The validation on the task name will display relevant dialogue boxes if the user does not enter a name. This is the same with the subject, whereby there must be data present in order to pass the validation. Each of these tests and their respective results can be found after developing the notes page from my client's feedback. Each test consisted of a screenshot representing the result. These screenshots are the dialogue boxes that are presented to the user if an invalid input is detected. I deliberately attempted a variety of invalid inputs that were all found and the appropriate message was given. The due date for the task is validated in multiple stages. Whilst adding a task, the day and month is given as an integer value whilst the month can be selected from a dropdown menu of strings. The month is converted to its integer format where before checking if the day selected is within the correct range for the selected month. If not, a message is displayed so that the user is aware of the issue and allows the user to correct this. If the day and month are valid, it should then check if the year is a leap year. This will only affect dates where the user has selected February as the month. The results for these tests can be found just after the other two validation checks. There are a series of tests that I had performed to ensure that the date validation works correctly as there are many aspects to the date that can be altered to give an invalid entry. I attempted every method for invalid date entries from, trying an invalid date to using the incorrect format. If the date is valid, it will continue to validate the next piece of data, else it will alert the user of the maximum number of days available on February for the year selected. Finally, if the tasks type has not been selected on any page, the user is also alerted. All message boxes that show have an 'OK' button which will dismiss the pop up. Once dismissed, the program will continue to show the add task page with the data that they've entered so that they can easily resurrect any issues without the need to restart this process. The task will only be entered as a record in the database if the data validation checks have all passed. This aspect of my program is a key function to the core of the program whereby the user records their tasks for future reference. Without this, sorting tasks may become problematic as SQL errors would arise if there were any invalid dates present. As well as this, if there were any fields mistakenly left empty by the user, remembering what the task data should be at a later date could disrupt their organisation, the main problem that this is a solution for.

These validation methods also apply and function perfectly when modifying the tasks. On the tasks page, the user has the option to modify a task by simply clicking the crayon icon by the task which they'd like to edit. This will open a page with their selected task at the top in the same format, with the addition of a large text area positioned below. The task name and subject are validated with the same function and therefore if they are empty upon saving the task, the user will be alerted with a pop up message. The date is also validated with the same rules and are all applied correctly. The task type cannot be changed and therefore does not require additional validation. The text area below that is designated for the user's notes on the task also require no validation and therefore can be left empty without causing any issues with the saving process. The notes area was incorporated as a result of stake holder feedback. Testing had been performed to ensure that the notes saved to specific tasks and these can be found in the testing chapter. These tests confirmed that the user has the ability to write additional information for their tasks and save them without any issues. I had implemented usability features within this page as the user can easily see what this page is designed to do. There is a clean, set layout following on from the previous page. The large text area allows quick access to the tasks notes and there are only three buttons on this page with clearly marked icons defining their functions.

Each sorting method rely on using SQL statements when querying the database table. The tasks can be sorted by the name, subject and date in any order form ascending to descending or vice versa.

There is also the option to sort by task type using the tabs along the top of the program and then also apply any additional sorting to this set of data. During the testing of this data with sample tasks, they all sorted and represented the data correctly. Each of these sorting options had been thoroughly tested with every tab confirming that my success criteria regarding the use of sorting methods has been fulfilled to its fullest extent.

Tasks which were overdue were highlighted in red text. Again, in my testing chapter, the results to this can be found through the tests. Not only did it appear to function correctly in the testing for overdue tasks, it also responded appropriately in every other test scenario where overdue tasks can be seen in red text. Again, this fulfils another success criteria.

The school maps load perfectly where a back button will return the user to the previous tasks page, whichever the user had selected. The contact details would open up any relevant applications when the links were clicked. The phone number will open any telephone applications, the email address would open a mail client with the ‘to’ field filled in with the correct details and the website will open a web browser with the website. This works similarly to the useful websites page. Each website within this list will all launch the web browser or open a new tab if the user’s default web browser is currently open.

In review, the program functions as expected and all pages load from one page to another. All validation methods that I have written work perfectly and always alert the user of any issues, allowing the user to modify their inputs without losing any progress made. All buttons are loaded in the correct colour depending on the tab that the user had secreted and link to the pages as designed.

Usability features were at the core of this solution. Some of my success criteria involved usability features as their main aim. The main goal with this program was to create an electronic planner to aid the organisation of the student. In order for this to be an effective solution, the user must be able to use this software well. I created a clean theme with no bloat with hidden or complicated menus. Every button that I had placed on the main activity followed the same material design theme. All buttons matched in colour and size and relative in position. I chose the most relevant icon I could to highlight the function of the button. Usability features can be found in the page sorting algorithms too. Where the up or down button would be disabled as appropriate so that the user can clearly see that scrolling is not available in that direction. On the tasks page, only essential information is displayed per task maintaining the clean theme. To view the additional task info, an icon beside the task can be used to show all details available for that task, including the notes area. In addition, the sorting methods that I had included were placed within the menu bar. This removes them from the main page so that they are not in the user’s main focus. By using the menu bar, I could create a dropdown list of these sorting methods and group them too, adding to the organisation of the program.

There are still features that could be added to this program to improve its role as a student organiser. For future development, I could make the program dynamically adjust its contents with a change in window size. This would make it easier for the student to view more of their tasks at any given time by increasing the size of the window. Another feature that is lacking is the ability to set event reminders, where a notification could appear at a set time before a task is due. This was a feature used in the other solutions that I had researched. Other than the software improvements, more planner material could be integrated to give the user access to more useful data such as a school/personal calendar, conversion tools and revision guides, for example.

There were criteria that had not been met. This criterion set the objective that the program should be able to delete tasks automatically. My initial idea for this was to add the option for the user to set an auto delete feature to remove any tasks from the database based on the duration since they had become overdue. This feature would automatically de-clutter any tasks that are no longer needed so that the user can focus on the current tasks. For further development on this project, I would add a settings page where this feature would reside and the user could adjust the time in terms of days for the rule.

The final specification that I had stated in the measurable success criteria states the following. “The software must be presented clearly so that it is easy to use. It should be presented in a logical manner where widgets are placed with care and appropriately. The design should be aesthetically pleasing.” Due to the nature of this, I cannot show success nor can I prove any failure of this point through testing myself. The success of this point relies on the opinion of the user and therefore I have decided to create feedback forms. These forms will allow me to question the user group to receive feedback on the program.

I can use these forms to assert that the final point has been met. To do this, I can ask multiple opinionative questions regarding the design and the user’s thoughts of the software. Using these results, I could confirm if the points have been met by taking the result that the majority of user’s opted for.

Student 1:

Statement	Agree	Neutral	Disagree	Comments
The program as a whole worked as expected.	✓			
Specification				
Tasks should be easy to create with the following data: name, subject, due date and type	✓			Yes, the different forms were clearly labelled and all require valid inputs.
It should be simple to make any changes to the tasks.	✓			Any of the information can be edited quickly.
The user should be able to view and edit notes for individual tasks.	✓			
The tasks should highlight red in text if overdue.	✓			Yes, although this took me a day to test.
The user should be able to sort tasks by their type.	✓			
Information				
School maps were clear.			✓	They appeared a little granular.
School contact details are displayed in a methodical manner.	✓			Both centres have the useful information listed under their respective badge.
Useful websites could be opened in any web browser.	✓			When clicked, they opened in the default browser.
GUI				
The design is aesthetically pleasing.	✓			
The program flowed in a logical order.	✓			
The user interface was easy to use.	✓			Was slightly odd to edit a task to view its notes.
The layout of pages was logical.	✓			

1. **What aspects of the program could be improved?**

The school map pictures could be made slightly larger to provide us with a sufficient chance to see it.

2. **Were there any features that could be included to improve functionality of this program?**

Having a reminder function so the user can tell the program to remind them a certain amount of time before a task is due. The program can then have an alert pop up when the program starts with the task and when it is due (time remaining).

And maybe have an orange colour for next one or two days.

3. **What worked well?**

The program allowed adding and removing tasks, with different types and they were added in their respective categories. Also, the tasks were highlighted red if they were past their due date which made them noticeable.

Student 2:

Statement	Agree	Neutral	Disagree	Comments
The program as a whole worked as expected.	✓			
Specification				
Tasks should be easy to create with the following data: name, subject, due date and type		✓		The form was clearly presented with small labels. A date picker would be preferable for entering the date.
It should be simple to make any changes to the tasks.	✓			
The user should be able to view and edit notes for individual tasks.	✓			
The tasks should highlight red in text if overdue.	✓			
The user should be able to sort tasks by their type.	✓			
Information				
School maps were clear.		✓		Could be made larger.
School contact details are displayed in a methodical manner.	✓			All the links would open appropriate applications.
Useful websites could be opened in any web browser.	✓			Each link opened a new tab in my default browser.
GUI				
The design is aesthetically pleasing.	✓			
The program flowed in a logical order.	✓			
The user interface was easy to use.		✓		
The layout of pages was logical.	✓			

1. **What aspects of the program could be improved?**

Could use a date picker to select the due date. This would be easier to view legitimate dates before selecting the submit button and receiving a message informing me of the incorrect values. It will also be easier to see a calendar moth with the days labelled.

The school map could be larger or add a method to zoom into sections.

2. **Were there any features that could be included to improve functionality of this program?**

Include a calendar view so that students can see their whole week / month at a time.

3. **What worked well?**

Adding and modifying tasks was easy to do with easy to use input boxes. Where data was required, the entry boxes were clearly labelled and if an incorrect value was entered a message box appeared with the issue.

The tasks sorted correctly when I clicked any of the sort options.

The GUI looked nice.

Student 3:

Statement	Agree	Neutral	Disagree	Comments
The program as a whole worked as expected.	✓			
Specification				
Tasks should be easy to create with the following data: name, subject, due date and type	✓			
It should be simple to make any changes to the tasks.	✓			
The user should be able to view and edit notes for individual tasks.	✓			
The tasks should highlight red in text if overdue.	✓			
The user should be able to sort tasks by their type.	✓			
Information				
School maps were clear.	✓			
School contact details are displayed in a methodical manner.	✓			
Useful websites could be opened in any web browser.	✓			
GUI				
The design is aesthetically pleasing.	✓			
The program flowed in a logical order.	✓			
The user interface was easy to use.	✓			
The layout of pages was logical.	✓			

1. What aspects of the program could be improved?

I would like to be able to resize the window – if I am using the program on a large monitor, it would be nice to be able to use all of the space available.

2. Were there any features that could be included to improve functionality of this program?

I would like to be able to use a smartphone client – this would allow me to view and edit my tasks from anywhere at any time, as I do not always have access to my computer.

3. What worked well?

The program solves the problem of storing a list of tasks very well. It is easy to use, with all of the features being easy to find. I like having control, such as the ability to sort the list however I choose. The user interface is very aesthetically pleasing: the design is enough to make the program stand out but not enough to detract from its usability.

Review:

Overall, the results from these students were very positive. All three students that participated agreed that the program worked as expected.

Students agreed with the majority of specification points. The purpose of including the success criteria as questions was to test how well I have created a solution to a set of objectives. As the overall result was very positive for each specification point, students found that the program implements these very well. One student gave the comment, “the different forms were clearly labelled and all require valid inputs.” Here, the student had commented upon the data validation that my program does when checking the user’s inputs. However, another student would prefer a date picker, rather than the three drop down boxes that I have used.

The information section was also very positive, where only the ‘school maps’ question was answered more neutral than positive. It appears that the students would prefer larger maps of higher quality. I agree with their comments on this as they .gif image that I had used for the two maps did lose quality from the original. As for the size, I could take this feedback for further development of this project to produce a method to zoom in on the map.

The final multiple choice set of questions were regarding the graphical user interface. This is the most important set of questions on this form as I will be collecting the user’s opinions for an objective element of the success criteria. From the results, all students though that the design of the solution was aesthetically pleasing, the layout of the pages were logical and the program flowed in a logical order. This is very positive feedback and supports the conclusion that my final specification in the measurable success criteria has been sufficiently met.

As well as multiple choice questions, I added in three open ended style questions so that I could get user’s opinions and their thoughts for improvements for future development. Extending from their previous comment, a student wrote: “Could use a date picker to select the due date. This would be easier to view legitimate dates before selecting the submit button and receiving a message informing me of the incorrect values. It will also be easier to see a calendar moth with the days labelled.” Currently, to enter a date, the user will need to select three values from a set of dropdown boxes. This system works but there is no way of knowing the day (name) of the date selected unless an external calendar is used. As well as this, the date could be invalid and the user would be required to wait until submission before a message box appears with the details. This is inefficient and a date picker would be a much more suitable alternative. This eliminates any chance of invalid inputs and is much easier to work with as dates are set out in sets of months with the day columns.

The next question asked for any features that could be included to improve functionality of this solution. A student suggested that the user should be able to set a reminder for their tasks where a message box could appear to alert the user. A useful feature to ensure users will not forget about a task and therefore increasing the productivity aspect of the software. Another point was that the tasks will be highlighted in red text if they are overdue and that using an orange text colour for tasks that are upcoming within a set number of days would be useful. A calendar view would be another useful feature so that students can view their whole week or month.

Finally, what worked well? Students commented on the way the program worked, “Adding and modifying tasks was easy to do with easy to use input boxes. Where data was required, the entry boxes were clearly labelled and if an incorrect value was entered a message box appeared with the issue.” Here, this shows that the input validation that I added to ensure smooth running of the program is working as intended and therefore supports that this solution is successful. Additionally, there was very positive mentions of the sorting algorithms: “The program solves the problem of storing a list of tasks very well. It is easy to use, with all of the features being easy to find. I like having control, such as the ability to sort the list however I choose. The user interface is very aesthetically pleasing: the design is enough to make the program stand out but not enough to detract from its usability.”

Overall, the feedback that I received was very successful in that all of my success criteria has been met and a range of suggestions for future development and improvements were given.

3.4.3 - Describe the Final Product:

The finished product is complete with all of the features as outlined in my initial specification. The program can store tasks of type homework, coursework and exam. The user can modify these tasks if necessary and any data entered will be validated using the same rules. A sorting system allows the user to view their tasks based on time, subject and task. This is a useful feature as the student could easily identify what the user needs to focus on and show which tasks are due soon compared to others. Any tasks that have a due date that has expired should be highlighted red so that the user can clearly see which tasks are overdue. Tasks that are no longer required can be deleted from the database easily. As well as the students being able to write their homework, coursework deadlines and exam dates there is an easy access data bank where information relevant to the school can be found. I have included a school map, contact details for the school with links to open external applications and a list of useful website links.

3.4.4 - Maintenance and development:

Maintenance and Limitations:

Some aspects of this project could be amended to suit additional needs of the users. The program is very static in terms of its structure. For example, the theme cannot be changed and a maximum of four tasks are displayed on each page. Some users may wish to adjust these properties and therefore a settings menu would be a suitable addition in the future. Additionally, the window is a fixed width and height. An improvement would be top allow the user to change the window size and update the contents dynamically so that more tasks can be displayed on any given page.

There are a few limitations in this version of the software including the amount of data that can be entered for each task, the types of task available and the length of the data that can be viewed in the tasks page. To maintain this, if the user should require more fields to store data within, the database would need updating and a method for inserting this additional information should be included. As for the amount of characters that can be seen for each piece of data in the tasks page, the size of the entry boxes could be adjusted in length so that more text is visible without the need for scrolling. Another solution would be to allow the user to dynamically adjust the width of the program and scale to contents within.

The program is mostly self-sustaining. The database file stores all of the user's tasks and will grow and shrink in size when required. If there is an issue with the database, for example a table could become corrupted, then the database file can be easily deleted. Upon the next load of this program, a new database file will be created and previous issues should be solved. However, this will case any tasks previously saved to be permanently deleted. If, in the future, the stake holders would like to store more information about a task, then the program as a whole will need updating. Entry widgets and validation methods would need creating so that the user can enter this new data and the database file or table would require more columns to save the additional data.

Future Development – Improvements to Limitations:

This solution works and has met or partially met most of the success criteria that I had created during the planning stage of this project. There are still possible improvements to be made in the form of adding new features or modifying existing features. I will be using the results from the feedback forms to collect ideas for future development. As well as feedback, I will be using my own ideas to ensure that the success criteria could be fully met if this project were to continue in the future.

There was one specification within my success criteria that has not been met. This stated that the program should be able to "automatically delete old tasks." Tasks can be deleted but there is no option to automatically remove them from the database. I could include a settings page where the user can make adjustments to the running of the program. I can include the option to automatically remove old tasks within this page, allowing the user to select the number of days a task should be allowed to remain once expired.

The only other specification that was only partially met was that, "There should be school information available to access (including: a school map, term dates, contact details and website links)." The program has currently an area to access the school map, contact details and useful websites, however there is no inclusion for the term dates. This is a calendar based project and to include this feature, I could create a new page in the 'Information' tab. Alternatively, I could create additional options in the settings page to allow the user to set term dates which will automatically insert into the database.

From the feedback form, there were requests to add a date picker to select a due date as it would improve the functionality. This could be included in a future development where I will remove the old entry methods for the date and replace them. In addition to this, I could include a week or month overview to allow students to view multiple days on a single page. The inclusion of this would result in the program becoming similar to a physical planner or calendar. A result which would improve the functionality as users can easily view when individual tasks are due respective o each other without the need to scroll between pages or look up corresponding dates with days. Another improvement could be to allow times to be included with the due date to assist the calendar view and reminders.

In future development, I could make improvements based on the limitations of the software. The window size is fixed, as is its content. I could redesign the pages so that the window can be resized dynamically and the content adjusts to this parent dimensions. To do so, I would have to increase the complexity of the page sorting algorithm to incorporate the height of the window to allow more or fewer tasks per set. This is because the default size as it is now can comfortably fit four tasks per set (per page). If the height is adjusted so that there is a larger area, there will be space free where tasks could be positioned. I could add another set of options to the settings page to allow the user to add more fields to their tasks. This will require the program to incorporate dynamic horizontal scaling of the window to fit with the one task per row theme.