Student Planner

Design Specification by Isaac Cheng

Contents

Analysis

Problem Analysis

Technology is quickly evolving, and there has been a great push over the past decade or so to use it to increase productivity. This has been particularly noticeable in education in the United States, where students are increasingly using mobile devices such as tablets and laptops such as iPads and Surface Go devices to take notes and perform classroom activities, as these allow students to conveniently share, collaborate and transport information. It is widely predicted that other countries such as the United Kingdom will also integrate technology in education more over the next few years; the BBC reported in 2014 that almost 70% of primary and secondary schools in the United Kingdom were using tablet computers in some form (http://www.bbc.co.uk/news/education-30216408).

There are currently successful solutions for applications that the user uses to view, edit, and produce documents, but there is a void in the market for solutions which help students specifically to effectively organise their workflows, which would help further increase their productivity and improve their education. Whilst there are a few applications which help with organisation, most of these are not directly aimed at students, and they are usually designed to be used for adults throughout each full day. The few applications which are aimed at students specifically are mobile applications, which can cause problems as many schools ban the use of mobile phones in school. My proposed alternative of a desktop application would avoid the issue of the solution breaking school rules.

Making a physical planner would require a lot more space to provide the user with the same amount of information. This can be seen in the real world, as students are constantly having to manage a plethora of files and documents, resulting in many cases of misplacing them and not being able to find them. Creating, updating, and managing information physically is very difficult, and it deters students from being organised, thus reducing their work efficiency.

By offering a computational solution, students would have their information organised and structured for them to whilst requiring minimal effort. As information would be stored on the hard drive, or even on the cloud, the possibility of physical damage becomes trivial, so it would create ease of mind that their information is safe and mostly accident-proof. The seamless, hassle-free aspect of this system would encourage users to be more organised as minimal effort would be required, making them more efficient in their work.

Based on the void in the market, my proposal is to create a desktop application which would enable students to plan and organise their workflows. This would help them to manage assignments set by teachers, manage their time more efficiently and ensure they meet deadlines.

Application Overview

My application will be an advanced school planner designed for students. It will include various aspects to help students manage their time better and work in a more structured, organised way so they can be more efficient with their learning and teaching.

The primary part of this system would be the inclusion of an interactive to-do list, as it would be the most beneficial for students by directly helping them to schedule themselves time, acting as a reminder for any tasks which need completing and tasks which they have already completed.

Other parts of the system could include a school timetable (for an overview of what may occur on different days), subject overview (to view topic lists, specification, exam information, and exam structure), and grade book (to view grades from each topic in a subject, dates of these grades, and feedback).

Stakeholders

As my application is related to organisation in education, it has various stakeholders whose needs must be catered for, even if not all of them are necessarily the end-users.

Students

The end-users of my application will be students. My application will be targeted more towards sixth form students, but it will also be designed to accommodate the needs of secondary school students.

Sixth form students are often set many assignments from the same few teachers for the same subjects. They need a method of compartmentalising their studies so that they avoid having tunnel vision and avoid neglecting some of their subjects. This would enable them to develop a greater understanding of all their A-Level subject courses, as opposed to specialising in one course but lacking understanding on the other courses, hence increasing the chances of them performing well in their A-Levels all-round and meeting conditions set for their future universities, apprenticeships, or work opportunities.

Secondary school students are set many assignments from various teachers across a broader range of subjects. They need a method of organising their school work so that they can easily recognise what assignments they have for each subject, being able to quickly differentiate between subjects so they know how much work they have for each one. This would allow them to manage their time more effectively and meet deadlines with ease.

My application would aid both types of students in differentiating between their subjects and their course topics respectively through colour coding and labelling of tasks. The built-in subject organiser would provide an easy way for these stakeholders to separate tasks of different subjects and topics through colour coding, with the user being able to edit the name and the colour of each subject/topic whenever they want.

Teachers

Teachers would be another major stakeholder of my application, as they would want an application which truly benefits their users, rather than giving a placebo effect. They would require their students to be able to quickly and easily record any assignments which they set during classes. This would save them time during classes as less time would be spent on organising homework, and more time can be spent on learning the material. Thus, students would be able to develop a greater understanding on the topics, and their grades, which the teachers are held somewhat responsible for, would likely improve. Teachers would also require my application to abide by school rules, many of which ban mobile phones from being used in school.

To achieve this, my application would provide a simple navigational user interface which allows them to add new tasks and their details, including the task title, subject, deadline, and additional written information about the task. The simple, intuitive nature of the interface would mean that there would be a gentle learning curve, with the interface being very quick and efficient to use. My application will be a desktop application rather than a mobile application so that students can use the application on their laptops or school computers, and thus avoid breaking school rules.

Parents

Parents would also be a stakeholder of my application, because they want their children (students) to succeed in their studies so that they set themselves up for success in later life. They want to ensure that their children gain as much from schooling as possible, so it is in their interest that classes consist of as much learning as possible, as opposed to time spent on organising assignments. By learning more in less time, their children would have more time to practice their understanding on topics to improve themselves independently, and thus rely less on their parents for help with studies, saving both parents and their children time. Many parents would also like to be more involved in their child's education by being notified of which tasks their child has to complete.

As aforementioned, my application would be suitable for parents by being quick and easy to use. The user interface would ensure that users are able to add new tasks with great ease of use. User inputs would be aided by dropdown menus so that users don't have to write out the subject name each time; less time would be spent on manually typing the subject names, and users would be able to easily set the deadlines without having to worry about what day of the week that numerical date is (or vice versa), as the Gregorian dates in the dropdown menu would be linked to day of the week (for example, *Monday* 14/05/2018). My application could also connect with the email addresses of parents to notify them whenever their child adds a new task to their agenda, with the details of that task, so that they keep up to date with their child's education.

Competitor Research

Google Tasks

Google Tasks is a task planner application recently released by Google for Android. It specialises in providing the user with an easy, quick method of organising tasks which they need to do in the future.

Due to the application only being recently released on the 24th April 2018, it lacks a lot of features and only offers basic functionality. Google have decided to prioritise perfecting the design and user interface of the application, refining the usability of the application with frequent updates, before adding new features further down the line.

The user is able to set tasks within different lists, with a title, description, and deadline for each task.

This application features Google's new design language, an evolution from Material Design guidelines, nicknamed 'Material Design 2'. Upon opening the app, you are greeted by a simple, yet sleek interface which aims to be minimalistic and easy to read.

As part of Google's Material Design 2 (https://material.io/design/), colour is used sparingly to avoid overwhelming the user. With sparing use of colour, Google is able to guide the user to functions which they are likely looking for. This is something which would be useful for my application to implement, as it would be quite an easy thing to do with a bit of planning, although there is a balance which needs to be struck between being minimalistic and using colour for specific purposes, and being overly dull and monotone.

For example, the button to 'Add a new task' is the only element in this page with colour; the medium blue is elegant, complements other colours in the page, and it attracts the user's eyes towards this function.

Adding a task uses up as little of the screen's real estate as possible. This aids user friendliness as it means that the user can glance at their existing tasks set to ensure that they do not record duplicate tasks. Adding a due date uses the standard calendar interface in the Android operating system, which allows the user to swipe through months of the year and then select a date.

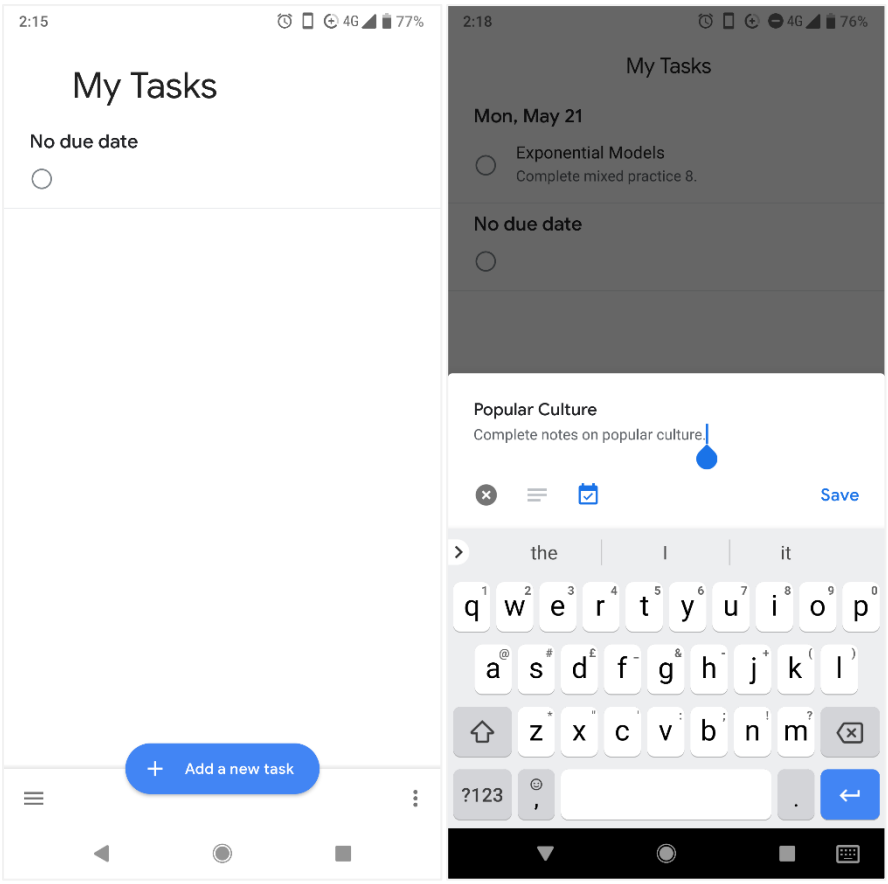
The application uses a navigational interface which should be familiar to the user; the hamburger style menu is commonly used in many other applications, and the three dots to open other options is also commonly used. This makes the application intuitive, and reduces the steepness of the learning curve for the user greatly.

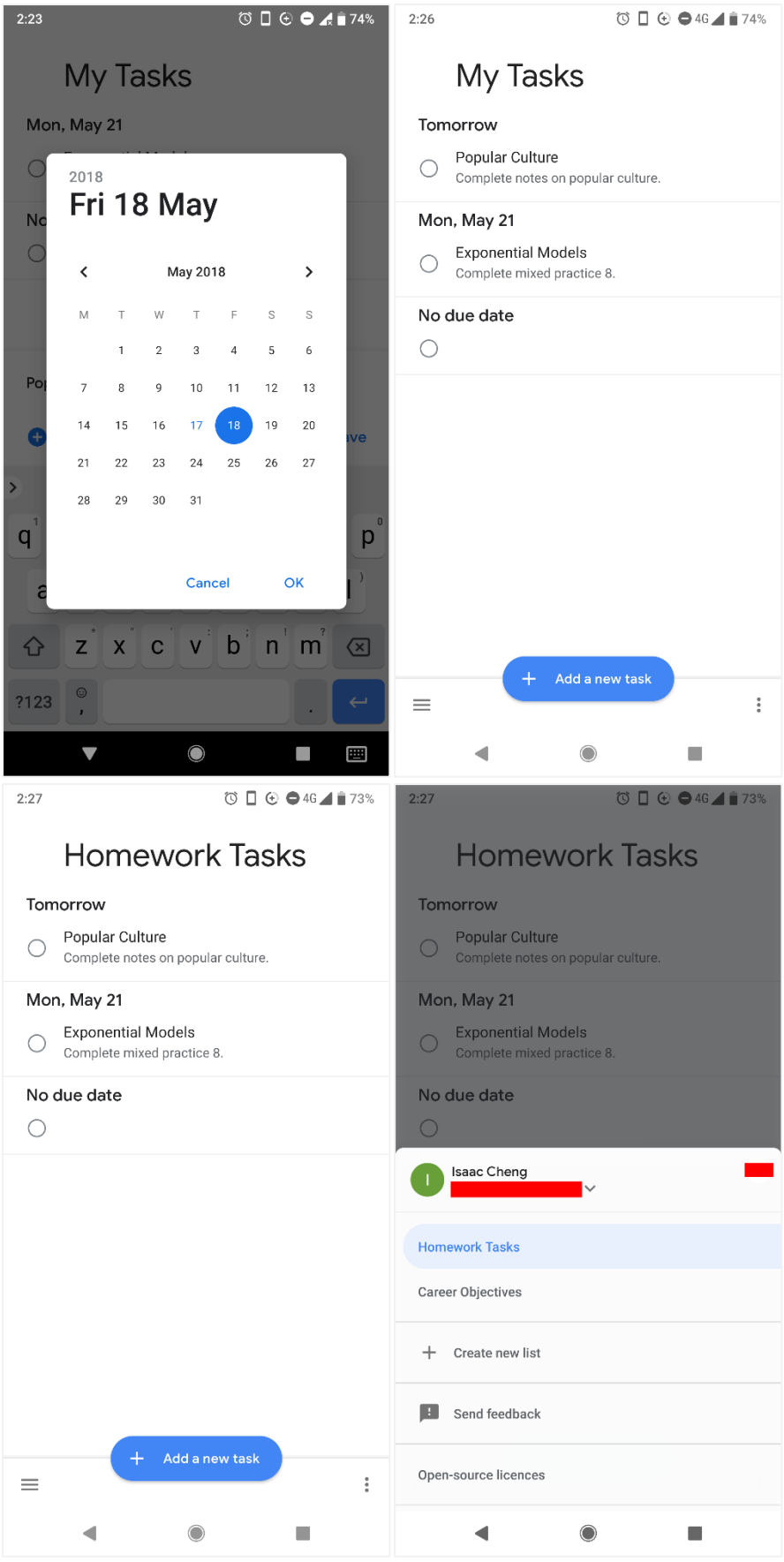
Aspects of this application which are positive and could be implemented similarly in my own application are the smart use of colour to guide user navigation, clean layout to avoid overwhelming the user, and use of standard navigational features to ensure that the application is intuitive for the user.

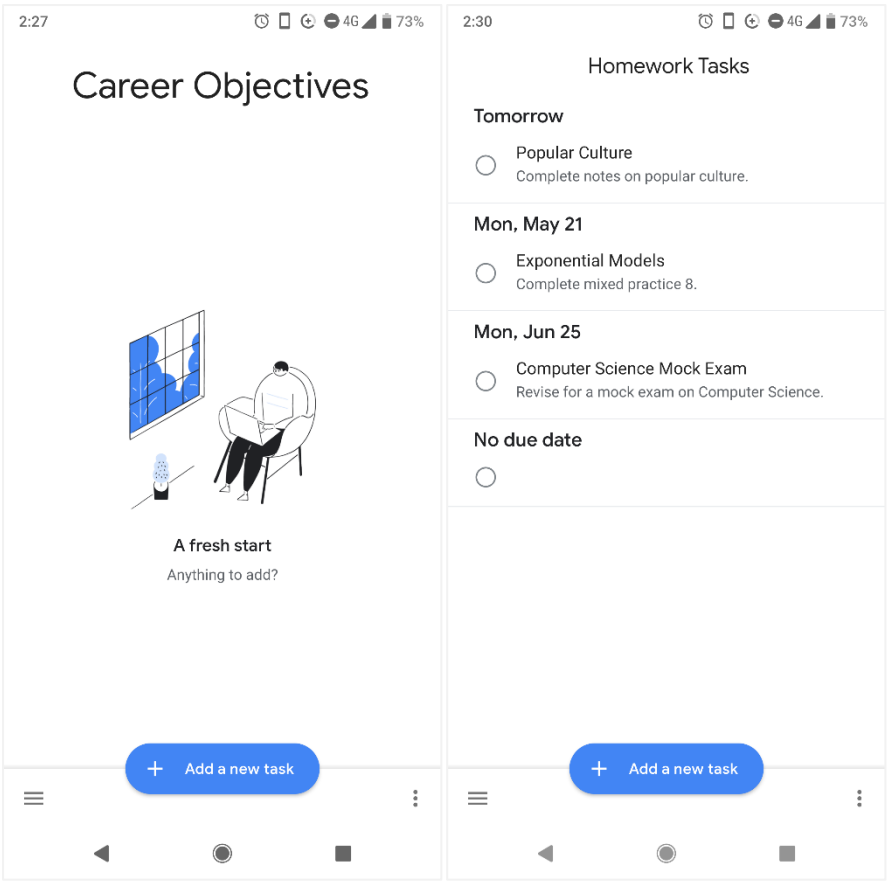
There are some points for improvement in Google Tasks, however. Functionality is very limited; the user has very few options to help them organise their tasks. Google has not added the ability for the user to colour code tasks, presumably as this may clash with the design style of their application, and because the application is aimed at daily life rather than student life. It is a simpler type of student planner than my application will be.

This application would suit my stakeholders very well, especially teachers and parents, as it is quick and easy to use, the user interface is intuitive, there is minimal bloat/unnecessary features, and tasks are organised chronologically. However, it may not cater to students as well as the other stakeholders, as the lack of colour in the application means that students may find using the application boring, and they may find it more difficult to associate tasks with their respective subjects due to the lack of colour coding options.

Overall, Google Tasks is an excellent task planning application which is simple, sleek, and barebones. My application will be designed to mimic the simplicity and ease of use of this application, but with a wider feature set to make my application more useful to students in particular in terms of functionality.


2:15
My Tasks
No due date
Add a new task
2:18
My Tasks
Mon, May 21
Exponential Models
Complete mixed practice 8.
No due date
Popular Culture
Complete notes on popular culture.
Save
the
I
it


2:23
74%
My Tasks
Mon, May 21
2018
Fri 18 May
May 2018
M T W T F S S
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 31
Cancel
OK
2:26
74%
My Tasks
Tomorrow
Popular Culture
Complete notes on popular culture.
Mon, May 21
Exponential Models
Complete mixed practice 8.
No due date
Add a new task
2:27
73%
Homework Tasks
Tomorrow
Popular Culture
Complete notes on popular culture.
Mon, May 21
Exponential Models
Complete mixed practice 8.
No due date
Add a new task
2:27
73%
Homework Tasks
Tomorrow
Popular Culture
Complete notes on popular culture.
Mon, May 21
Exponential Models
Complete mixed practice 8.
No due date
Isaac Cheng
[redacted]
Homework Tasks
Career Objectives
Create new list
Send feedback
Open-source licences



School Planner

School Planner is a school planner application designed by Andrea Dal Cin for Android. It is an application with a wide feature set, designed to help students to manage their entire student life, including an agenda (task planner), calendar, and timetable.

This application is quite mature, which means that the developer has been able to design, test, and implement many features; it was released on the 2nd November 2015, and updates are released for this application every two weeks. The developer designed the application to be sleek and intuitive, whilst offering a lot of functionality.

The user is able to add new tasks to their agenda, add subjects to their timetable for later viewing, see their homework in a calendar view, record their grades, and note down information about their teachers such as contact details.

Upon initial start-up of the application, the user is greeted with a few setup screens so that the user can enhance their experience to be specific to their school life, allowing them to configure the number of terms, types of notifications, time for notifications, and frequency of notifications.

The application was designed with Google's Material Design guidelines (Material Design 1, the predecessor to Google Tasks' Material Design 2) in mind. This makes the application especially intuitive and easy to use because the menu navigation is similar to other applications with Material Design, which developers are encouraged to use because it matches the design style of the Android operating system.

However, School Planner takes a different approach to the implementation of Material Design. Although it primarily consists of two colours in white and blue, the application uses a variety of other bright colours to bring the user's attention to the various functions of the application. This is an implementation which would be similar to that of my application; colour would be used to highlight important parts of the screen.

An example of School Planner's approach to Material Design is with the agenda. They took a similar approach with how my application will use colour coding to help the user differentiate between the tasks they have for different subjects. It is presented in a subtle way which the user can easily see.

Unlike Google Tasks, adding a task to the agenda takes up the entire screen. Despite this, the application makes up for it by offering various ways for students to record their homework in a more detailed way, which helps students ensure that they are staying on track for the task at hand. They can add details such as the title, date due, subject, pictures, and additional notes about the task. It is the small details which add to the user experience here. Subjects are selected from a dropdown menu, so users do not have to type out their subject name every time they set a task for it; they record what subjects they study, and then they can select their subject from the list. There are shortcuts for users to set commonly used due dates; later today, tomorrow, and next week. This saves time for students as they skip having to navigate through the calendar interface.

Whereas Google Tasks only contains an agenda/task planner, as the name suggests, this application has features to help the student organise their entire school life in one place. The application includes a calendar view, so students can see their workload and plan in the longer term to ensure that they do not become overloaded with tasks. It also includes a timetable, which helps students as they can enter their timetable with their subject, start time and end time, room, teacher, and any additional information. This is especially helpful for students studying in a new environment, where they may not know where particular places are.

Like Google Tasks, this application uses a hamburger menu for navigation, and the three dots for additional options. This reduces the steepness of the learning curve for the user somewhat, though it should be noted that this application is still a lot more difficult for the user to navigate than Google Tasks, as it is packed heavily with different functions, which can be very overwhelming at first, and makes navigation seem a lot more cluttered.

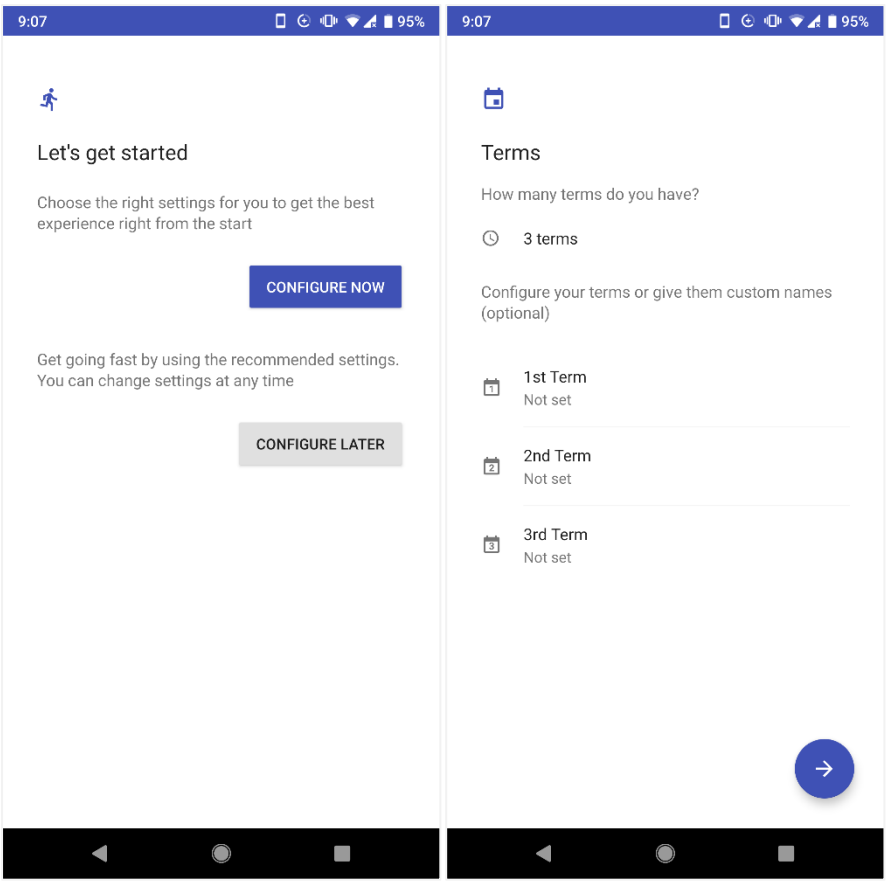
Positive aspects of this application are the good balance between minimalism and functionality in terms of colour usage for easier organisation, large feature set, and standard navigational features to ensure the application is intuitive.

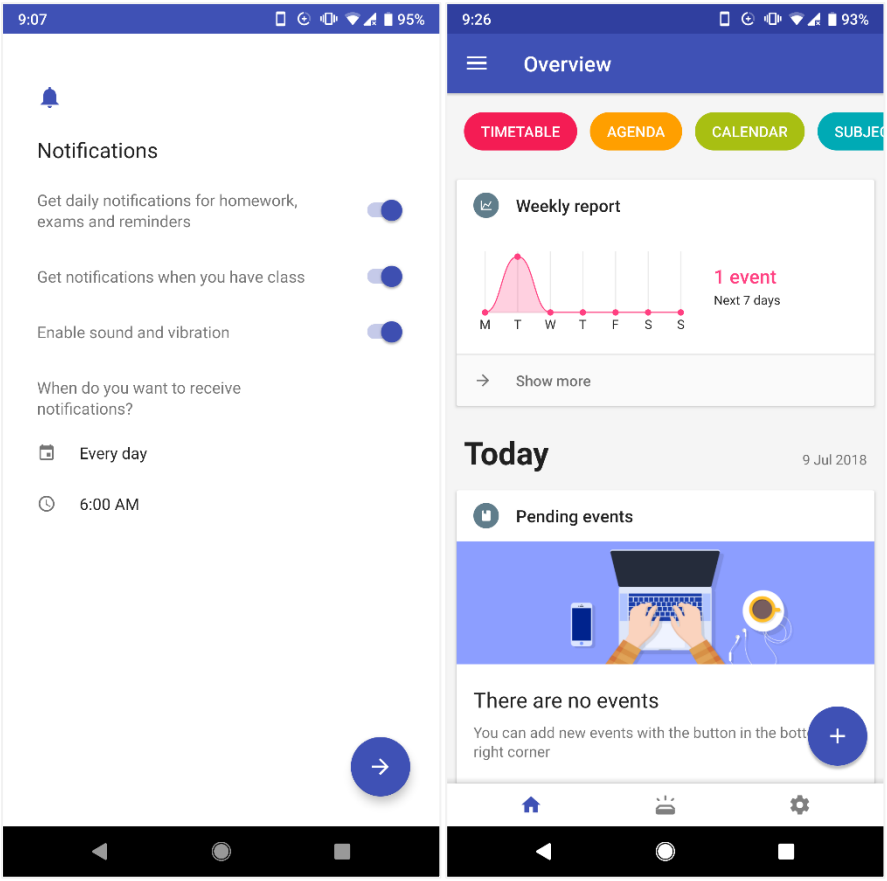
The main downside of this application is that it can easily be overwhelming to new users; the main menu especially looks cluttered, which can confuse students and cause them not to use the application. If the application used the overview screen as a main menu for the user to navigate more quickly to the screen they wanted, rather than having to use the slower method of the hamburger menu. As this would remove the briefing from the overview, a separate 'Briefing' screen could be used. The developer could also add an option for the user to change which screen is displayed when the application starts, which would be useful for students who only use the application for its agenda, for example. One minor thing to note is that I found that the months are not capitalised when adding new tasks; as shown one of the screenshots, July appeared as 'july' due to the lack of capitalisation.

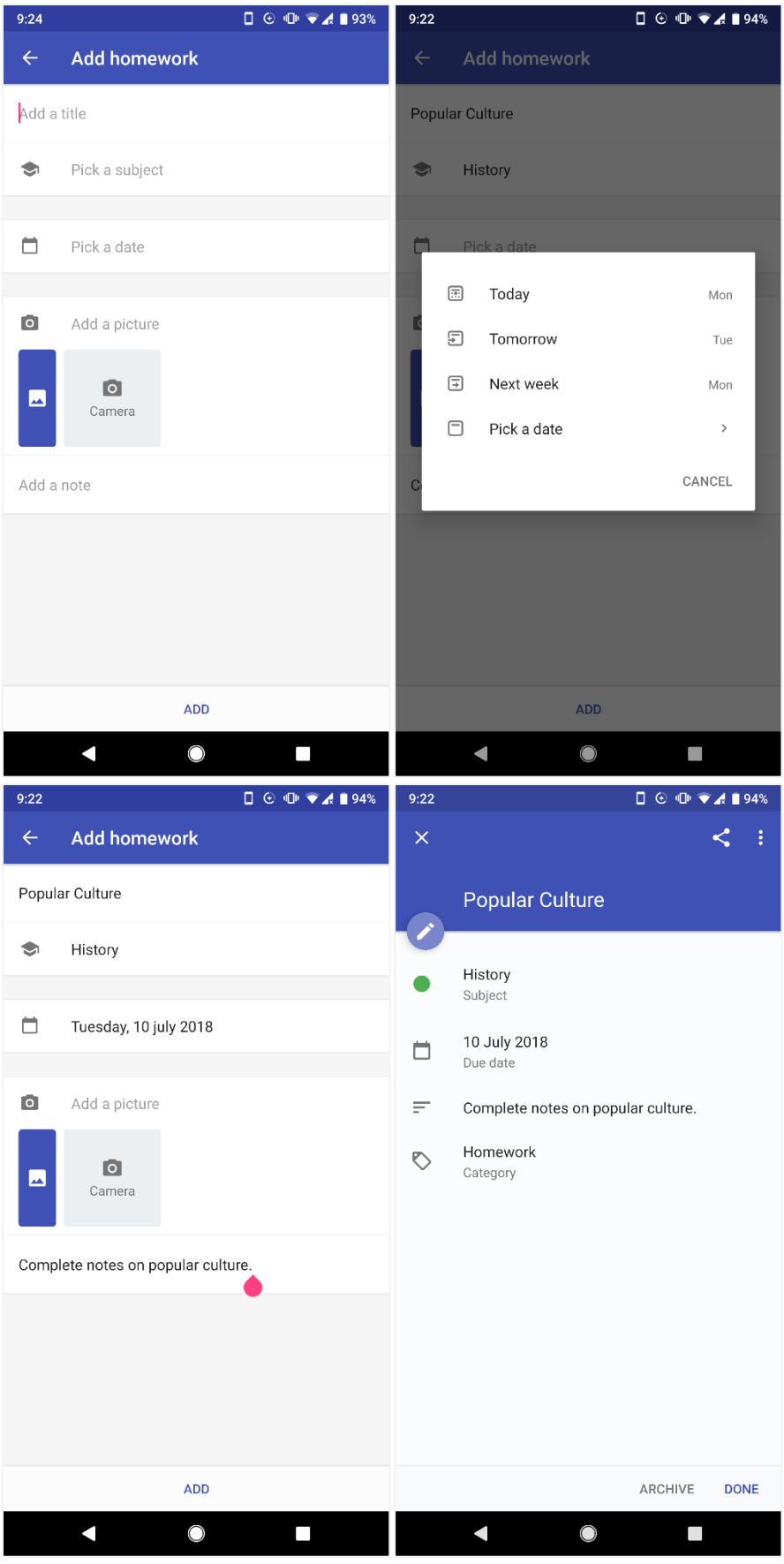
This application would suit my stakeholders very well, especially students, because its wide feature set allows students to organise their entire school life in one application, they can

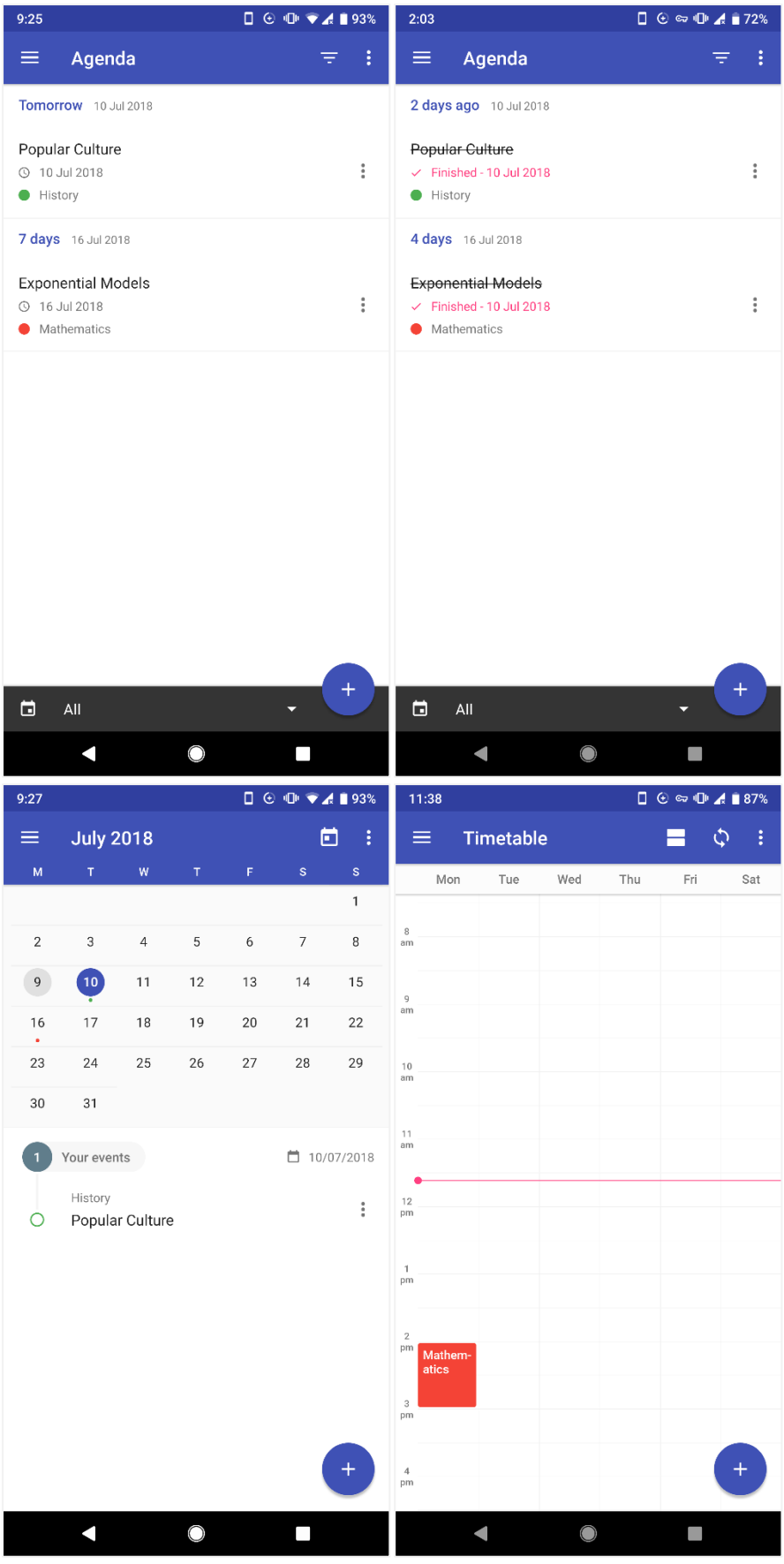
record tasks such as homework in more detail in the agenda, and shortcuts in their application means that students will be able to record tasks in less time, so they can dedicate more time to actually completing their tasks, as opposed to spending time navigating and typing details of their tasks. It also suits the needs of teachers and parents though, because an application which is useful for students will help them to work more efficiently, and thus achieve higher grades.

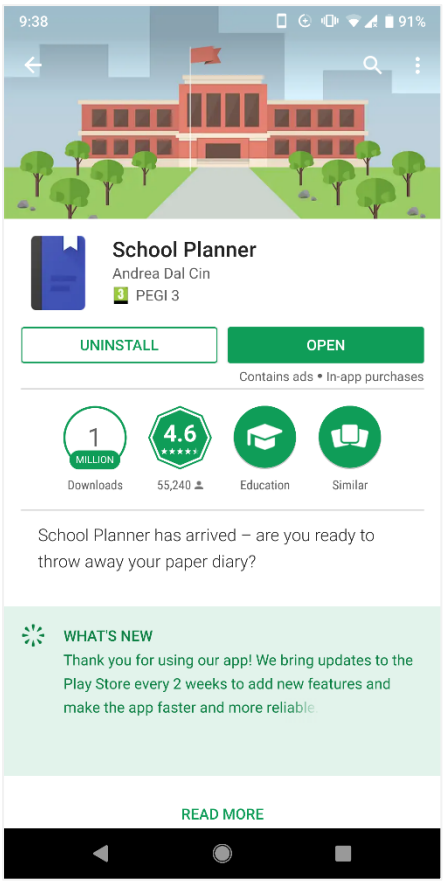
In summary, School Planner is a great school planner application which is feature-packed, functional, and customisable. My application will be designed to use similar implementation of Google's sleek Material Design guidelines alongside a large feature set, similar to this application, but my application will be less cluttered, so the user can navigate through the interface more easily.


9:07
95%
Let's get started
Choose the right settings for you to get the best experience right from the start
CONFIGURE NOW
Get going fast by using the recommended settings. You can change settings at any time
CONFIGURE LATER
9:07
95%
Terms
How many terms do you have?
3 terms
Configure your terms or give them custom names (optional)
1st Term
Not set
2nd Term
Not set
3rd Term
Not set


9:07
95%
Notifications
Get daily notifications for homework, exams and reminders
Get notifications when you have class
Enable sound and vibration
When do you want to receive notifications?
Every day
6:00 AM
9:26
93%
Overview
TIMETABLE
AGENDA
CALENDAR
Weekly report
M T W T F S S
1 event
Next 7 days
Show more
Today
9 Jul 2018
Pending events
There are no events
You can add new events with the button in the bott
right corner


9:24
93%
Add homework
Add a title
Pick a subject
Pick a date
Add a picture
Camera
Add a note
ADD
9:22
94%
Add homework
Popular Culture
History
Pick a date
Today
Mon
Tomorrow
Tue
Next week
Mon
Pick a date
CANCEL
ADD
9:22
94%
Add homework
Popular Culture
History
Tuesday, 10 july 2018
Add a picture
Camera
Complete notes on popular culture.
ADD
9:22
94%
Popular Culture
History
Subject
10 July 2018
Due date
Complete notes on popular culture.
Homework
Category
ARCHIVE
DONE


9:25
93%
Agenda
Tomorrow 10 Jul 2018
Popular Culture
10 Jul 2018
History
7 days 16 Jul 2018
Exponential Models
16 Jul 2018
Mathematics
All
2:03
72%
Agenda
2 days ago 10 Jul 2018
Popular Culture
Finished - 10 Jul 2018
History
4 days 16 Jul 2018
Exponential Models
Finished - 10 Jul 2018
Mathematics
All
9:27
93%
July 2018
M T W T F S S
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 31
1 Your events
10/07/2018
History
Popular Culture
11:38
87%
Timetable
Mon Tue Wed Thu Fri Sat
8 am
9 am
10 am
11 am
12 pm
1 pm
2 pm
3 pm
4 pm
Mathem-atics



Egenda

Egenda is a task planner application designed by gr8bit Studios for Android. It is a simple application with a more focused, no-frills feature set, designed to help students manage their class assignments with an agenda.

This application was developed for release on the 20th August 2016, and it was last updated on the 4th September 2016, which suggests that very little updates have occurred since release. However, this may be because the developer designed the application to be simple and easy to use, with minimal features so that the user is not overwhelmed with various options.

The user is able to add their subjects to the agenda and set a colour for each subject for easier recognisability, then set homework, quizzes, projects, and tests from these subjects with an intuitive user interface. Once tasks have been set, they are ordered in chronological order in a list, which shows the task title, subject and its colour, and due date in a compact view. From this list, the user can swipe away tasks to mark them as completed. The user can access list for their due tasks, completed tasks, and tasks for a specific subject. From the completed tasks list, the user can permanently delete homework (to hide them from view in case a task was accidentally set), or they can restore homework (in case they accidentally swiped a task away).

As with School Planner, this application was designed with Google's Material Design guidelines, which makes the application intuitive and easy to use due to the user being likely to be familiar with this style of menu navigation. It forms the middle ground between Google Tasks and School Planner, as it does use different colours to bring the user's attention to important aspects of the interface, but not to the large extent that School Planner does. This implementation is something which I will look to use in my application.

The main example of how Egenda's user interface becomes helpful to the user is in the main task list. They use colour in a significant way to help the user distinguish between the different subjects, all without overbearing the user. The different tasks are presented clearly in a sleek manner, whilst remaining compactness to give the user as much information as possible without needing to scroll down for more. It strikes the balance between having enough white space to avoid looking cluttered, and overwhelming the user with too much information at once.

Adding tasks is a very straightforward experience for the user in Egenda. It takes the user to a new screen, where they can enter the name of their assignment, the class (subject) it is for, the date, and any additional notes about the assignment. There are some small features which help the user enter tasks more quickly, although not as many as School Planner. The subject is selected from a dropdown menu so that students do not have to repeatedly enter the same subject name whenever they enter new assignments. The date is selected from Android's date picker, but it lacks the shortcuts to pick commonly used due dates such as tomorrow and next week, like in School Planner.

Egenda is more similar to Google Tasks than School Planner, as it only contains a task planner. This is more of a lightweight application than a fully-featured planner for students.

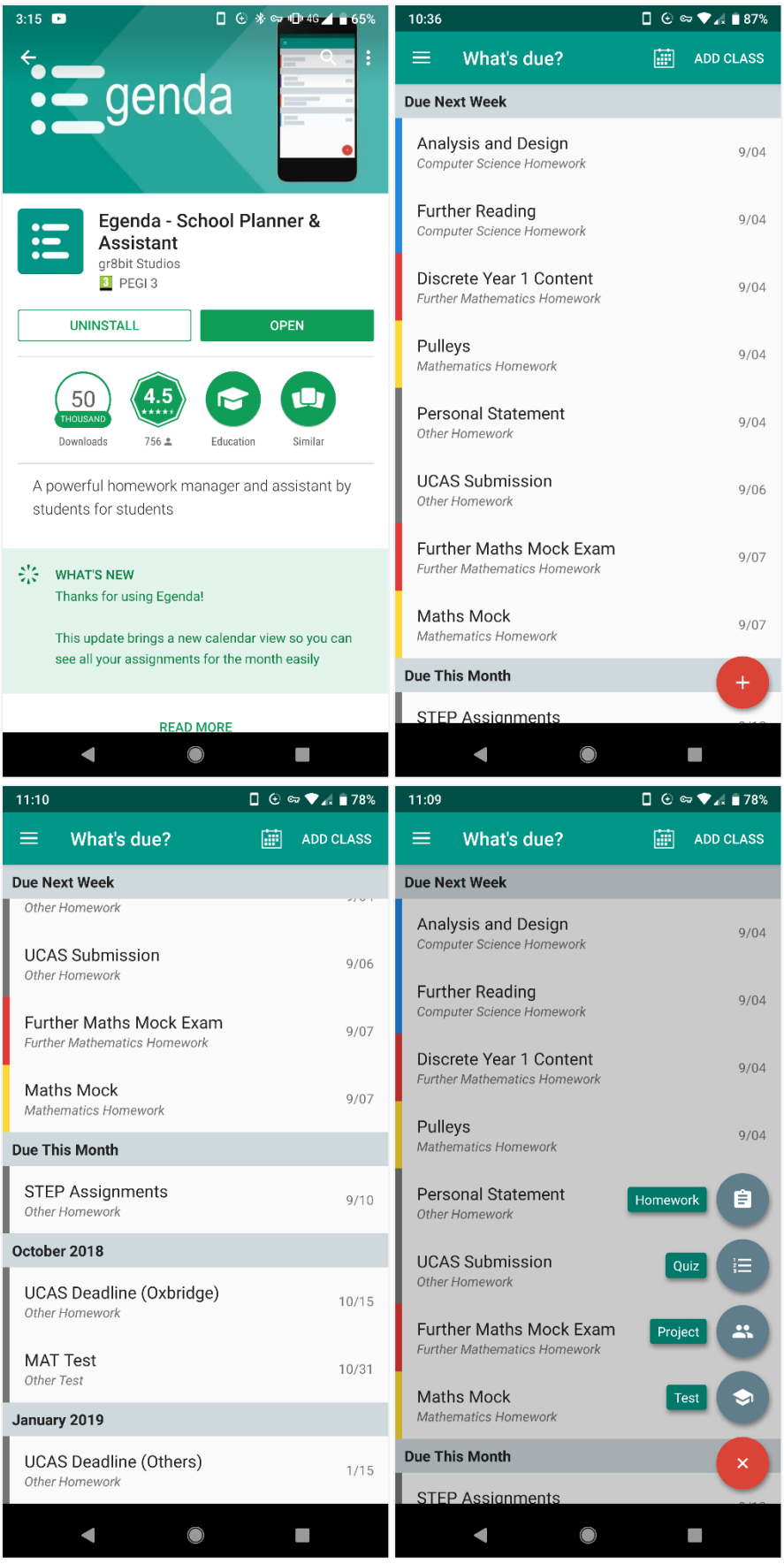
The user can navigate through a hamburger menu, where they can view due tasks and completed tasks, edit existing classes (to change their name, colour, order on the dropdown menu, and delete classes), filter to only view tasks for a specific subject, and enter the settings menu. In the settings menu, the user can toggle notifications on or off (to let the user know if they have any homework tomorrow), select the notification time, mark all assignments as completed, delete all completed assignments, and delete all data (to start fresh).

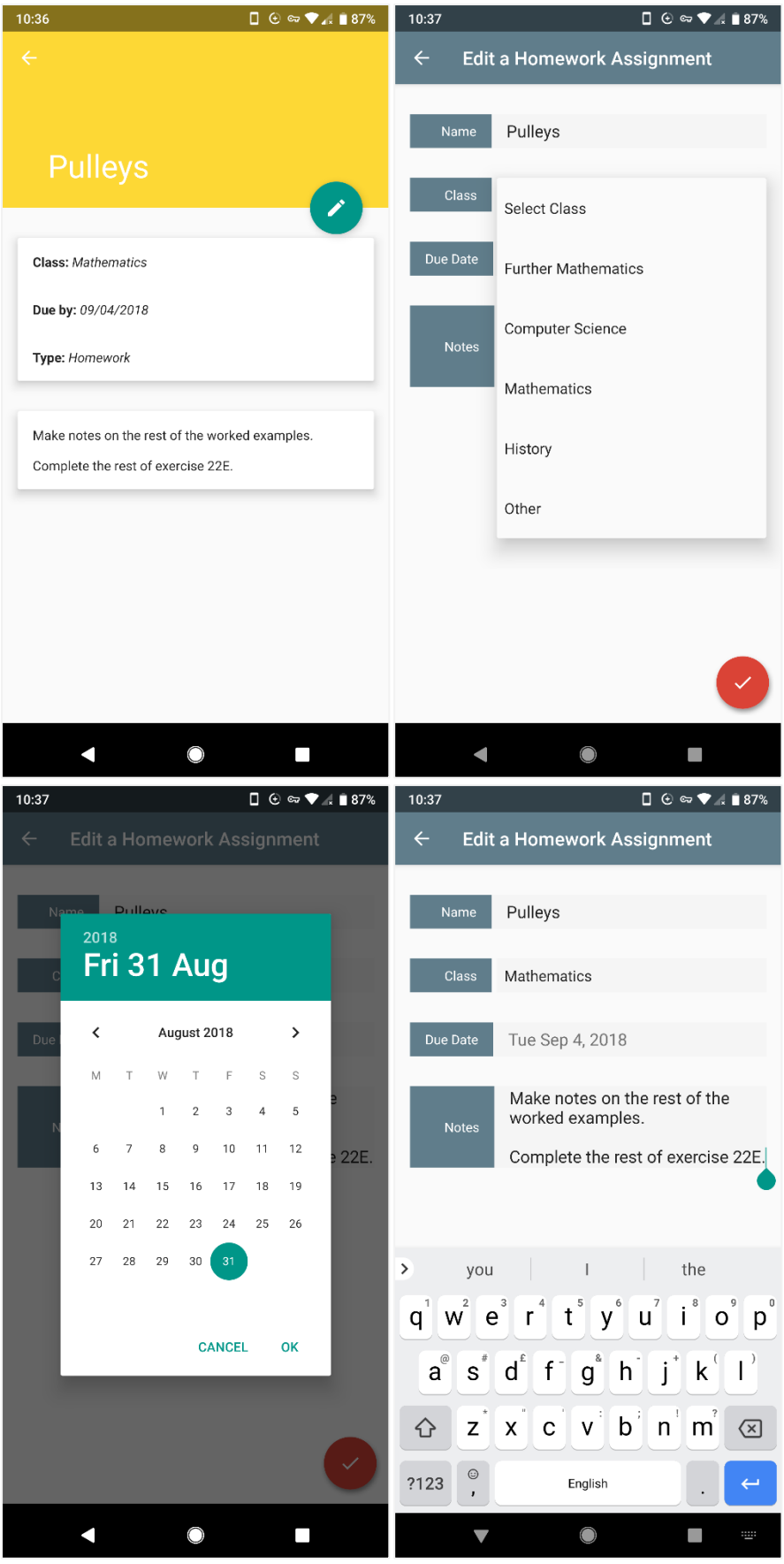
The upsides of this application are the ease of use, minimalistic design, good use of colour to give the user more information, and small learning curve.

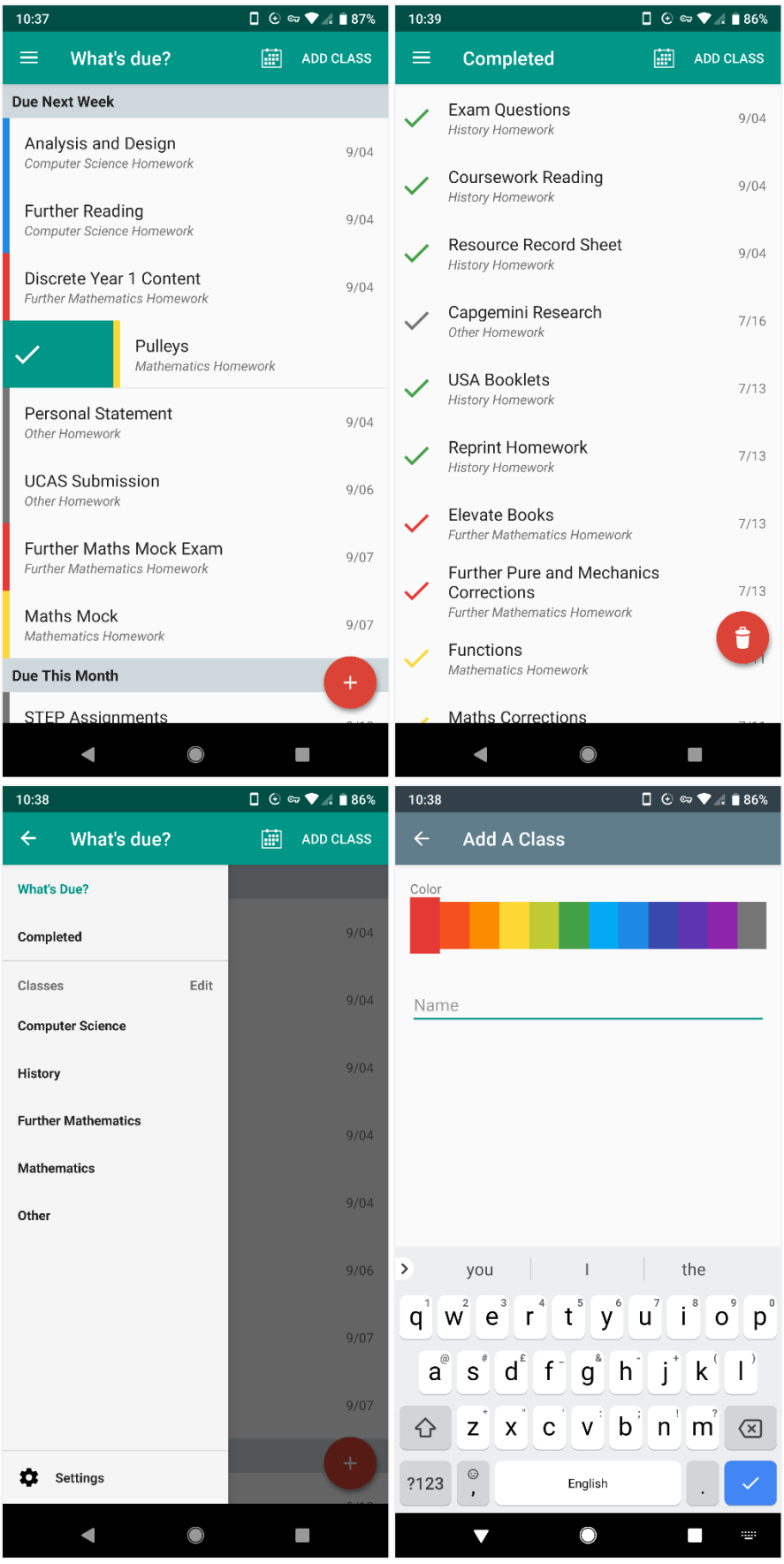
Lack of features is the major downside of this application. Whilst the features which the application provides work very well, there are parts of the application which could have additional features to make it more functional to the user. For example, it could have a feature similar to School Planner, where pictures and videos can be attached to a task to give the user additional information. The application could also implement a timetable, where the user can fill in their class schedule in case they forget.

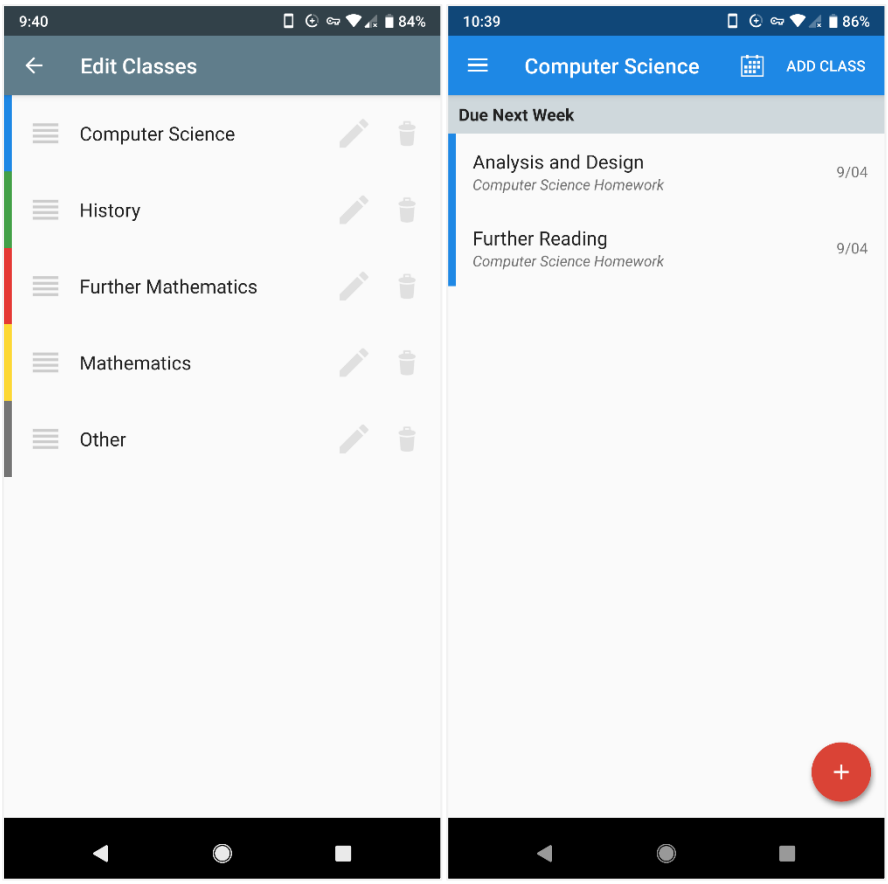
This application would suit all my stakeholders very well, because it is easy to use, with high information density, and good design through aspects such as colour coding, which makes accessing information quicker.

Egenda is an excellent task planner application which is well-polished and performs all of its features perfectly. It is quick, easy to use, and customisable, which are the three most important things for a student. My application will be designed to have the same balance of minimalism and information density as Egenda, but with more features, so that the user can record more information and use it to manage their entire school life.


3:15
65%
genda
Egenda - School Planner & Assistant
gr8bit Studios
PEGI 3
UNINSTALL
OPEN
50
THOUSAND
Downloads
4.5
756
Education
Similar
A powerful homework manager and assistant by students for students
WHAT'S NEW
Thanks for using Egenda!
This update brings a new calendar view so you can see all your assignments for the month easily
READ MORE
10:36
87%
What's due?
ADD CLASS
Due Next Week
Analysis and Design
Computer Science Homework
9/04
Further Reading
Computer Science Homework
9/04
Discrete Year 1 Content
Further Mathematics Homework
9/04
Pulleys
Mathematics Homework
9/04
Personal Statement
Other Homework
9/04
UCAS Submission
Other Homework
9/06
Further Maths Mock Exam
Further Mathematics Homework
9/07
Maths Mock
Mathematics Homework
9/07
Due This Month
STEP Assignments
11:10
78%
What's due?
ADD CLASS
Due Next Week
Other Homework
UCAS Submission
Other Homework
9/06
Further Maths Mock Exam
Further Mathematics Homework
9/07
Maths Mock
Mathematics Homework
9/07
Due This Month
STEP Assignments
Other Homework
9/10
October 2018
UCAS Deadline (Oxbridge)
Other Homework
10/15
MAT Test
Other Test
10/31
January 2019
UCAS Deadline (Others)
Other Homework
1/15
11:09
78%
What's due?
ADD CLASS
Due Next Week
Analysis and Design
Computer Science Homework
9/04
Further Reading
Computer Science Homework
9/04
Discrete Year 1 Content
Further Mathematics Homework
9/04
Pulleys
Mathematics Homework
9/04
Personal Statement
Other Homework
Homework
UCAS Submission
Other Homework
Quiz
Further Maths Mock Exam
Further Mathematics Homework
Project
Maths Mock
Mathematics Homework
Test
Due This Month
STEP Assignments


10:36
87%
Pulleys
Class: Mathematics
Due by: 09/04/2018
Type: Homework
Make notes on the rest of the worked examples.
Complete the rest of exercise 22E.
10:37
87%
Edit a Homework Assignment
Name
Pulleys
Class
Select Class
Due Date
Further Mathematics
Notes
Computer Science
Mathematics
History
Other
10:37
87%
Edit a Homework Assignment
2018
Fri 31 Aug
August 2018
M T W T F S S
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 31
CANCEL
OK
10:37
87%
Edit a Homework Assignment
Name
Pulleys
Class
Mathematics
Due Date
Tue Sep 4, 2018
Notes
Make notes on the rest of the worked examples.
Complete the rest of exercise 22E.
you
I
the
English


10:37
87%
What's due?
ADD CLASS
Due Next Week
Analysis and Design
Computer Science Homework
9/04
Further Reading
Computer Science Homework
9/04
Discrete Year 1 Content
Further Mathematics Homework
9/04
Pulleys
Mathematics Homework
Personal Statement
Other Homework
9/04
UCAS Submission
Other Homework
9/06
Further Maths Mock Exam
Further Mathematics Homework
9/07
Maths Mock
Mathematics Homework
9/07
Due This Month
STEP Assignments
10:39
86%
Completed
ADD CLASS
Exam Questions
History Homework
9/04
Coursework Reading
History Homework
9/04
Resource Record Sheet
History Homework
9/04
Capgemini Research
Other Homework
7/16
USA Booklets
History Homework
7/13
Reprint Homework
History Homework
7/13
Elevate Books
Further Mathematics Homework
7/13
Further Pure and Mechanics Corrections
Further Mathematics Homework
7/13
Functions
Mathematics Homework
Maths Corrections
10:38
86%
What's due?
ADD CLASS
What's Due?
Completed
Classes
Edit
Computer Science
History
Further Mathematics
Mathematics
Other
Settings
10:38
86%
Add A Class
Color
Name
you
I
the
English
?123



Show My Homework

Show My Homework is a school planner application by Naimish Gohil in the form of a website. It is an application with a fully-fledged feature set, designed to help students manage their student life. It includes a task planner, homework calendar, timetable, and grade book.

This service was released in 2011, so the developers have had a long time to make refinements to design and functionality. It has been designed to be easy to use, with simplicity being the key so that less computer literate students and teachers can use the service.

Unlike other competitors, this service works for the teachers and the students. The school administrators add the usernames of students and teacher to classes, and from there, teachers can set new tasks to their classes. Students can view their tasks in a to-do list or calendar, access their class timetable, and access a record of their grades.

The design of the website is very straightforward. It has a clean interface to avoid overwhelming the user with information. The colour scheme is very bland; there are colourful highlights here and there for branding and aesthetic purposes, but the only functional use of colour is in the task viewer (where it uses green to indicate date set, and red to indicate date due), and the task calendar (where it uses colours to differentiate between the types of tasks).

When viewing a task, the student is presented with the type of assignment (such as homework, or class test), title, date set, date due, description, estimation for time required to complete the task, and instructions for how to hand in the homework. The most important details, the date set and date due, are highlighted with green and red boxes respectively,

which contrasts with the rest of the grayscale interface to attract the user's attention. A comment can be sent to the teacher if the student has any queries about the task. The student can also submit their homework online through Show My Homework, and they can check for their grade on that specific homework.

The timetable was non-functional at the time of testing. This is because the service relies on the school administrators providing the information to be relayed to the student; if the information is not inputted, the student cannot use the timetable.

Meanwhile, the grade book showed a list of the homework tasks completed, but it did not indicate the status or grades from these tasks. This is because the service relies on the status and grade being updated by the teacher; if these are not updated by the teacher, the student cannot access this information.

There is a static navigation bar on the left, which allows the user to navigate through the different functions quickly. This is something which my application could benefit from implementing, as it would make my application easy to use, and reduce the learning curve for the user.

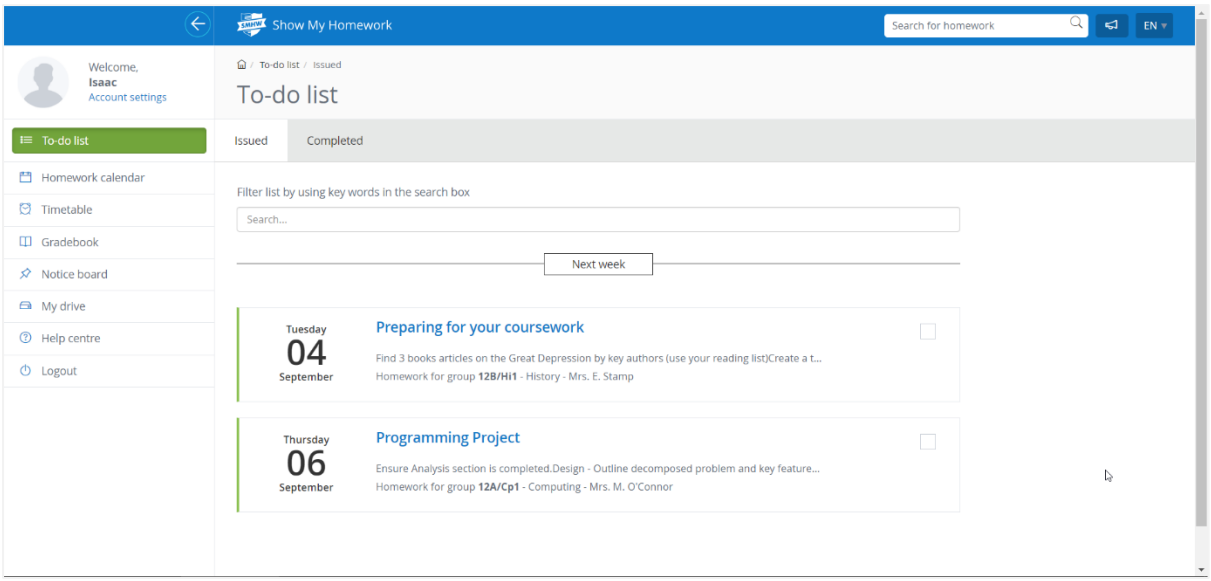
As this service comes in the form of a website, the information can be accessed easily from any device with an internet connection, which makes it more convenient than some alternatives.

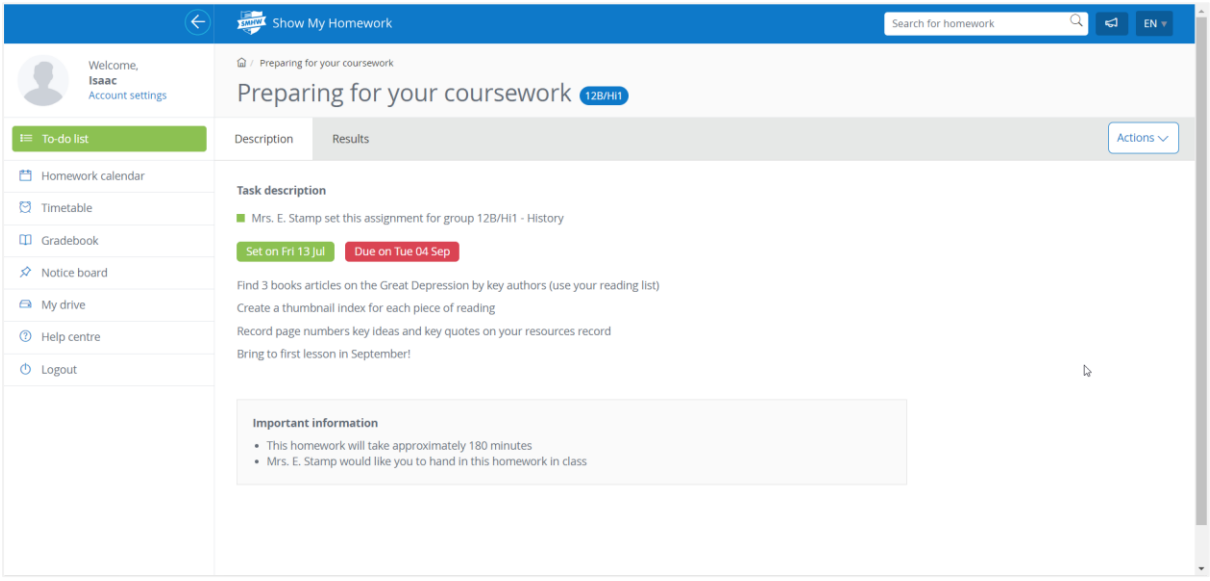
The upsides of this application are the ease of use, ease of access and convenience, and synchronisation of information between teachers and students.

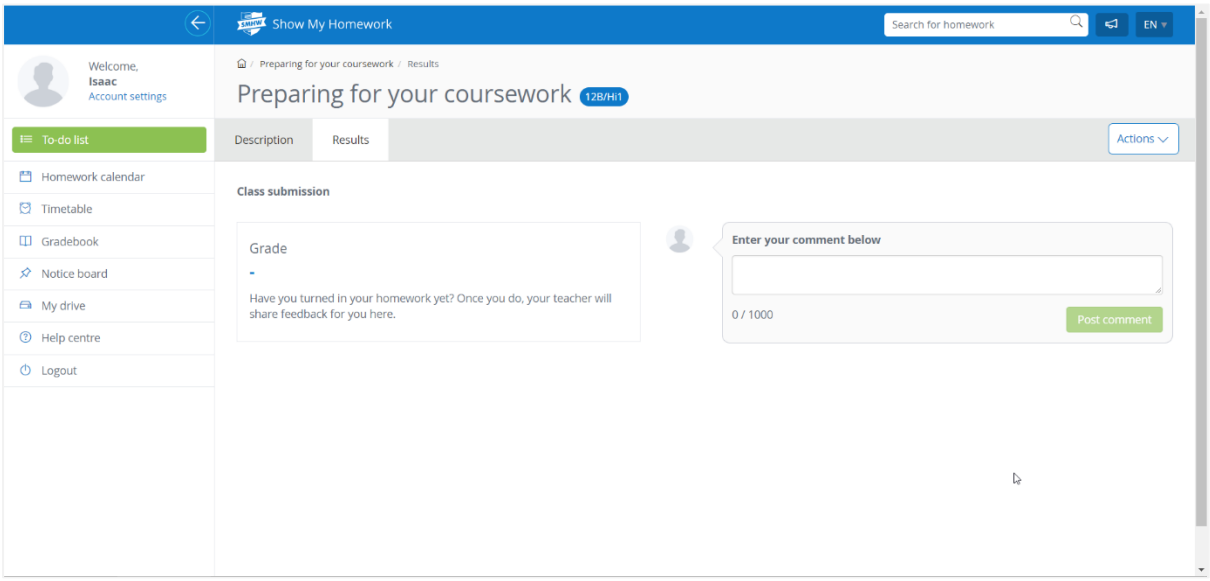
Areas where this service could improve upon are reliance on the school and teachers to provide information for the student to access, lack of functionality for students, and lack of colour coding to provide additional information to the student.

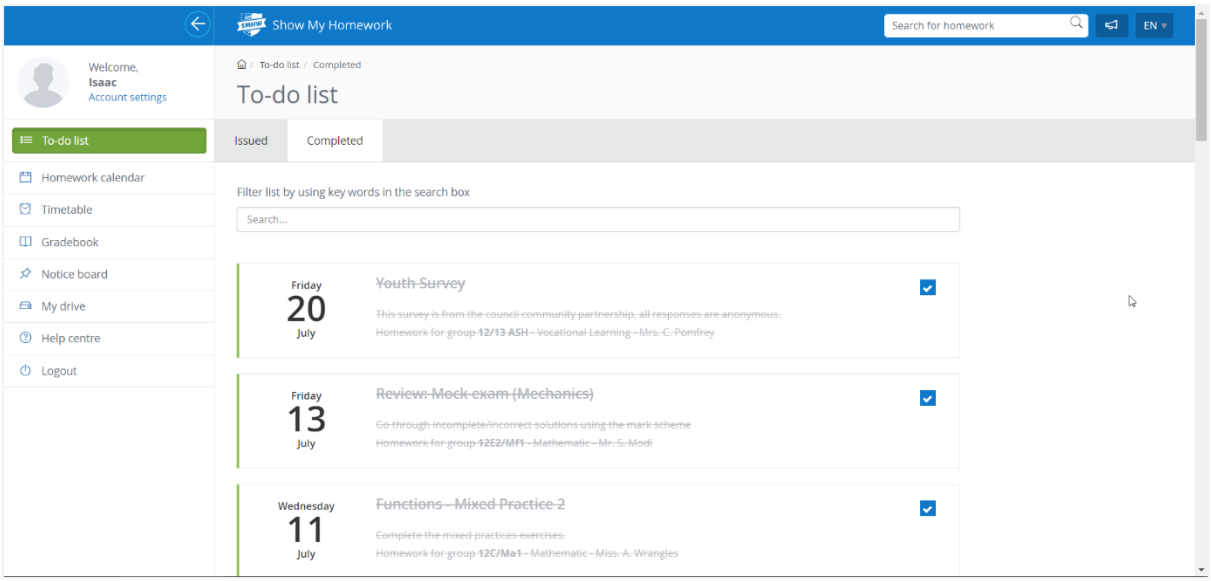
This service would suit my stakeholders decently for teachers, because it is easy to use, and provides a method of synchronising information with their students. However, it may not suit the students or parents as much, because the lack of information available to students may cause them to waste time in trying to enquire teachers for more information in cases where they suspect a wrong due date has been set, perhaps due to a lack of computer literacy skills from teachers.

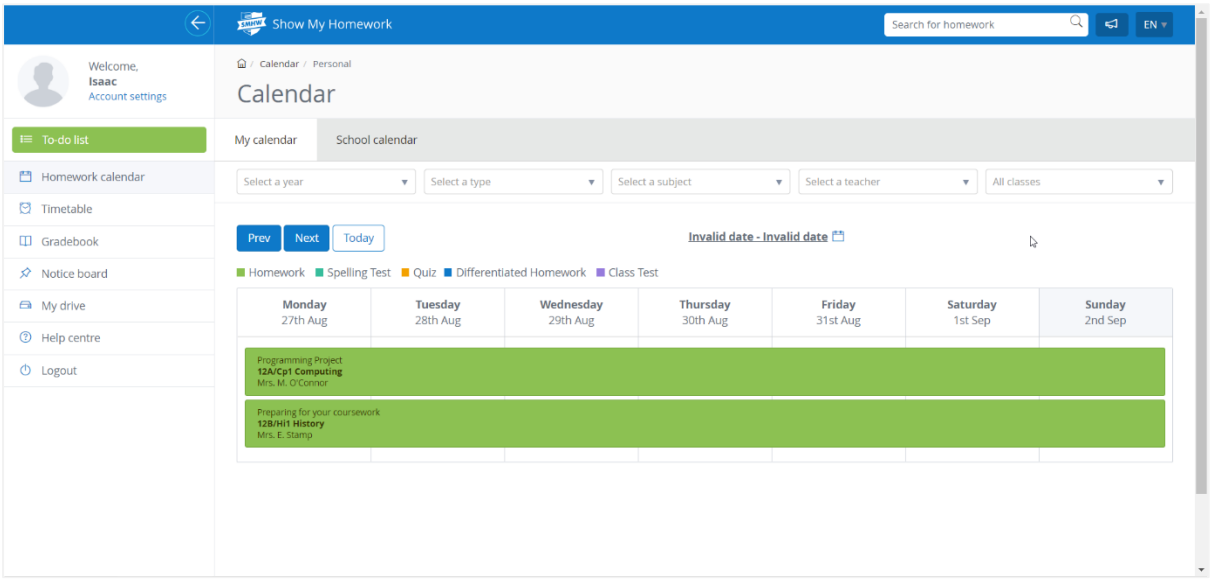
In conclusion, Show My Homework is a good school planning application which is simple, easy to use, and convenient. My application will be designed to mimic its ease of use, whilst allowing the student to record their own information rather than relying on the school and teachers, and providing greater functionality.

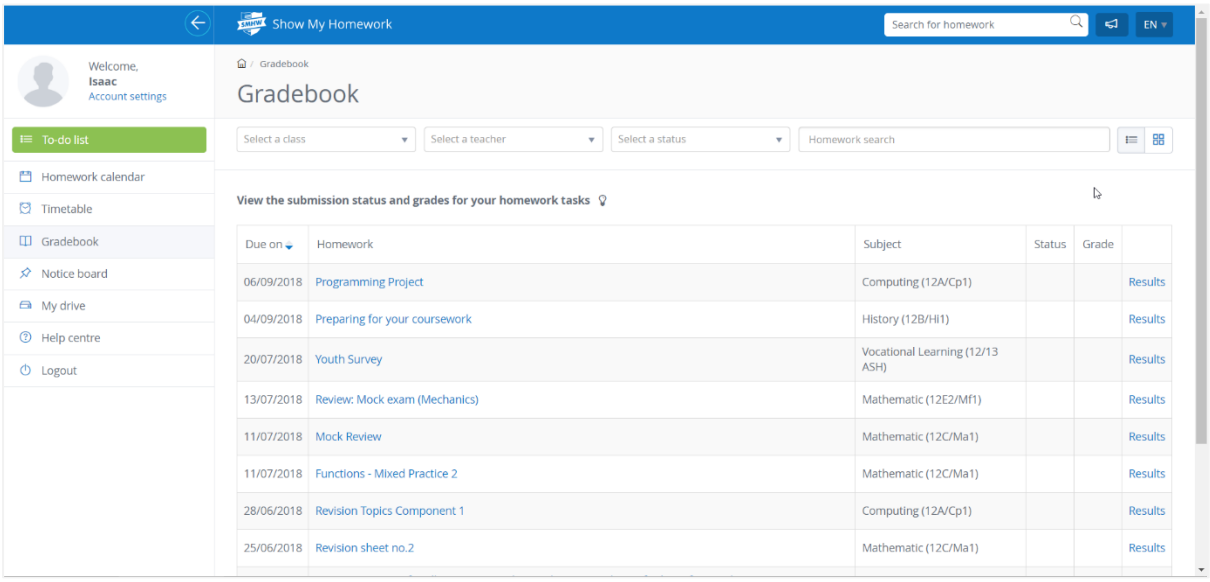

Show My Homework
Search for homework
EN
Welcome,
Isaac
Account settings
To-do list
Homework calendar
Timetable
Gradebook
Notice board
My drive
Help centre
Logout
To-do list / Issued
To-do list
Issued
Completed
Filter list by using key words in the search box
Search...
Next week
Tuesday
04
September
Preparing for your coursework
Find 3 books articles on the Great Depression by key authors (use your reading list)Create a t...
Homework for group 12B/Hi1 - History - Mrs. E. Stamp
Thursday
06
September
Programming Project
Ensure Analysis section is completed.Design - Outline decomposed problem and key feature...
Homework for group 12A/Cp1 - Computing - Mrs. M. O'Connor


Show My Homework
Search for homework
EN
Welcome,
Isaac
Account settings
To-do list
Homework calendar
Timetable
Gradebook
Notice board
My drive
Help centre
Logout
Preparing for your coursework
Preparing for your coursework 12B/Hi1
Description
Results
Actions
Task description
Mrs. E. Stamp set this assignment for group 12B/Hi1 - History
Set on Fri 13 Jul
Due on Tue 04 Sep
Find 3 books articles on the Great Depression by key authors (use your reading list)
Create a thumbnail index for each piece of reading
Record page numbers key ideas and key quotes on your resources record
Bring to first lesson in September!
Important information
This homework will take approximately 180 minutes
Mrs. E. Stamp would like you to hand in this homework in class


Show My Homework
Search for homework
EN
Welcome,
Isaac
Account settings
To-do list
Homework calendar
Timetable
Gradebook
Notice board
My drive
Help centre
Logout
Preparing for your coursework / Results
Preparing for your coursework 12B/Hi1
Description
Results
Actions
Class submission
Grade
-
Have you turned in your homework yet? Once you do, your teacher will share feedback for you here.
Enter your comment below
0 / 1000
Post comment


Show My Homework
Search for homework
EN
Welcome,
Isaac
Account settings
To-do list
Homework calendar
Timetable
Gradebook
Notice board
My drive
Help centre
Logout
To-do list / Completed
To-do list
Issued
Completed
Filter list by using key words in the search box
Search...
Friday
20
July
Youth Survey
This survey is from the council community partnership, all responses are anonymous.
Homework for group 12/13 ASH - Vocational Learning - Mrs. C. Pomfrey
Friday
13
July
Review: Mock exam (Mechanics)
Go through incomplete/incorrect solutions using the mark scheme
Homework for group 12E2/MF1 - Mathematic - Mr. S. Modi
Wednesday
11
July
Functions - Mixed Practice 2
Complete the mixed practices exercises.
Homework for group 12C/Ma1 - Mathematic - Miss. A. Wrangles


Show My Homework
Search for homework
EN
Welcome,
Isaac
Account settings
To-do list
Homework calendar
Timetable
Gradebook
Notice board
My drive
Help centre
Logout
Calendar / Personal
Calendar
My calendar
School calendar
Select a year
Select a type
Select a subject
Select a teacher
All classes
Prev
Next
Today
Invalid date - Invalid date
Homework
Spelling Test
Quiz
Differentiated Homework
Class Test
Monday 27th Aug
Tuesday 28th Aug
Wednesday 29th Aug
Thursday 30th Aug
Friday 31st Aug
Saturday 1st Sep
Sunday 2nd Sep
Programming Project
12A/Cp1 Computing
Mrs. M. O'Connor
Preparing for your coursework
12B/Hi1 History
Mrs. E. Stamp


Show My Homework
Search for homework
EN
Welcome,
Isaac
Account settings
To-do list
Homework calendar
Timetable
Gradebook
Notice board
My drive
Help centre
Logout
Gradebook
Gradebook
Select a class
Select a teacher
Select a status
Homework search
View the submission status and grades for your homework tasks
Due on
Homework
Subject
Status
Grade
06/09/2018 Programming Project Computing (12A/Cp1) Results
04/09/2018 Preparing for your coursework History (12B/Hi1) Results
20/07/2018 Youth Survey Vocational Learning (12/13 ASH) Results
13/07/2018 Review: Mock exam (Mechanics) Mathematic (12E2/Mf1) Results
11/07/2018 Mock Review Mathematic (12C/Ma1) Results
11/07/2018 Functions - Mixed Practice 2 Mathematic (12C/Ma1) Results
28/06/2018 Revision Topics Component 1 Computing (12A/Cp1) Results
25/06/2018 Revision sheet no.2 Mathematic (12C/Ma1) Results

Specification

Based on the considerations made in the problem analysis, application overview, and competitor research, my proposed solution must meet the following specifications:

- A program which is coded in Python through PyQt5, a comprehensive set of Python bindings which implement over 35 extension modules.

- A user interface which is easy to read and has high information density.
    - Clean design with a balance of whitespace and information compactness to prevent the user requiring to repeatedly scroll, or strain their eyes to find specific details.
        - Use aspects from Google's Material Design guidelines to achieve the correct balance.
            - *This may be limited based on time constraints, as adhering to all of Material Design guidelines could take a long time due to the guidelines being very particular.*
            - *Instead of directly following Material Design guidelines, my application may take inspiration from certain aspects of it.*
    - Colour coding to differentiate between subjects more easily.
        - *This may be limited as it may conflict with the clean design of the user interface which I also want. It depends on how the user interface will look in practice, as PyQt5 may have a different aesthetic to what I have planned.*

- A navigational interface which is quick and easy to use.
    - Static navigation bar to reduce number of clicks required.
    - Buttons with text labels to avoid ambiguity.
    - Optionally, simple icons next to text labels to help users who prefer to use visual information to navigate.
        - *This may be limited due to time constraints; designing high quality icons of a consistent theme for all of the buttons in my application is likely to take too much time.*
        - *It may also be a distraction to the user in an application where users may prefer simplicity for organisation.*
    - Lay out elements in a logical way to avoid confusing the user.

- Wide feature set which provides good user functionality.
    - Record information about subjects and teacher names.
        - *This may be limited as teacher names would not be as important for the user, meaning it would be a lower priority feature. Depending on*
    - Add new tasks to an agenda, with details such as task title, subject, date due, and additional notes to manage assignments.
        - Use a dropdown menu to select subject.
        - Use a calendar interface to select date due, with shortcuts for common dates such as tomorrow and next week.
            - *This may be limited as it is dependent on the modules available in Python; creating my own user interface from scratch to select dates would be both very difficult, and very time consuming.*
            - *I would need to perform more research into PyQt5 to see if there are any built-in widgets for this. Some preliminary*

research has discovered that the QCalendar widget may be very useful for my application.
- *Instead of a calendar interface like the one built-in from Android, my application may use dropdown menus for the day, month, and year.*

- Create a school timetable so the user can refer to it in case they forget.
  - Grid interface.
  - User can select a ready-made template for their timetable, then fill in the cells of the grid.
    - *This may be limited because it would be difficult to make my user interface scale correctly for different timetable templates; a user interface with 5 x 5 cells would scale very differently to a user interface with 10 x 10 cells.*
    - *As most people have a timetable template of five periods across the five week days, it may be more appropriate to make the timetable have a fixed size, as it is easiest to design the program for the lowest common denominator user/the largest user base.*
- Record grades in a grade book, which specifies date, subject, topic, grade, and feedback.
  - *This may be limited because it would not be as important as the other parts of my application, such as Agenda and Timetable.*
  - *These two parts would be used much more than Grade Book, so it would be a higher priority to develop these features first.*

Hardware and Software Requirements

<u>Hardware Requirements</u>

- CPU:
  - 1.0 GHz or higher clock speed.
    - Required to ensure that the user's system will remain stable whilst this program and other common programs in the background are run simultaneously.
  - 2 or more cores.
    - Required to ensure that the user's system will remain stable whilst this program and other common programs in the background are run simultaneously.
- Memory:
  - 1GB or more.
    - Required to ensure that the user's system will remain stable whilst this program and other common programs in the background are run simultaneously.
- Storage:
  - 50MB or more available.
    - Required to ensure that the data persistence in my application will be fully functional.
- Peripherals:
  - Keyboard and mouse.
    - Required to navigate the user interface and access all functions of my application.

Software Requirements

- Operating system:
    - Windows 8 or later.
        - The application will not be tested on earlier versions of Windows, so there is no guarantee that my application will be fully functional.
    - Python.
        - The application will be made running the Python programming language.
    - PyQt5.
        - The application will be running using the Python binding called PyQt5.

Success Criteria

To ensure that my project will be completed to specification, a success criterion will be used as a check that development remains focused on the objectives.

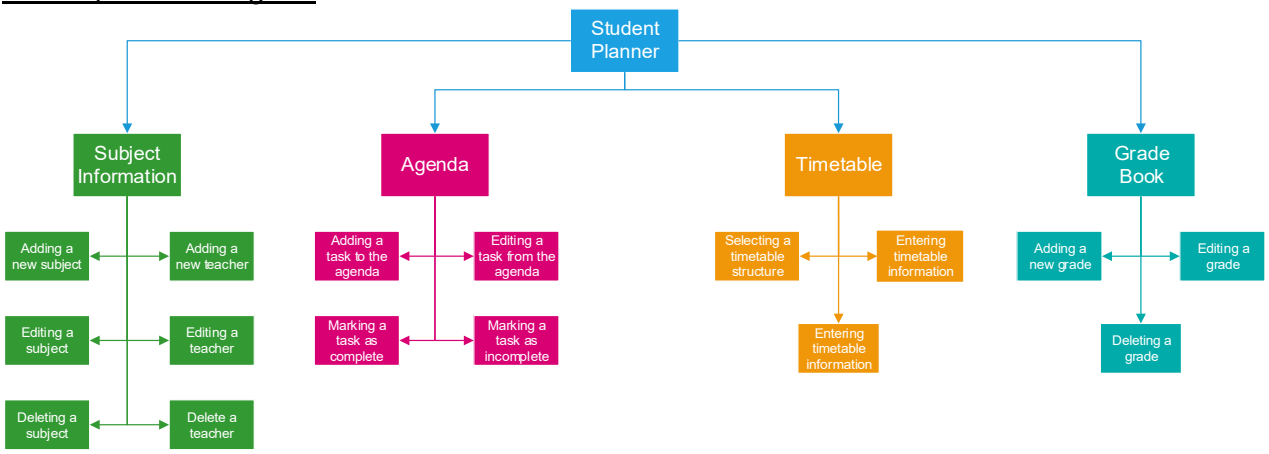
- The user must be able to enter their subjects and assign them a colour, which should both be stored in data files. This is because these subjects will be needed for the dropdown menu when adding a task to the agenda, and the colours will be displayed alongside the task on the agenda to help the user distinguish between the tasks from various subjects.
- The user must be able to add a task to the agenda with a title, subject, due date, and additional notes, which should all be stored in data files. This is so that the user can view a list of their tasks, and expand tasks from this list to view details about the task to help organise their workload.
- The user must be able to select the subject which a task is delegated for through a dropdown menu. This is to improve the user experience, allowing the user to create tasks more quickly and with less mistakes.
- The user must be able to select a timetable template through a user interface, and enter subjects in individual cells of the table, which would be stored to data files. This would allow the user to create a timetable that shows the lessons the user will have on each day, on a period by period basis, to help the user arrive to the correct lessons on time.
- *(Optionally, depending on time constraints)* The user must be able to enter the grades which they achieve in a subject to a grade book, with the title of the topic, grade, date achieved, and feedback, which would be stored to data files. This would allow the user to access an overview of their grades at another time, so that they can see which topics they need to improve at.

Design

Decomposition of Problem

As a school planner, my application will have various problems which need to be decomposed before they are solved, to ensure that each part of the problem is addressed fully. The following chart shows a visual representation of my decomposition of my student planner application, alongside a table which explains and justifies why the solutions to these problems meet the needs of my stakeholders.

Decomposition Diagram



Decomposition Table: My Subjects

Problem	**Breakdown**	**Justification**
Adding a new subject.	When the user wants to add a new subject, they should be able to click on a + icon on this screen, where they will be able to enter their subject name, and give it a colour to represent that subject.	A graphical user interface would make the menu more accessible for the user to navigate. By colour coding each subject, the information from the rest of the application will be easier and quicker to read as the user can scan through information by colour association.
Editing a subject.	When the user wants to edit an existing subject, they should be able to click on the subject, then click on the pencil icon to take them to a screen where they will be able to edit their subject name and colour.	By being able to edit their subject, the user is able to update that subject whilst retaining all their existing information associated with that subject, such as homework set for it, and timetable slots. Using the pencil icon to represent the edit button would be good for the user experience because this is a standard in the software industry.
Deleting a subject.	When the user wants to delete a subject, they should be able to click on the subject, then click on the bin icon. This would prompt the user on a confirmation screen to check that they are sure about deleting that subject, where they can confirm it to delete the subject.	The user may no longer be studying a subject, so being able to delete a subject would remove the clutter from that subject from the rest of the application, such as redundant tasks on their agenda. The use of a confirmation prompt will prevent accidental deletions.
Adding a new teacher.	When the user wants to add a new teacher to their subject, they should be able to click on a + icon on the subject screen, where they will be able to enter their teacher name, classroom, and email address.	This would allow the user to refer back to the application if they need to contact their teacher, either in person or via email. It would also help them to remember which classroom they need to be in if their subject has multiple teachers in different classrooms.
Editing a teacher.	When the user wants to edit the information of a teacher, they should be able to click on that teacher's icon, then click on a pencil icon to take them to a screen where they will be able to edit their	This would allow the user to edit their teacher's information if they changed their name (if they changed their surname due to being newlywed, for example), classroom, or email address,

	teacher name, classroom, and email address.	whilst retaining information associated with that teacher, such as timetable cells which show the teacher and room for each period.
Deleting a teacher.	When the user wants to delete a teacher from their subject, they should be able to click on that teacher's icon, then click on a bin icon to take them to a prompt for the user to confirm that they are sure about deleting that teacher, where they can confirm it to delete the teacher.	The user may no longer be taught by that teacher, so being able to delete a teacher would remove the clutter from that subject from the rest of the application, such as redundant lessons on their timetable. The use of a confirmation prompt will prevent accidental deletions.

Decomposition Table: Agenda

Problem	**Breakdown**	**Justification**
Adding a task to the agenda.	When the user wants to add a new task, they should be able to click on the + button on the agenda screen to add it, classify the type of task (homework/test/other), and give it a title, subject, due date, and additional notes. Once the task has been added, it would be added to the 'Due assignments' list, which would be ordered chronologically, showing the task title, due date, subject, and subject colour.	A graphical user interface would make the menu more accessible for the user to navigate. They need to include the information so that they can remember the important details about the assignment they need to complete.
Editing a task from the agenda.	When the user wants to edit an existing task, they should be able to click on that task, and click on a pencil icon from that screen to edit details of that task such as type of task, title, subject, due date, and additional notes.	This would provide a shortcut to the user, so that they do not have to mark a task as complete, then add the same task with the updated details to the agenda. It will save the user time, and it will be more convenient to the user.
Marking a task as complete.	When the user wants to mark a task as complete, they should be able to tick a checkbox, which will then move the task from the 'Due assignments' list, and to the 'Completed assignments' list.	Structuring the user's agenda into a list of due assignments and completed assignments would reduce clutter to make the agenda easier to read for the user.
Marking a task as incomplete.	When the user changes their mind and wants to mark a task as incomplete, they should be able to navigate to the 'Completed assignments' list and untick the checkbox, which will move that task back from the 'Completed assignments' list to the 'Due assignments' list.	After marking a task as complete, they may realise that they have forgotten to complete a part of that task later on. If the user is able to mark a task as incomplete again, it will save them time as they will not have to re-

		enter their task again unnecessarily.

Decomposition Table: Timetable

Problem	**Breakdown**	**Justification**
Selecting a timetable structure.	When the user wants to start creating their timetable, they should be able to select a template so that the grid fits the timetable of their school lessons. For example, if the user's school divides the lessons into five periods over five days, the user might want to select a 5x5 template.	This would save the user time as the table would not have to be created from scratch. After selecting the structure, all the user would need to do manually is input their own subjects in the correct grid cells.
Entering timetable information.	After the user has selected their timetable template, they should be able to enter their subjects in the relevant cells to complete their timetable. Subjects should be selected through a dropdown menu. Once selected, the cell would be filled with the subject, teacher, room, and a small strip of colour coding.	Using a dropdown menu would save the user time, as it would prevent them from having to repetitively type in the same subjects. By creating this optional timetable, the user would be able to refer back to it if they forgot what lesson they had at a certain time, especially useful for the start of school years where the user may have a different timetable to previous years.
Editing a timetable.	The user should be able to edit their timetable's layout and cells through a graphical user interface after clicking a pencil button on their timetable screen.	This would save the user time as they would be able to make quick changes in the case that their lessons have been permanently rescheduled, as they would not have to delete their timetable and restart it.

Decomposition Table: Grade Book

Problem	**Breakdown**	**Justification**
Adding a new grade.	When the user would like to add a new grade which they achieved, they should be able to click on a + button on the grade book screen, which would take them to a new screen which would allow them to enter their assessment title, grade achieved, subject, date achieved, and additional feedback from that assessment (such as areas for improvement, or particularly good aspects from that work). Once a new grade has been added, it will be added to the grade book, which would order chronologically,	This would allow the user to record grades and display an overview of their grades achieved, so that they can see their progress over time, and which areas they need to improve on. This would encourage improvement for future assessments.

	showing the assessment title, grade achieved, date achieved, subject, and subject colour.	
Editing a grade.	When the user would like to edit a grade they achieved, they should be able to click on a pencil icon from that grade record screen, where they can edit the assessment title, grade achieved, date achieved, subject, and subject colour.	This would save the user time as it would prevent them from having to delete a grade record and re-enter the updated grade with the correct details. The user may have to edit their grade record because their assessment may have been incorrectly marked originally, but then corrected in a later amendment.
Deleting a grade.	When the user would like to delete a record of a grade they achieved, they should be able to click on a bin icon from that grade record screen to take them to a prompt for the user to confirm that they are sure about deleting that grade record, where they can confirm it to delete it.	A grade may have been accidentally by the user, so adding the ability for them to delete their grade records would help the user to clear up any clutter in the application. The use of a confirmation prompt will prevent accidental deletions.

Structure of Problem

The structure of the program can be demonstrated through a layering system, as shown below, where each screen has various processes and decisions which are associated with it. If a process or decision is on the same layer, then it will occur after the process/decision above it. If a process or decision is indented, it will occur if the outdented process/decision has been triggered through a condition. These layers will be explained in greater detail in the next section.

When the user starts the program, they land on the agenda screen, as this is the most important screen in the application, and it will usually be the screen which is used most by the students.

(Colour has been added to the levels of indentation for easier viewing.)

- Agenda (initial landing screen)
    - | Display agenda
    - | Button pressed?
        - | ‘View task’ button pressed?
            - | Display selected task.
            - | Pencil button pressed?
                - | Display task adding screen.
                - | Tick button pressed, but not all required information filled in?
                    - | Display prompt for missing information, and loop back to task adding screen.
                - | Tick button pressed, and all required information filled in?

- | Add task to agenda, and loop back to displaying the agenda.
- | Back button pressed?
  - | Loop back to displaying the agenda.
- | ‘Add task’ button pressed?
  - | Display task adding screen.
  - | Tick button pressed, but not all required information filled in?
    - | Display prompt for missing information, and loop back to task adding screen.
  - | Tick button pressed, and all required information filled in?
- | ‘Completed task’ button pressed?
  - | Move task to completed tasks list, and then continue checking if a button has been pressed.
- | User navigated away?
  - | Exit screen and navigate to the selected screen.

- Timetable
  - | Timetable doesn’t exist?
    - | Display timetable creator interface.
    - | Create timetable button pressed?
      - | Continue to display the timetable.
  - | Timetable exists?
    - | Continue to display the timetable.
  - | Display timetable.
    - | Cell in timetable pressed?
      - | Display cell editing screen.
        - | Delete button pressed?
          - | Delete cell from timetable.
        - | Tick button pressed, but not all required information filled in?
          - | Display prompt for missing information, and loop back to displaying the cell editing screen.
        - | Tick button pressed, and all required information filled in?
          - | Add/update cell to table, and then loop back to display the updated timetable.
    - | ‘Add lesson’ button pressed?
      - | Display cell editing screen.
        - | Delete button pressed?
          - | Delete cell from timetable.
        - | Tick button pressed, but not all required information filled in?
          - | Display prompt for missing information, and loop back to displaying the cell editing screen.
        - | Tick button pressed, and all required information filled in?
          - | Add/update cell to table, and then continue checking if a button has been pressed.
    - | User navigated away?
      - | Exit screen and navigate to the selected screen.

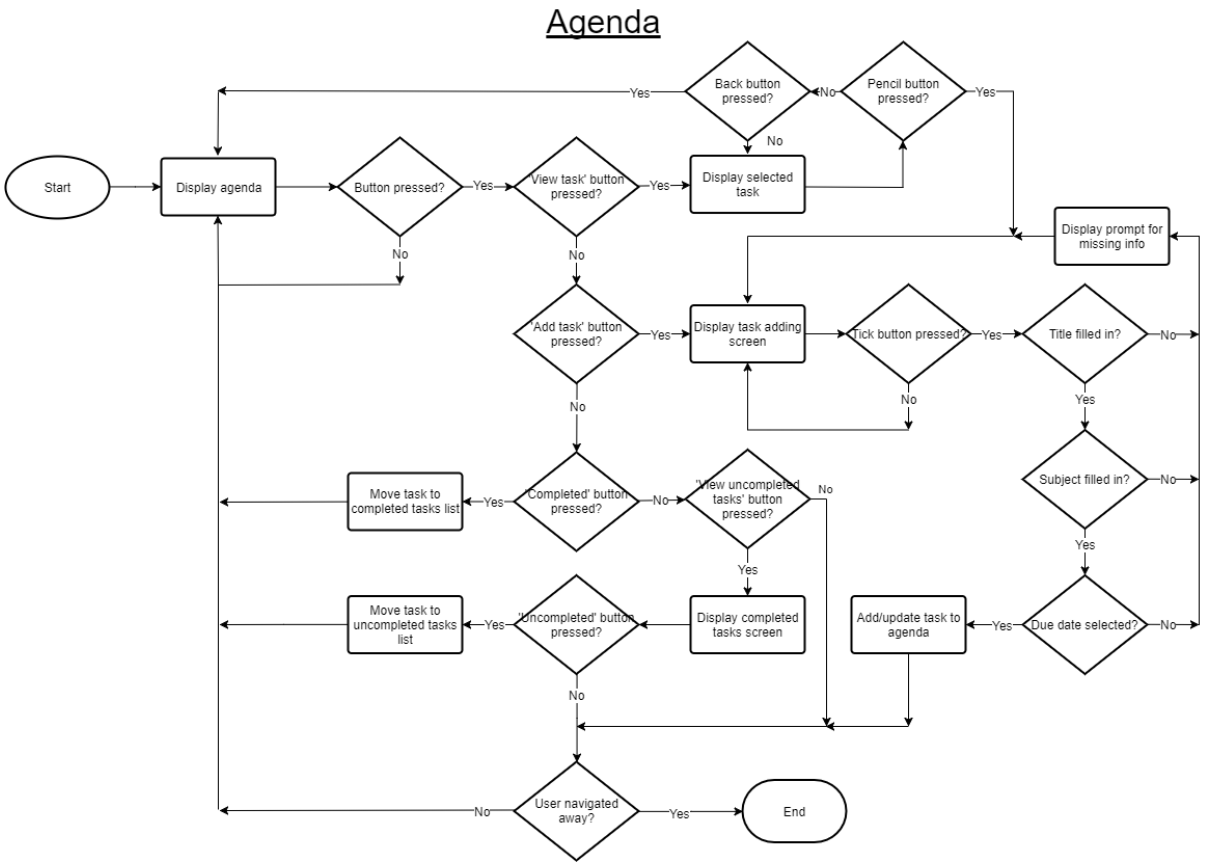
- Grade Book
    - | Display grade book
    - | Button pressed?
        - | ‘View grade’ button pressed?
            - | Display selected grade record.
            - | Pencil button pressed?
                - | Display grade editing screen.
                - | Tick button pressed, but not all required information filled in?
                    - | Display prompt for missing information, and loop back to displaying the grade editing screen.
                - | Tick button pressed, and all required information filled in?
                    - | Update grade record to grade book.
            - | Delete grade button pressed?
                - | Display confirmation prompt, and delete grade record.
        - | ‘Add grade’ button pressed?
            - | Display grade editing screen.
            - | Tick button pressed, but not all required information filled in?
                - | Display prompt for missing information, and loop back to displaying the grade editing screen.
            - | Tick button pressed, and all required information filled in?
                - | Add/update grade record to grade book, and then continue checking if a button has been pressed.
        - | User navigated away?
            - | Exit screen and navigate to the selected screen.

Algorithms

Flowcharts

The algorithms which my application will have to follow have been simplified through the following flowcharts. This will make it easier for me to code the various decisions, processes, and validations which need to occur in my application, as it will break down which of these elements will be needed, when they are needed, and where they are needed.

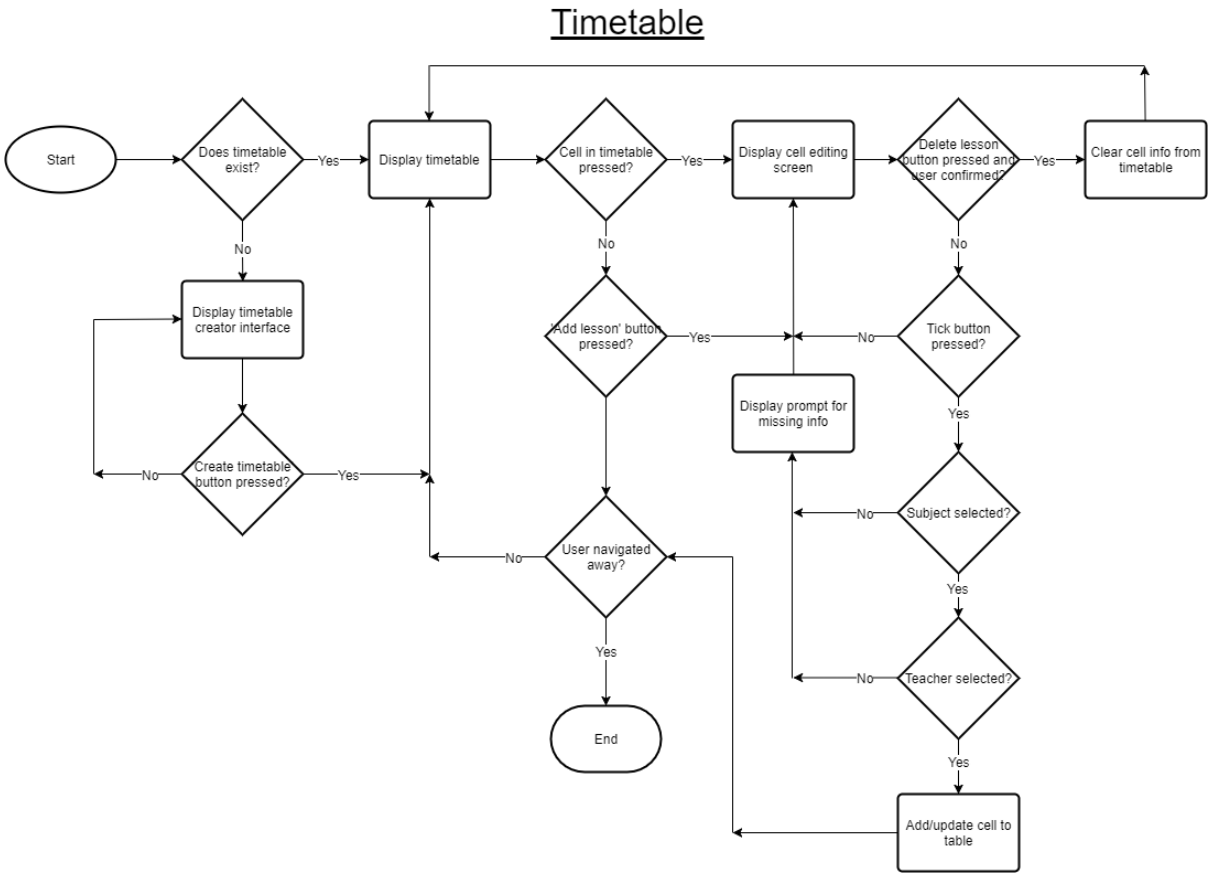
After each flow chart, there is a detailed explanation using the layering system, where each screen has various processes and decisions which are associated with it. If a process or decision is on the same layer, then it will occur after the process/decision above it. If a process or decision is indented, it will occur if the outdented process/decision has been triggered through a condition. These layers will be explained in greater detail in the next section.



(Colour has been added to the levels of indentation for easier viewing.)

- | When the user opens the agenda screen, the program will display the user's current agenda, regardless of whether it is empty, or it is filled up.
- | When the agenda is being displayed, the program is constantly checking whether the 'View task' button, 'Add task' button, 'Completed task' button, or a button in the static navigation interface (to navigate away from the agenda screen) has been pressed.
- | If the 'View task' button is pressed, the program will display the selected task with all the details associated with it, such as the title, subject, due date, and additional notes.
    - | If the user presses the pencil button, the program will enter the task editing screen.
        - | Once the user feels like they have finished making their changes, and press the tick button, the program will check whether required information has been filled in.
        - | If any of them have not been filled in, it will display a prompt to the user to fill in the missing information, and it will take them back to the task editing screen again so that they can make the necessary changes.
        - | If the title, subject, and date due have been selected, the program will add the new task to the agenda, and then go back to displaying the agenda overview.
    - | If not, the program will check whether the user has pressed the back button to go back to the agenda overview screen.
        - | If they have pressed the back button, the program will take the user back to the agenda overview screen.

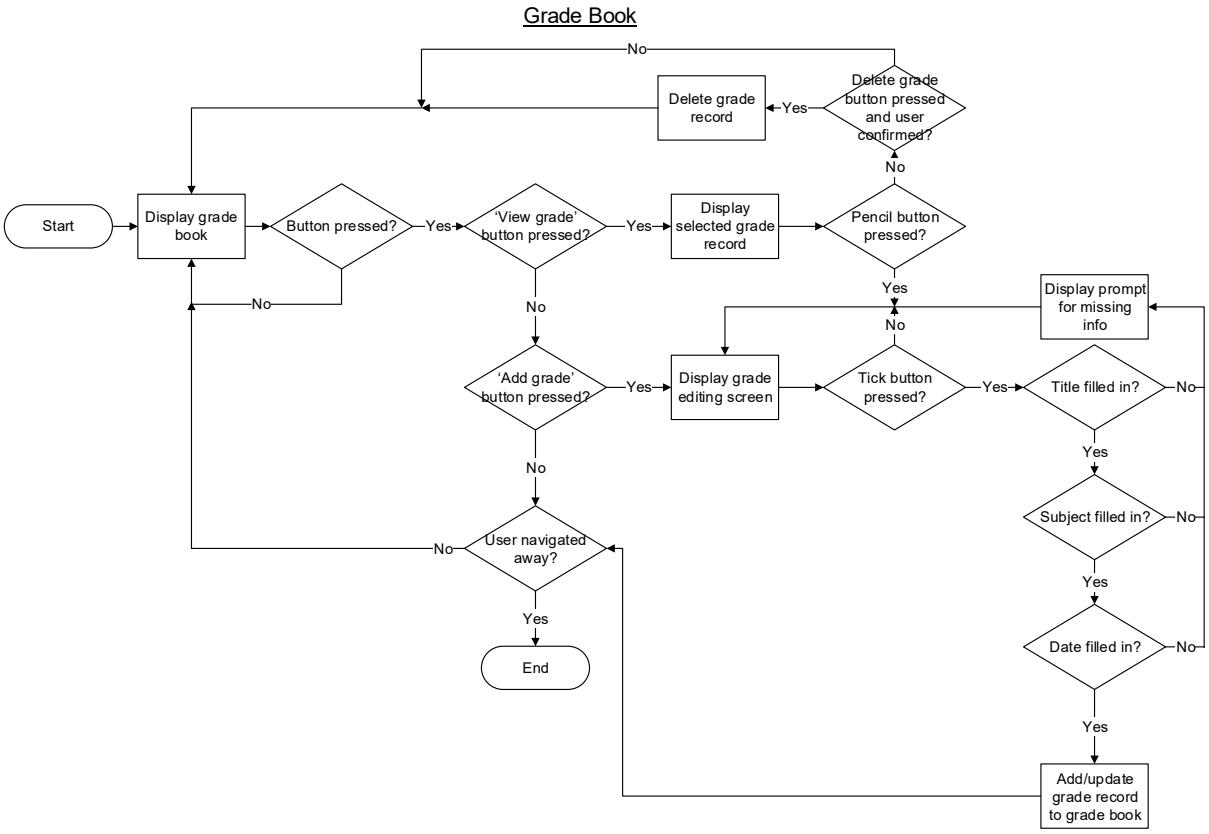
        - | If not, the program will continue to display the selected task.
- | If the ‘Add task’ button has been pressed, the program will display the task editing screen.
    - | Once the user feels like they have finished making their changes, and press the tick button, the program will check whether required information has been filled in.
    - | If any of them have not been filled in, it will display a prompt to the user to fill in the missing information, and it will take them back to the task editing screen again so that they can make the necessary changes.
    - | If the title, subject, and date due have been selected, the program will update the changes to the agenda, and continue checking if a button has been pressed.
- | If the ‘Completed’ button has been pressed, the program will move the task out of the ‘Due tasks’ list (the list which the agenda overview displays), to the ‘Completed tasks’ list.
- | If the user uses the static navigation bar to navigate to a different part of the program, this sequence will end as they will no longer be on the agenda screen.



(Colour has been added to the levels of indentation for easier viewing.)

- | When the user is on the timetable screen, the program checks if the timetable exists.
    - | From there, it displays the timetable if a timetable has already been created (and thus exists), or it displays the timetable creator interface to the user, so that the user can start their new timetable.

        - | Once the user has created their timetable, the program will then display their timetable as well.
- | When the timetable is being displayed, the program is constantly checking whether a cell in the timetable, the ‘Add lesson’ button in the timetable section, or a button in the navigation interface (to navigate away from the timetable section) has been pressed. If none of these buttons have been pressed, the program continues to display the timetable.
- | If a cell in the timetable has been pressed, or the ‘Add lesson’ button has been pressed, the program will display the cell editing screen. This gives the user the option to delete the lesson or edit the lesson if they want, as there will be a bin button (to represent deleting), and a pencil button (to represent editing).
    - | If the user presses the button to delete the lesson, they will be asked whether they would like to confirm their deletion, which will prevent accidental deletions, and if they confirm, it will delete that cell’s information from the timetable. If the user does not confirm, they will be taken back to the cell editing screen.
    - | Otherwise, the user is free to edit the information in that cell using the cell editing screen.
        - | Once they feel they have finished editing that cell, and press the tick button to confirm their changes, the program will check that the required information has been filled in; it will check whether a subject has been selected, and whether a teacher has been selected.
        - | If either of them has not been filled in, it will display a prompt to the user to fill in the missing information, and it will take them back to the cell editing screen again so that they can make the necessary changes.
        - | If both a subject and teacher has been selected, the program will update cell on the table, and continue checking if a button has been pressed.
- | If the user uses the static navigation bar to navigate to a different part of the program, this sequence will end as they will no longer be on the timetable screen.



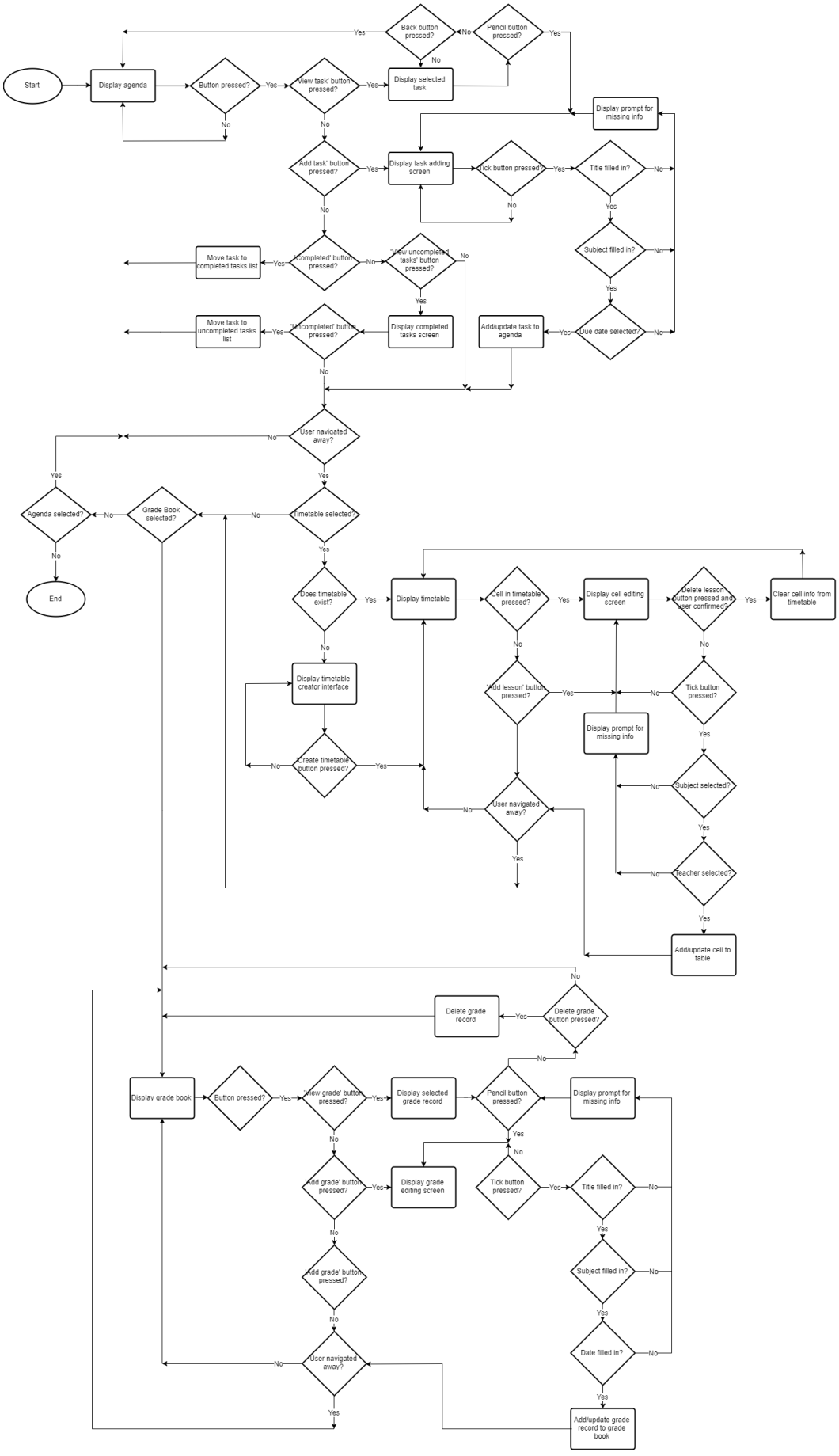
(Colour has been added to the levels of indentation for easier viewing.)

- | When the user is on the grade book screen, the program will display the grade book straight away. It constantly checks if a button is pressed. If a button has not been pressed, it will continue to display the grade book.
- | If the 'View grade' button has been pressed, the program will display the selected grade record.
    - | If the pencil button (for editing) has been pressed, it will display the grade editing screen.
        - | The program waits on the grade editing screen until the user presses the tick button.
            - | Once the tick button has been pressed, the program checks that the title of the assessment, subject, and date of the assessment have been filled in.
            - | If any of these boxes have not been filled in, the program will display a prompt for the missing information to inform them what they are missing, and then take the user back to the grade editing screen.
            - | If all these boxes have been filled in, the grade record will be added to the grade book.
    - | If the delete grade button was pressed instead, the program will delete the grade record after a confirmation screen (to prevent accidental deletions), and then go back to displaying the updated grade book.
- | If the 'Add grade' button has been pressed, the program will display the grade editing screen.
    - | The program waits on the grade editing screen until the user presses the tick button.

        - | Once the tick button has been pressed, the program checks that the title of the assessment, subject, and date of the assessment have been filled in.
        - | If any of these boxes have not been filled in, the program will display a prompt for the missing information to inform them what they are missing, and then take the user back to the grade editing screen.
        - | If all these boxes have been filled in, the grade record will be added to the grade book, and the program will continue to check if a button has been pressed.
- | If the user uses the static navigation bar to navigate to a different part of the program, this sequence will end as they will no longer be on the timetable screen.

These flowcharts can be combined into one master flowchart which shows all the processes occurring in the application, so that the user navigation flow can be planned thoroughly.

Student Planner



Pseudocode

Using my flowcharts, I created some pseudocode to represent how my solution could be developed through programming. It will be used in the development phase to help me structure my programming, and to guide me through the decisions and processes which need to be programmed.

My programming solution will be a lot more advanced than the pseudocode. I avoided using classes in the pseudocode because I feel that it would make it more complicated to refer back to during development, but they will be used in the code for my program so that the code is easier to read and less repetitive.

```
display_agenda = 1
display_timetable = 0
display_gradebook = 0
agenda_pencil_button = 0
grade_book_pencil_button = 0

WHILE display_agenda == 1
    DISPLAY Uncompleted Tasks Overview screen
        WHILE Uncompleted Tasks Overview screen is displayed
            IF any button is pressed
                CHECK what button is pressed
                IF View Task Button is pressed
                    DISPLAY selected task
                    IF pencil button pressed
                        agenda_pencil_button == 1
                    IF back button pressed
                        CONTINUE button check
                IF Add Task Button is pressed or agenda_pencil_button == 1
                    DISPLAY Task Adding screen
                    IF tick button is pressed
                        IF title, subject, and due date are all filled in
                            ADD task to agenda
                            CONTINUE button check
                        ELSE
                            DISPLAY prompt for missing information
                IF Completed Button is pressed
                    MOVE task to Completed Tasks list
                IF View Completed Tasks Button is pressed
                    DISPLAY Completed Tasks Overview screen
                    IF any button is pressed
                        IF Uncompleted Button is pressed
                            MOVE task to Uncompleted Tasks Overview screen
                IF button on navigation menu is pressed
                    IF Timetable Button is pressed
                        display_timetable = 1
                        display_agenda = 0
                    IF Grade Book Button is pressed
                        display_gradebook = 1
                        display_agenda = 0
```

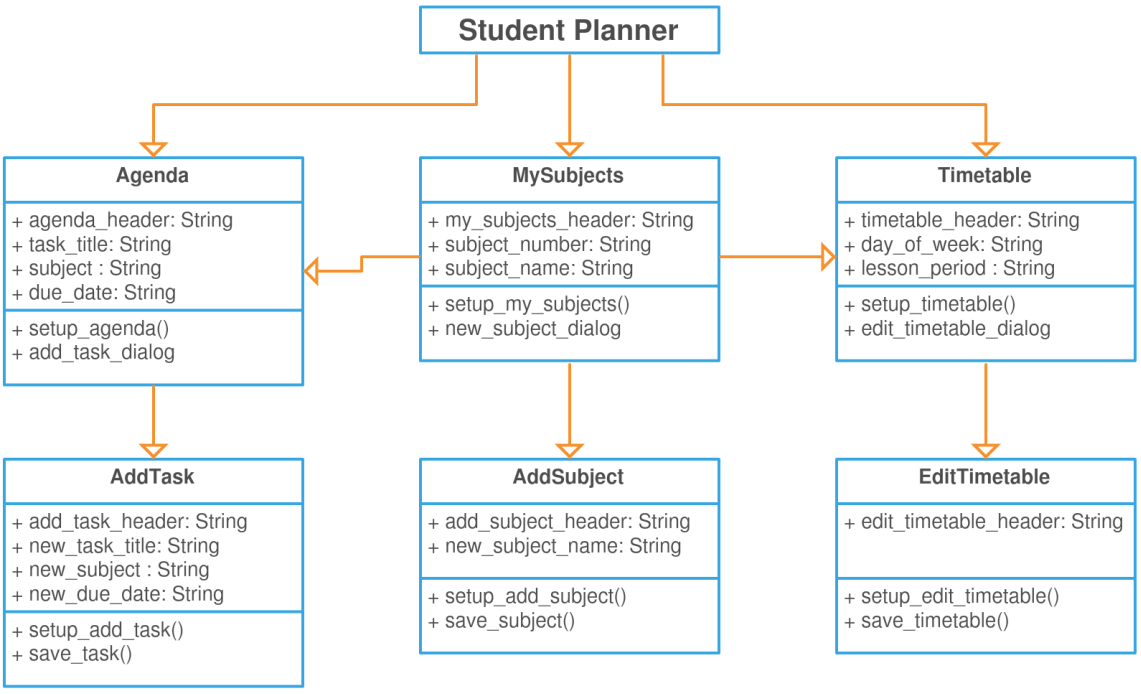
```
40
41
42  WHILE display_timetable == 1
43      CHECK if timetable exists
44      IF timetable does not exist
45          DISPLAY Timetable Creator Interface
46          WHILE Timetable Creator Interface is displayed
47              IF Create Timetable Button is pressed
48                  BREAK from loop
49      DISPLAY Timetable
50      IF cell in timetable is pressed
51          DISPLAY Cell Editing screen
52          IF Delete Lesson Button is pressed and user confirmed
53              CLEAR cell information
54          IF tick button pressed
55              IF subject and teacher are both selected
56                  UPDATE cell to timetable
57              ELSE
58                  DISPLAY prompt for missing information
59      IF Add Lesson Button is pressed
60          DISPLAY Cell Editing screen
61                  IF Delete Lesson Button is pressed and user confirmed
62                      CLEAR cell information
63                  IF tick button pressed
64                      IF subject and teacher are both selected
65                          UPDATE cell to timetable
66                          CONTINUE button check
67                      ELSE
68                          DISPLAY prompt for missing information
69      IF button on navigation menu is pressed
70          IF Agenda Button is pressed
71              display_agenda = 1
72              display_timetable = 0
73          IF Grade Book Button is pressed
74              display_gradebook = 1
75              display_timetable = 0
76
77  while display_gradebook == 1
78      DISPLAY Grade Book
```

```
    CHECK if button is pressed
    IF View Grade Button is pressed
        DISPLAY selected grade record
        IF pencil button is pressed
            grade_book_pencil_button == 1
        IF Delete Grade button is pressed and user confirmed
            DELETE grade record
    IF Add Grade Button is pressed or grade_book_pencil_button == 1
        DISPLAY Grade Editing screen
        IF tick button is pressed
            IF title, subject, and date are all filled in
                ADD grade record to Grade Book
                CONTINUE button check
            ELSE
                DISPLAY prompt for missing information
    IF button on navigation menu is pressed
        IF Agenda Button is pressed
            display_agenda = 1
            display_gradebook = 0
        IF Timetable Button is pressed
            display_timetable = 1
            display_gradebook = 0
```

Class Diagram

Classes are a pillar of object orientated programming, as they enable code to be better organised and more reusable. They are especially important when developing user interfaces, because it makes it easier for the developer to maintain code over time.

To plan some of the attributes and methods of my classes, I have created a class diagram. This will also help me consider when and where I may use class inheritance, which will further streamline my code by encouraging reusability of code.



The diagram shows that the three main classes which will be inherited from are *Agenda*, *MySubjects*, and *Timetable*.

The default screen will be Agenda, but the class *Agenda* still inherits from *MySubjects* to obtain the data about the user's subjects for the dropdown menu functionality when adding new tasks.

Timetable also inherits from *MySubjects* to obtain data about the user's subjects for dropdown menu functionality.

AddTask inherits from *Agenda*, as this dialog window will be an extension of the Agenda main window.

This is the same in the cases of *AddSubject* inheriting from *MySubjects*, and *EditTimetable* inheriting from *Timetable*. All three windows are dialog windows, and they will not function as standalone windows, so they must inherit from *Agenda*, *MySubjects*, and *Timetable* respectively.

Usability Features

Input

- Buttons; used for simple actions which only have a single purpose and need to be performed quickly and conveniently.
    - General navigational interface so that the user can switch between the sections of the application, such as going from the agenda to the timetable.
    - Adding tasks to the agenda, selecting a task to view, editing a task, and saving and confirming edits made to tasks.
    - Selecting timetable templates, adding a lesson to the timetable, selecting a timetable slot to view, editing a timetable slot, deleting a timetable slot, and saving and confirming edits made to timetable slots.
    - Adding a grade record, selecting a grade record to view, editing a grade record, deleting a grade record, and saving and confirming edits made to grade records.
- Tick boxes; used for marking tasks as complete or incomplete, with the ability to quickly untick/re-tick a task if it has been accidentally pressed, as a way of undoing their previous action.
    - Marking tasks in the agenda as complete, or tasks in the 'Completed tasks' section as incomplete.
- Dropdown menus; used for selection of user input from predetermined options to make it easier and quicker to input/change things repetitively.
    - Selecting a subject for a task in the agenda.
    - Selecting a subject and teacher for a timetable slot in the timetable.
    - Selecting a subject and teacher for a grade record in the grade book.
- Calendars; used for convenient selection of dates to make it easier for the user to identify which date they are looking for, such as by providing them the day of the week which corresponds to each date, like a traditional calendar.
    - Selecting a date due for a task in the agenda.
    - Selecting a date achieved for a grade record in the grade book.
- Text boxes; used to give the user flexibility to add any notes/additional information they want to remind themselves of, as opposed to limiting them with other input mediums. May also be used to input details about subjects and teachers to enable the dropdown menu to be used.
    - Notes/additional information about a task in the agenda.

    - Notes/additional information about a grade record in the grade book.
    - Subject names, teacher names, and teachers' classrooms to be used as options in dropdown menus.

Output

- Headers; used to help the user navigate across the application by informing them where in the application they currently are.
    - The overview of each part of the application (Agenda, Timetable, and Grade Book) will have a header to guide the user.
- Sub-headers; used to guide the user within each screen, summarising what information each part of that screen contains and/or the purpose of that part of the screen.
    - Grouping due tasks by date range, such as 'Overdue', 'Due this week', and 'Due next week'.
    - Grouping grade records by month and year, such as 'September 2018', 'October 2018', and 'November 2018'.
- Text labels; used to provide additional information to enhance the understanding of the user, or as hints for what the user should input.
    - Italic labels for the agenda to specify which subject each task is for.
    - Labels next to the inputs for the agenda, timetable, and grade book, such as 'Task title:', 'Teacher', and 'Date achieved:'.
- Icons; used to provide the user with visual information about a button.
    - Three rows of lines to represent a button.
    - Bin buttons to represent deleting a timetable slot/grade record.
    - Pencil buttons to represent performing an edit.
    - Tick buttons to represent saving changes.
- Sound effects; optional audio cues after clicking to provide feedback that the system is processing their click.
    - General click/tap sounds when pressing buttons.

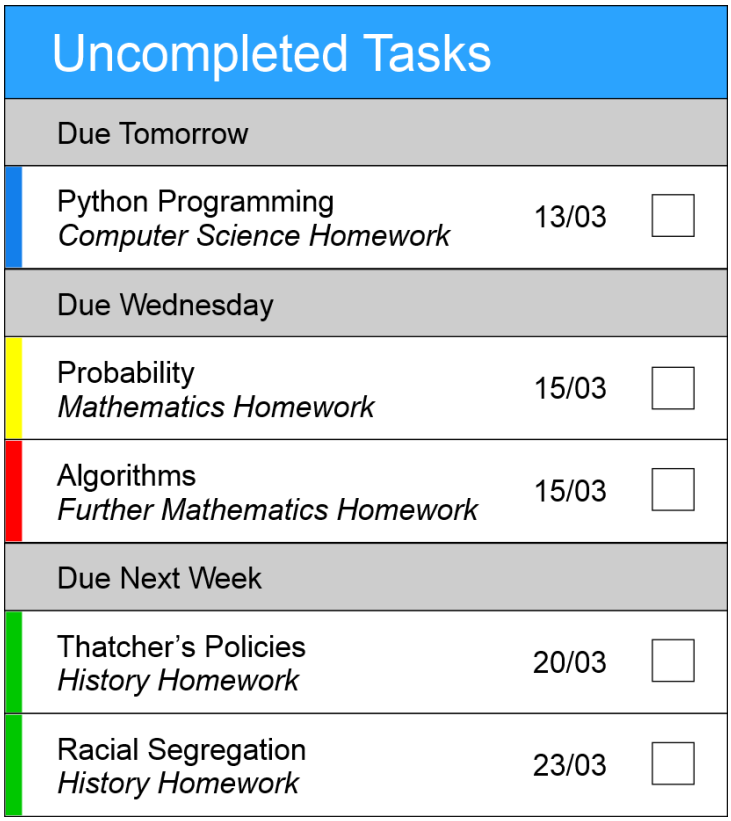
Storage

- Temporary storage; used for storing data and transporting it across a program to be used during the execution of a program.
    - Tracking the list of subjects as it is being updated by the user on My Subjects.
- Permanent storage (data persistence) using .txt files and .json files; used for data which would need to be kept even after the application is closed so that the user can record things such as their tasks, timetable slots, and grades.
    - Agenda would store tasks and their details, such as task title, subject, date due, and additional information. This allows the user to return to their agenda, so they can organise their workload.
    - Timetable would store the student's timetable template and the slots on that timetable, with additional details such as subject title, teacher, and room. This would provide them with a reference in case they forget what lesson they have at a certain time.
    - Grade book would store the student's grade records and the details on those grades, such as assessment title, subject, date achieved, and feedback. This would allow them to reflect on their past assessments, so they know what they need to improve on especially.

Prototype Design

Prototype Design #1: Agenda

Following analysis on the approach of competitors, I created my first prototype of what the agenda on my application could be designed to look like. This is because the agenda would be the main part of the application, as it would be the most important feature of a student planner application, and it would be the first thing developed in my application.



My agenda uses a large header at the top to make it clear to the user that they are viewing uncompleted tasks, as opposed to completed tasks, for example. There are sub-headers such as 'Due Tomorrow', 'Due Wednesday', and 'Due Next Week' to group the tasks by date range so that the list does not appear as one long list which would be difficult to read.

On the left side, there is the colour coding which was seen on the Egenda application. My application would load the user's subjects and corresponding colours which the user would enter in a settings/setup screen, and automatically colour code tasks so that the user can have a better overview of what their workload is for each subject.

For each task, there is a text label to state the task title, and another text label underneath it in italics to specify what subject that task is for, as well as the type of task (homework or test). On the right side, there is the due date (in the DD/MM layout, but this will be something which will be changeable in the settings) and a tick box which the user can tick when they have completed the task. Once the user has ticked a task, it will be moved to the 'Completed tasks' screen, but the task will remain on the screen with the tick box checked until they refresh the screen or revisit it, which would allow the user to untick that box straight away if they accidentally marked it as completed.

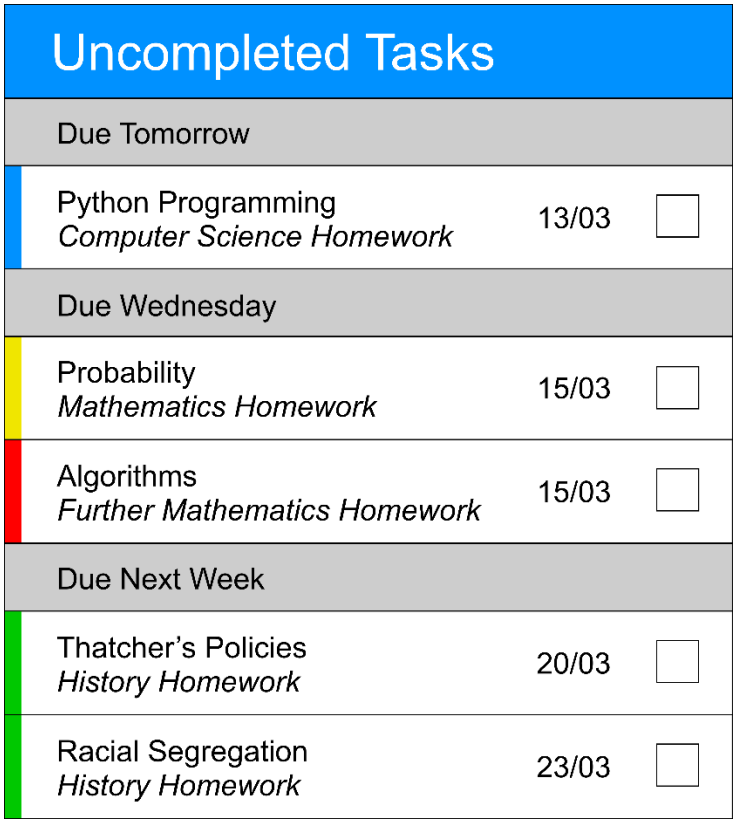
The screen for the completed tasks would be the same, but the header will be changed to ‘Completed Tasks’, and the tick boxes will be ticked by default to add an element of continuity (as they are completed tasks, so they would have been ticked by the user).

On reflection, I am quite happy with how the first design panned out; there is only a couple of minor design changes which could be made to improve the user interface in this first prototype. At the top, the background panel for ‘Uncompleted Tasks’ could be a less harsh colour so that it is more aesthetically pleasing to the user, and less straining on the eyes. When comparing my first prototype design with the designs of competitors, it would also be worth considering reducing the thickness of the outlines for my user interface panels.

Outside of design changes, the alignment of the panels could be improved upon so that there are no white gaps between the colour coding strips and the background, for example.

Prototype Design #2: Agenda

Based on the self-reflection on my first prototype for the agenda screen, I made some slight changes, mostly around making the design more consistent and the colours less harsh.

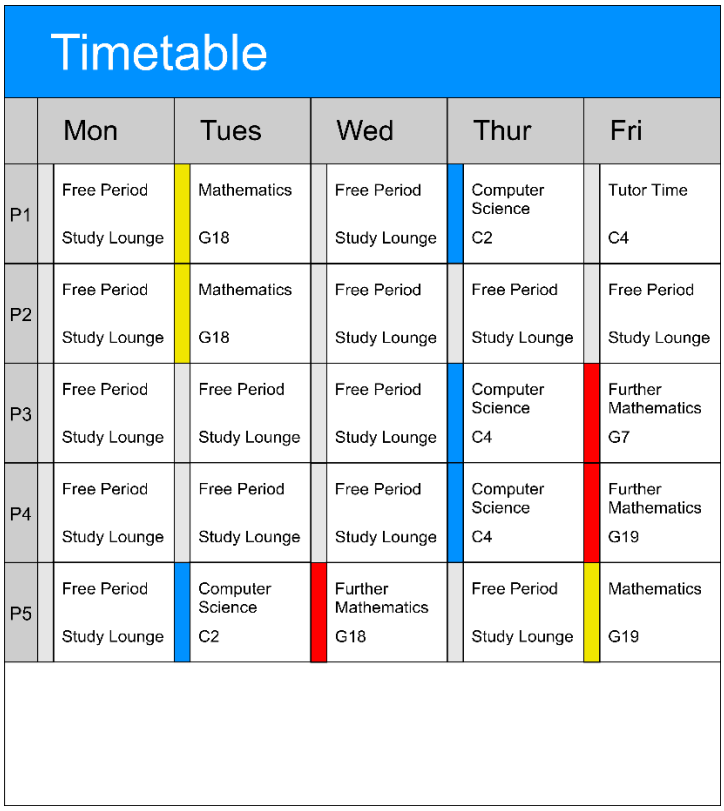


The most obvious changes are the changes in the colour of the ‘Uncompleted Tasks’ panel and the yellow colour code strip for Mathematics. Both were made less bright so that they are less harsh on the user’s eyes, which would make my application more comfortable to use over longer periods.

I also rearranged the user interface elements to avoid misplaced white spaces between the panels. This made the user interface more consistent, making it more aesthetically pleasing, but it also solved the problem of reducing the thickness of the outlines for my user interface panels.

Prototype Design #1: Timetable

One part of my application which most competitors have not included is a school timetable for students. This means that it will be more difficult to create a prototype design for my application's timetable, due to the lack of reference points for comparison.



On this screen, there is a large header on the top to inform the user that they are viewing the timetable. The timetable is presented in a table format, like how timetables are traditionally presented, with sub-headers at the top of the columns to indicate the day of the week, and sub-headers on the left of the rows to indicate the period.

In each cell of the timetable, the subject title and the room for that lesson is displayed next to a colour bar for that corresponding subject.

Prototype Design #1: Grade Book

The grade book in my application would take similar user interface elements from my agenda, but it would remove some aspects to suit the purpose of being a grade book, as opposed to an agenda.

Grade Book		
December 2017		
Computer Architecture *Computer Science*	A*	02/12
January 2018		
Year 12 Winter Mock *Mathematics*	A*	10/01
Year 12 Winter Mock *Further Mathematics*	B	11/01
February 2018		
Year 12 Winter Mock *Computer Science*	A*	01/02
Year 12 Winter Mock *History*	B	02/02

Assessments in the grade book are grouped per month, as this would be a good balance of helping students to differentiate between the date ranges of their assessment, whilst retaining the ability for the user to have an easily accessible list of grades. Grouping the assessments like the agenda (per day) would result in too many sub-headers, making it more difficult for the user to access, as the list would increase in size massively. On the other hand, grouping the assessments per three months would not differentiate the dates enough.

I also removed the tick boxes, as they would not serve any purpose in a grade book. This meant that I moved the date over to the far right. On the left side of the dates, I added the grades, which enables the user to have a quick overview of the grades they have achieved in their assessments (as opposed to being required to open each assessment individually to check).

Apart from the fundamental interface layout changes to accommodate the purpose of being a grade book, I did not make any changes to the design, as I feel it would be unnecessary.

Key Variables and Data Structures

My application will be using a wide range of variables and data structures throughout the various menus and sections, depending on which algorithms are being used. Some variables will also be in constant use in my application, as they may be fundamental for a smooth execution of the code.

It is important that variables names follow the de facto naming conventions for the programming language I will be using, Python. The Python Software Foundation created a Python Enhancement Proposal (PEP 8) guide (https://www.python.org/dev/peps/pep-0008/) which makes recommendations on what syntax developers should use. This makes it easier for developers to read code, so that it can be understood more easily, making feedback easier to give and receive. For example, the guide recommends that if multiple words are

being used in a single variable name, it is best to separate them using an underscore, with all variable names in lower case.

By planning the key variables which I will be using in my program, I will have a better idea of how I could structure my program, which will increase the quality of my code overall. I have created a table below as a plan for my key variables, including the variable name, variable type, and what the purpose of each variable is.

Variable Name	**Data Type**	**Purpose**	**Validation**
display_agenda	Global Integer	Program displays the agenda if this value is '1'.	No validation; this is not an input.
display_timetable	Global Integer	Program displays the timetable if this value is '1'.	No validation; this is not an input.
display_grade_book	Global Integer	Program displays the grade book if this value is '1'.	No validation; this is not an input.
agenda_pencil_button	Local Integer	Program displays the editor for the current task if this value is '1'.	Validation would have already occurred; this is the action.
grade_book_pencil_button	Local Integer	Program displays the editor for the current grade record if this value is '1'.	Validation would have already occurred; this is the action.
agenda_add_button	Local Integer	Program displays the creator for a new task if this value is '1'.	Validation would have already occurred; this is the action.
grade_book_add_button	Local Integer	Program displays the creator for a new grade record if this value is '1'.	Validation would have already occurred; this is the action.
task_title	Local String	Displays the title of the current task, which the user can edit through a textbox.	Length check; maximum of 30 characters long. Presence check; required input.
task_subject	Local String	Displays the subject of the current task, which the user can edit through a dropdown menu.	Length check; maximum of 30 characters long. Presence check; required input.
task_due_date	Local Date	Displays the due date of the current task, which the user can edit through a calendar menu.	Type check; date format. Presence check; required input.
task_notes	Local String	Displays the notes about the current task, which the user can edit through a textbox.	Length check; maximum of 1000 characters long.

lesson_subject_name	Local String	Displays the name of the subject in a lesson, which the user can edit through a dropdown menu.	Length check; maximum of 30 characters long. Presence check; required input.
lesson_teacher	Local String	Displays the name of the teacher in a lesson, which the user can edit through a dropdown menu.	Length check; maximum of 30 characters long. Presence check; required input.
grade_record_title	Local String	Displays the title of the current grade record, which the user can edit through a textbox.	Length check; maximum of 30 characters long. Presence check; required input.
grade_record_subject	Local String	Displays the subject of the current grade record, which the user can edit through a dropdown menu.	Length check; maximum of 30 characters long. Presence check; required input.
grade_record_date	Local Date	Displays the due date of the current grade record, which the user can edit through a calendar menu.	Type check; date format. Presence check; required input.
grade_record_notes	Local String	Displays the notes about the current grade record, which the user can edit through a textbox.	Length check; maximum of 1000 characters long.

Test Data

Test Data for Development

It is important to have a wide variety of test data during development, as it enables a developer to identify potential flaws in their algorithms, which could break the entire program if they are not addressed.

Validation is the primary method of the developer to ensure that their program runs smoothly, so this is the main thing which test data will target during development. There are four types of data to validate; normal, extreme, erroneous, and null. Normal data is acceptable data which is well within the boundaries of what is allowed, and it is useful for checking how common data inputs are handled. Extreme data tests the boundaries of the acceptable data range, and is helpful to test whether data validation is working correctly. Erroneous data is unacceptable, and should cause the validation algorithms to trigger an error message; it is useful for testing what actions the program takes when an incorrect input is used.

To ensure that validation is tested thoroughly and systematically, I have created a table to show what type of data should generally be entered for testing:

Input	**Nature of Data**	**Data Validation**	**Is Data Accepted?**	**Output**

	Normal			
	Extreme			
	Erroneous			
	Null			

Examples of tests which I may perform for adding a subject are shown below. The last two columns have not been filled in, as it would depend on how successful my program was at that stage.

Input	**Nature of Data**	**Data Validation**	**Is Data Accepted?**	**Output**
Typed 'Computer Science' and saved the subject.	Normal	Length check, presence check	?	?
Typed 'World Philosophy and Ethics' and saved the subject.	Extreme	Length check, presence check	?	?
Typed 'Computer Science and Further Mathematics' and saved the subject.	Erroneous	Length check, presence check	?	?
Typed nothing and saved the subject.	Null	Presence check	?	?

I have also created a test table for more general cases, where the data may not be the primary focus, to test things such as the accessibility of the user interface.

Test	**Expected Result**	**Outcome**	**Further Actions**

Examples of general tests which I may perform for My Subjects have been shown below in the table. The outcome and further actions columns have not been filled in, as they would depend on how successful my program was at that stage.

Test	**Expected Result**	**Outcome**	**Further Actions**
Is there a functional button for the user to add new subjects?			
Does adding a new subject update the list			

in My Subjects?			
Is the list of subjects ordered in alphabetical ascending order?			

Test Data for Beta Testing

Before my program can be considered ready for public release, it must be beta-tested. In the beta testing phase, my student planner application will be subjected to real world testing by the target audience of my application (students). The testers will provide feedback on how well various features work, and comments to suggest what improvements could be made to the user experience.

From there, I will be able to make changes in later development iterations, which will gradually improve the quality of my programs as more beta testing is performed.

The beta testing will be continued on from in the evaluation section, where I will be performing user acceptance testing, which will be for the final version of my application as opposed to beta testing during the development of my application.

I have created a plan with sets of data profiles which the testers can use, stating the intended outcome from my application when these data profiles are used.

Examples of test profiles which may be used for beta testing Agenda have been shown below.

Task Profile 1: Normal Data
- Task Title: Probability
- Subject: Mathematics
- Due Date: 13/05/2019
- Notes: Complete the task sheet on conditional probability.

This profile should be accepted, as all data is entered correctly, meaning it should pass validation.

Task Profile 2: Extreme Data
- Task Title: Conditional Probability, Tests
- Subject: Mathematics
- Due Date: 16/05/2018
- Notes: Complete the task sheet on conditional probability and tests.

This profile should be accepted, as all data is entered correctly, although they are on the boundaries of length validation for the task title limit of 30 characters.

Task Profile 3: Erroneous Data
- Task Title: Binomial Distribution, Discrete Random Variables, Poisson Distribution
- Subject: Further Mathematics
- Due Date: 19/05/2018
- Notes: Complete the questions on binomial distribution, discrete random variables, and Poisson distribution.

This profile should be rejected, as the task title is beyond 30 characters, which should cause a length validation error.

Task Profile 4: Null Data

- Task Title:
- Subject:
- Due Date:
- Notes:

This profile should be rejected, as the required information (task title, subject, and due date) have not been entered, which should cause a presence validation error.

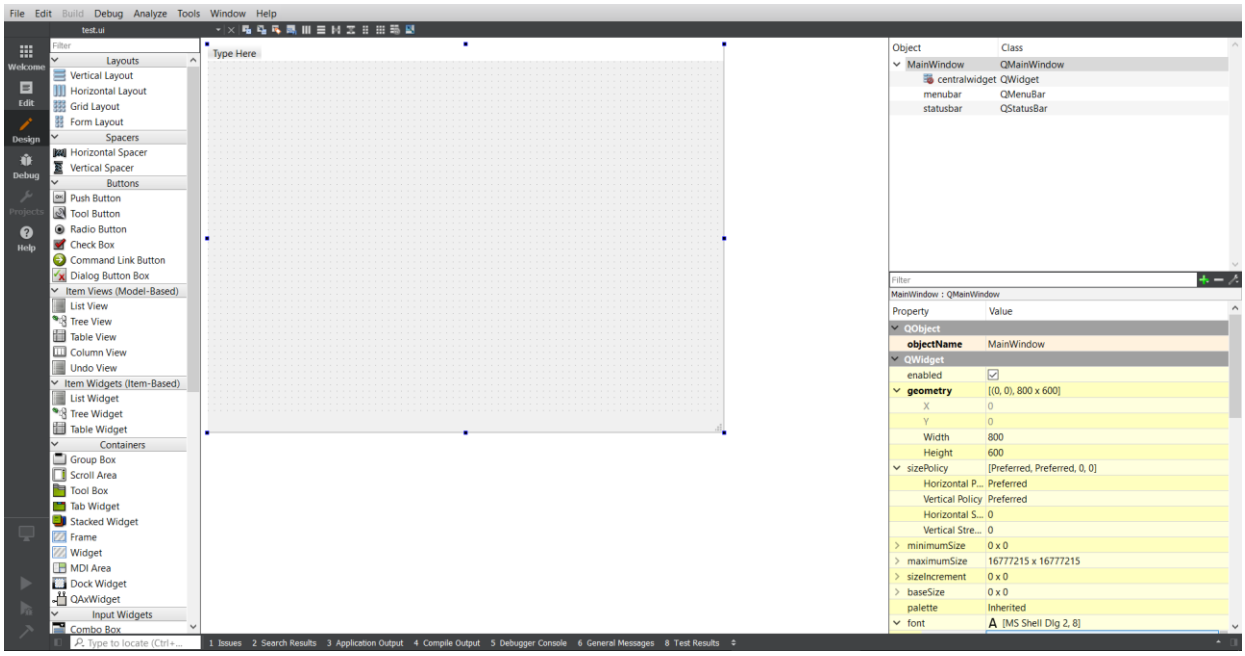
Development

Preamble

For the development of my program, I have chosen to use Python as the programming language, with the PyQt5 plugin as a Python binding of the graphical user interface toolkit, Qt.

I chose this combination because I know the Python programming language the best out of the mainstream programming languages, and PyQt is widely accepted to be the best graphical user interface library for creating a highly customisable interface which will look native on any mainstream operating system.

PyQt also has the option of using Qt Creator to create graphical user interfaces more easily; it enables you to design your user interface using a WYSIWYG (what you see is what you get) designer program, which makes for quicker and more precise development, as I will be able to see my interface in real time as it gets updated, as opposed to being required to run the application to view my user interface during development.

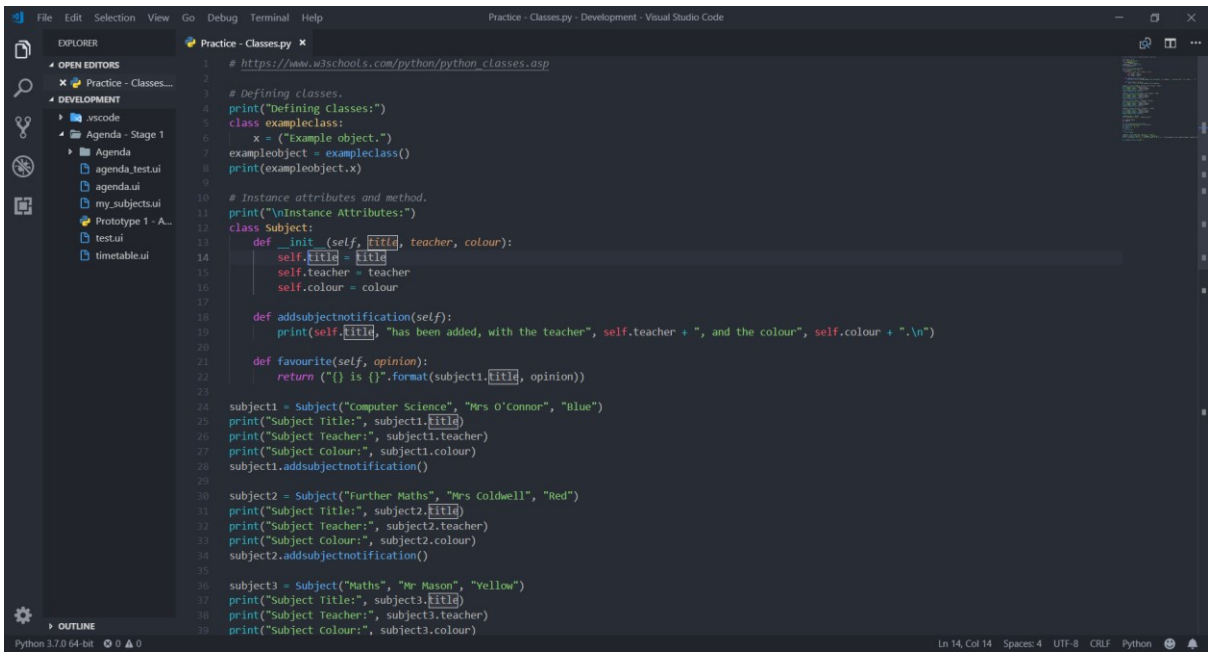


Once I had chosen to use PyQt5 as my Python library for the GUI, I performed some research to learn about how Qt Creator works so that the learning curve for creating my user interfaces wouldn't be as steep once I started actual development of my screens. I found this YouTube video (https://www.youtube.com/watch?v=GLqrzLIIW2E) particularly helpful for teaching me the basics.

The pseudocode which I have created in the design phase will be used as a guide for how I will be creating this program. However, the code demonstrated in these prototypes will be more detailed and more efficient, as the pseudocode was only designed to give a rough overview of how parts of my program could be coded. In practice, it is likely that my program will have code which has a very different structure to the pseudocode, largely due to how Qt Creator will create user interfaces in .ui files (which are coded using XML), and then convert it to Python through some command lines.

After I had initially started to experiment with using Qt Creator to design my user interfaces, I learned that it would be a better idea to change some of the design philosophies which I had in mind. Qt Creator is designed to create user interfaces which will appear native to the operating system the application is running on, and in this case, my application will be running on Windows. Adapting my design philosophy to accommodate this discovery means that the user interfaces I develop will likely veer away from some aspects of my prototype designs, such as the panel designs behind the elements of the interface, but my application will continue to follow the specification, as it will still take some inspiration from Google Material Design guidelines in areas such as spacing of elements for improved readability. This is not an issue for me, because Google's Material Design guidelines encourage UX (user experience) design which is already considered to be good practice; in many cases, it is merely reinforcing guidelines of common practice.

For my IDE (integrated development environment), I have decided to use Visual Studio Code because it is the IDE which I have become accustomed to using, as the multi-language support would enable me to use it for other languages such as Java if I needed to. The editor supports a wide range of features such as line numbering, refactoring (to change all cases where a variable name was used when the name is changed, which saves time), colour coding of different types of code, and debugging.



Development will be split into separate stages, each representing the development of a different part of my student planner.

At the start of each stage, I will explain what my objectives are, expanding more on the stage title. This will provide me with a better plan about what I should be doing in each stage, and it will provide a comparison point for review at the end of the stage, as I will be able to compare my targets to the tangible results in that stage.

Next, there will be a prototype program created, which will either form part of the development later in that stage, or possibly form a part of a future development.

Then, the first development iteration will begin, explaining what I have coded in that stage with screenshots, text annotations, and comments within the code itself.

This is followed by a testing phase for that iteration, which will involve beta testing to test different types of inputs, and a more general test to check that the developments I made were meeting the objectives at the start of that stage. If the first development iteration was not fully successful, another development and testing iteration will be held, and so on.

After the completed code, there will be a section on reviewing that stage, providing a summary on how I succeeded my targets from that stage, or explaining why development did not go as planned if it was unsuccessful.

Finally, at the end of each stage, the next steps for that part of the program will be stated, which will provide a starting point for future development stages.

Throughout the development stage, I will be using *italic* notation when referring directly to *coding terms* or *file names* for easier reading.

As a reminder, the success criteria for my application is as follows:

- The user must be able to enter their subjects and assign them a colour, which should both be stored in data files. This is because these subjects will be needed for the dropdown menu when adding a task to the agenda, and the colours will be displayed alongside the task on the agenda to help the user distinguish between the tasks from various subjects.
- The user must be able to add a task to the agenda with a title, subject, due date, and additional notes, which should all be stored in data files. This is so that the user can view a list of their tasks, and expand tasks from this list to view details about the task to help organise their workload.
- The user must be able to select the subject which a task is delegated for through a dropdown menu. This is to improve the user experience, allowing the user to create tasks more quickly and with less mistakes.
- The user must be able to select a timetable template through a user interface, and enter subjects in individual cells of the table, which would be stored to data files. This would allow the user to create a timetable that shows the lessons the user will have on each day, on a period by period basis, to help the user arrive to the correct lessons on time.
- *(Optionally, depending on time constraints)* The user must be able to enter the grades which they achieve in a subject to a grade book, with the title of the topic, grade, date achieved, and feedback, which would be stored to data files. This would allow the user to access an overview of their grades at another time, so that they can see which topics they need to improve at.

Stage 1: Agenda

Objectives

As this is the first stage, the most important objective is that I should have a user interface which allows the user to view information about their tasks, as the agenda is the most important part of my program. The agenda should be created so that it can be read easily, and contains an overview on the key information of their tasks, such as task title, subject, due date, and whether it is completed.

In this stage, functionality is not an objective, as this will be something which will be developed in a later stage. This stage is simply about developing a user interface to act as a foundation for functionality in future development.

Prototype Program

Before I started the more formal development process for this stage, I quickly created some prototype code which could be used in the agenda either in this stage, or at a later point.

One key feature of the agenda is that the user must be able to add a new task, as mentioned in the second part of the success criteria. Based on this, I decided to create some code which would allow the user to enter the details of their task, have these details validated, and then display a notification as feedback to let the user know that their task has successfully been added to the agenda.

```
1  # Defines a class for adding a new task to the agenda.
2  class NewTask:
3      def __init__(self, title, subject, due_date, notes):
4          self.title = task_title
5          self.subject = task_subject
6          self.due_date = task_due_date
7          self.notes = task_notes
8
9      def new_task_notification(self):
10         print("\nA new task for", self.subject, "has been added to the agenda.\n")
11
12 # Validates the entries of task details.
13 task_title = str(input("Title: "))
14 while len(task_title) > 30:
15     print("\nThis title is too long.\n")
16     task_title = str(input("Title: "))
17
18 task_subject = str(input("Subject: "))
19 while len(task_subject) > 30:
20     print("\nThis subject name is too long.\n")
21     task_subject = str(input("Subject: "))
22
23 task_due_date = str(input("Due Date: "))
24 while len(task_due_date) > 10:
25     print("\nThis due date is too long.\n")
26     task_due_date = str(input("Due Date: "))
27
28 task_notes = str(input("Notes: "))
29 while len(task_notes) > 1000:
30     print("\nThese notes are too long.")
31     task_notes = str(input("Notes: "))
32
33 # Adds a new task to the agenda using the user inputs, and creates a notification once it has been added.
34 task1 = NewTask(task_title, task_subject, task_due_date, task_notes)
35 task1.new_task_notification()
```

In this program, I used a class to handle the process of creating a new task based on the user's inputs for the task title, subject, due date, and notes. Classes are a part of object orientated programming, and there are several advantages to using object orientated programming (https://stackoverflow.com/questions/33072570/when-should-i-be-using-classes-in-python), with the most significant reasons in my case being organisation of code, defining and keep track of state, and reusability of code. In other occasions where I would like my code to do a specific command/action, and not have multiple commands/actions associated with the objects within that code, it might be better for me to use functions, as explained by this Stack Overflow thread (https://stackoverflow.com/questions/18202818/classes-vs-functions).

From line 1, I initialised the class *NewTask* with attributes *title*, *subject*, *due_date*, and *notes*, as these are the details which the user would be adding to each new task. Then, I created a method to display a notification as feedback for the user that their new task has successfully been added.

The next part of my program from line 11 enables the user to input their task details. For each detail, once they have entered something, the program validates that their input does not exceed the character limit for each task, as mentioned in the earlier section about key variables and data structures. Each part of the task must pass validation, as the user will be continuously prompted for a valid version of each attribute until they provide one which is valid. For this prototype, I simply used length validations for all attributes, but it is likely that my program will only need these length validations for task title and notes in reality, as the subject could be selected through a dropdown menu, and the due date could be selected through a calendar interface.

Lines 32 onwards add the task to the agenda using the validated inputs from the user. Then, it calls for the new task notification, which will confirm to the user that the new task for their subject has been successfully added to the agenda.

```
Title: Python Programming
Subject: Computer Science
Date Due: 13/03
Notes: Complete programming practice using the tasks given in class.

A new task for Computer Science has been added to the agenda.
```

An example of this prototype working when the user inputs all the attributes correctly is shown above. They enter the task details, the task is added, and a notification is shown to confirm that the task has been added.

```
Title: Python Programming sddfkaskfdaspodkasdkopasdkasopdkpasdkaspdapdkopaskodpasopdkaspodkoaspkdopaskopkdopkdopaskd

This title is too long.

Title: Python Programming jasiodjojasdioasjdiojasiodasjdioasjdioasjdioasdioasdiojasiodjas

This title is too long.

Title: Python Programming
Subject: Computer Science isadofaspdjkoaspdjiasojdopjojpasjopdjasopjdpoajsopdjaspojopjopasopdjoap

This subject name is too long.

Subject: Computer Science
Due Date: 13/03
Notes: Complete programming practice using the tasks given.

A new task for Computer Science has been added to the agenda.
```

The example above is a demonstration of the program working when the user inputs task details incorrectly, with the initial inputs for the task title and subject being too long. In each case, they are continuously asked for another input until they input something which is within the length requirement.

Development: Iteration #1 – User Interface

Development for my application will start with the GUI for the agenda, as this is the main part of my application; the grade book and timetable have lower priority, as they are extensions of my program which could be considered as extra parts.

Starting with the agenda, I set the default window resolution to 960x720, which will also be the default window. This is because the 4:3 aspect ratio provides a good balance between width and height, meaning I will have more flexibility to lay the elements of my user interface on the screen. I have considered the various devices my audience of students will be using to access my application, and almost all modern displays have a resolution of at least 720p, whether it be on a monitor connected to a desktop computer, a laptop, or a two-in-one tablet computer, which is why I chose this type of 4:3 resolution. Despite this, my program will be

resizable to adapt to the user's needs, and I will try to design the elements in my program to adapt shape and size when the user resizes the window.

For the font, I chose Arial because it is a sans-serif typeface, meaning that it is easier to read and recognise, especially for a younger audience who have grown up to be more accustomed to sans-serif fonts. This was backed up by my research, as these points were mentioned in this article (http://www.scribe.com.au/tip-w017.html), which discusses the differences between serif and sans serif fonts.

As an agenda will likely be filled with many tasks, a lot of space in the window will be used, and when the window becomes filled with tasks, there will need to be a way for the user to access the remaining tasks. There are two options here; I could design the application to work in a page layout, where the user will be able to go onto the next page to access the other tasks, or I could design the application to be scrollable, where the user will be able to use a scroll bar, arrow keys, or a mouse scroll wheel to move the page down.

To help me make this decision, I performed some additional research on the internet. This led me to an article (https://uxplanet.org/ux-infinite-scrolling-vs-pagination-1030d29376f1) which thoroughly discussed the advantages and disadvantages. To sum up, it depends on the use case, and I have chosen to go with scrolling over pagination because it is the better choice for user engagement, faster for the user to navigate, and easier to navigate on mobile devices such as two-in-one tablet computers, which is one category of device which students may be using to access my application.

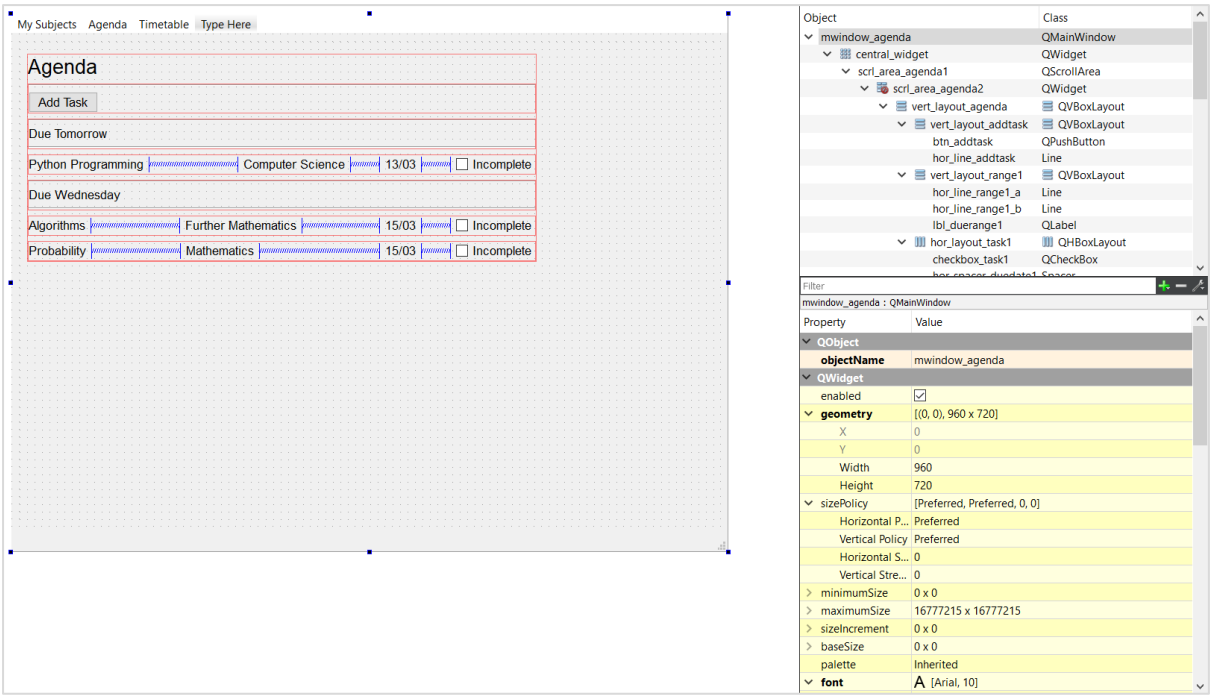
Implementing scrollable content for this agenda screen is easy, as I used a scroll area widget for which all elements will be built on, as this enables all content in the scroll area to be scrollable.

To structure the main elements of this screen, I used a vertical layout, as the agenda will have a largely vertical nature, with tasks being stacked above each other in a table layout.

A basic navigation menu for the user to navigate to different parts of the student planner has been created at the top of the window using Qt Creator's built-in menu creator. To start with, I only added buttons for My Subjects, Agenda, and Timetable, because these will be the three parts of my application which I will focus on; other parts of the application will be developed if time constraints are not an issue at that point.

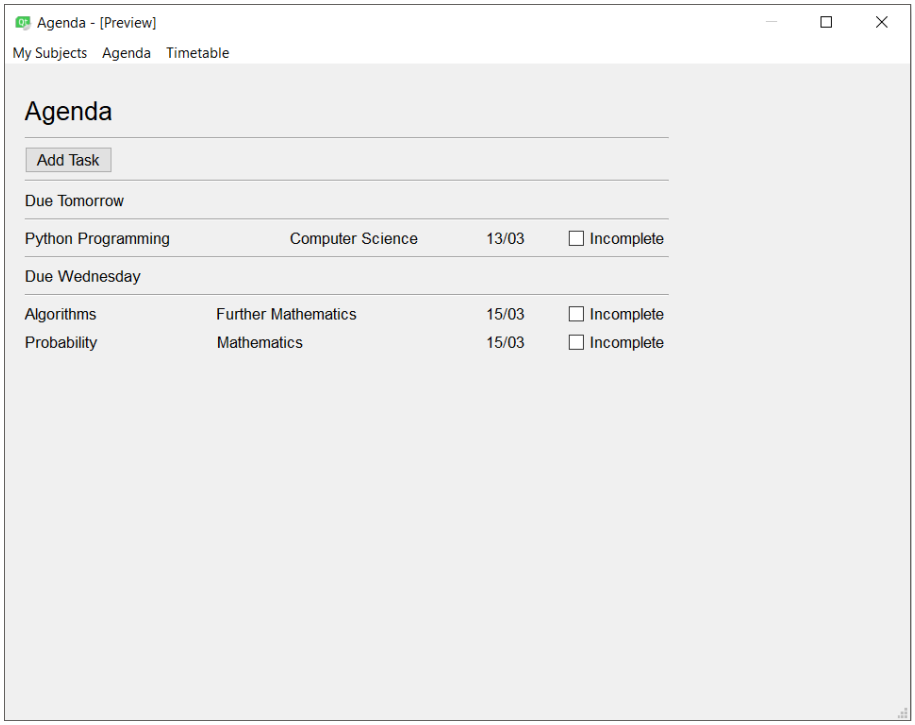
I created a label for the header text reading 'Agenda' in a larger font size, which helps the user to recognise which part of the application they are currently on for ease of navigation. Below that, I added a horizontal line for aesthetics and spacing, followed by an 'Add Task' button to provide the user a way to add new tasks. This was positioned above another horizontal line to separate it from the list of tasks below.

The next part of the user interface was a polarising decision. At first, I created elements for tasks with a horizontal layout for each task, each including the task title, subject, due date, and a completion checkbox. Horizontal spacers were added between task details as part of Google Material Design guidelines, which mention that elements should be spaced out appropriately to avoid a cluttered aesthetic. Tasks were grouped by date range, as seen in my prototype design.



Once I had finished that part of the user interface, I decided that it would be a good point to test it, because any additional features would take a longer time to implement, or they would be unnecessary to add at this stage.

Test: Iteration #1 – User Interface



As I had not programmed the buttons or checkboxes to perform any actions at this point, I did not expect anything to occur when I clicked on these buttons or checkboxes. This turned out to be true; there was the animation effect when I clicked these buttons, but nothing happened after this.

Once I had completed this first development iteration, I performed tests focused around the objectives set for this stage, and how well these were met.

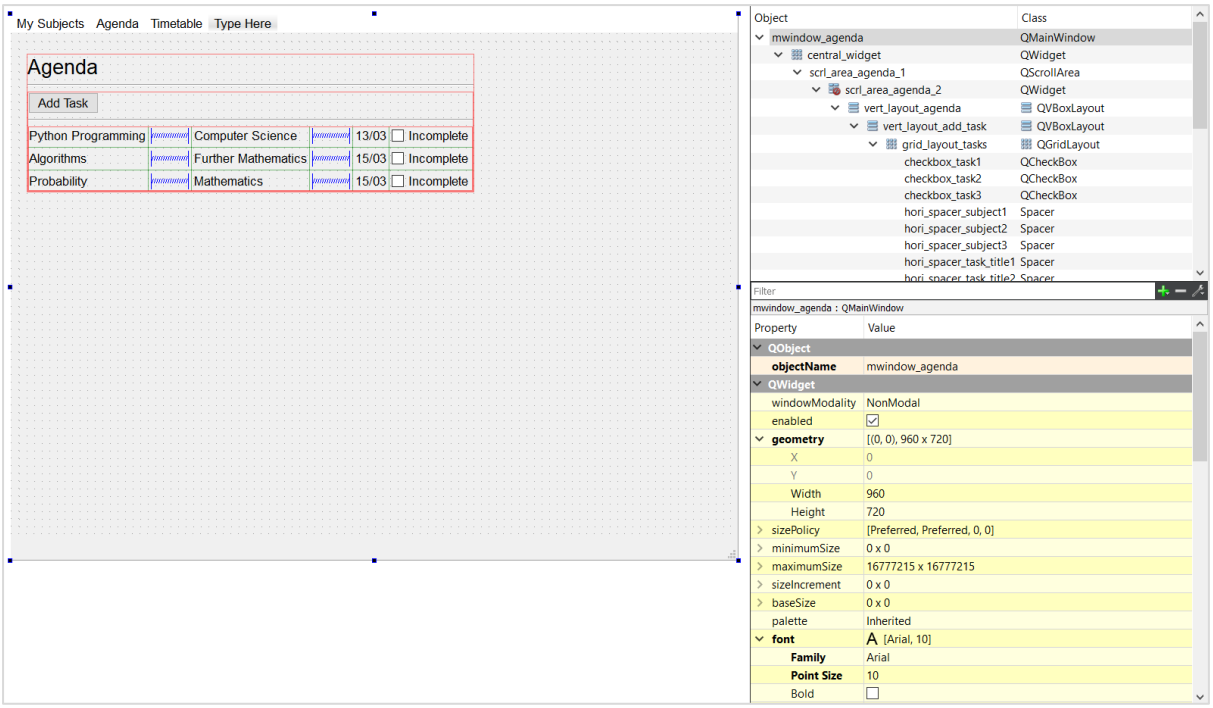
Test	Expected Result	Outcome	Further Actions
Is the user able to access an overview of their tasks?	User should be able to read an overview of key information about their tasks.	User is able to read an overview of key information about their tasks, including task title, subject, due date, and whether it is completed.	N/A.
Is the user able to easily read the agenda?	User should be able to read agenda easily, with data which are related aligned to indicate their relation.	User can differentiate between the individual tasks easily, but details are vertically misaligned.	Vertically align the details of the task for clearer distinction about what type of information is being displayed (task title, subject, due date, or completion).

When I previewed the user interface, I realised that it was a poor user interface for readability; the varying character lengths of task titles and subject names were causing the task details to be misaligned, even after using horizontal spacers. I experimented with different properties for the spacers, including using expanding size spacers (spacing will expand as far as it can), minimum size spacers (spacing will be a minimum of the size set), and maximum size spacers (spacing will be a maximum of the size set). In all cases, the issue persisted, resulting in misaligned subject names between different tasks.

Not all objectives were met, so an additional development iteration for this stage was required to address the readability issue with Agenda.

Development: Iteration #2 – UI Improvements

To fix this issue, I experimented using a grid layout for the tasks in my agenda, as this is a layout which is more typically used to structure data in a table format. This should solve my issue, as each column should expand its width to the maximum required to fit the content in the grid, with all tasks sharing the same grid. There may be different solutions for my issue, but this appears to be the simplest solution possible, as the size of columns is automatically determined and adjusted to suit the content, which makes for easier design. The only downside to this change is that it required me to remove the grouping of tasks by date range, but this may not be a necessary feature, and the issue may be assessed at a later time.



To convert this window I created in Qt Creator from a .ui file to a .py file, as mentioned in this article (http://projects.skylogic.ca/blog/how-to-install-pyqt5-and-build-your-first-gui-in-python-3-4/), I went into the command prompt and set the file directory to where I saved the user interface file using the command 'pushd' followed by the file directory. Then, I used the command *pyuic5 -x agenda.ui -o agenda.py*, which reads *agenda.ui* and outputs *agenda.py* in the same file directory. This process will need to be repeated at the end of every stage.

For reference, the commands I used have been shown below. This will not be shown in later stages, as the commands are the same every time, but with different file names. For example, I would change *agenda.ui* and *agenda.py* to *my_subjects.ui* and *my_subjects.py* for the next stage.

```
C:\Users\isaac>pushd C:\Users\isaac\OneDrive - Stanborough School\Documents\School Work\A-Level\Computer Science\Program
ming Project\Project\Development\Stage 1 - Agenda

C:\Users\isaac\OneDrive - Stanborough School\Documents\School Work\A-Level\Computer Science\Programming Project\Project\
Development\Stage 1 - Agenda>pyuic5 -x agenda.ui -o agenda.py
```

This created the code for my user interface, as shown below in my IDE:

```python
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'agenda.ui'
#
# Created by: PyQt5 UI code generator 5.11.3
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_mwindow_agenda(object):
    def setupUi(self, mwindow_agenda):
        mwindow_agenda.setObjectName("mwindow_agenda")
        mwindow_agenda.resize(960, 720)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        mwindow_agenda.setFont(font)
        self.central_widget = QtWidgets.QWidget(mwindow_agenda)
        self.central_widget.setObjectName("central_widget")
        self.gridLayout = QtWidgets.QGridLayout(self.central_widget)
        self.gridLayout.setObjectName("gridLayout")
        self.scrl_area_agenda_1 = QtWidgets.QScrollArea(self.central_widget)
        self.scrl_area_agenda_1.setFrameShape(QtWidgets.QFrame.NoFrame)
        self.scrl_area_agenda_1.setWidgetResizable(True)
        self.scrl_area_agenda_1.setObjectName("scrl_area_agenda_1")
        self.scrl_area_agenda_2 = QtWidgets.QWidget()
        self.scrl_area_agenda_2.setGeometry(QtCore.QRect(0, 0, 938, 648))
        self.scrl_area_agenda_2.setObjectName("scrl_area_agenda_2")
        self.layoutWidget = QtWidgets.QWidget(self.scrl_area_agenda_2)
        self.layoutWidget.setGeometry(QtCore.QRect(10, 16, 591, 183))
        self.layoutWidget.setObjectName("layoutWidget")
        self.vert_layout_agenda = QtWidgets.QVBoxLayout(self.layoutWidget)
        self.vert_layout_agenda.setContentsMargins(0, 0, 0, 0)
        self.vert_layout_agenda.setObjectName("vert_layout_agenda")
        self.lbl_agenda = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setPointSize(16)
        self.lbl_agenda.setFont(font)
        self.lbl_agenda.setObjectName("lbl_agenda")
        self.vert_layout_agenda.addWidget(self.lbl_agenda)
        self.hori_line_agenda = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_agenda.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_agenda.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_agenda.setObjectName("hori_line_agenda")
        self.vert_layout_agenda.addWidget(self.hori_line_agenda)
        self.vert_layout_add_task = QtWidgets.QVBoxLayout()
        self.vert_layout_add_task.setObjectName("vert_layout_add_task")
        self.btn_add_task = QtWidgets.QPushButton(self.layoutWidget)
        self.btn_add_task.setObjectName("btn_add_task")
        self.vert_layout_add_task.addWidget(self.btn_add_task, 0, QtCore.Qt.AlignLeft)
        self.hori_line_add_task = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_add_task.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_add_task.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_add_task.setObjectName("hori_line_add_task")
        self.vert_layout_add_task.addWidget(self.hori_line_add_task)
        self.grid_layout_tasks = QtWidgets.QGridLayout()
        self.grid_layout_tasks.setObjectName("grid_layout_tasks")
        self.checkbox_task3 = QtWidgets.QCheckBox(self.layoutWidget)
        self.checkbox_task3.setEnabled(True)
        font = QtGui.QFont()
        font.setKerning(True)
        self.checkbox_task3.setFont(font)
        self.checkbox_task3.setCheckable(True)
        self.checkbox_task3.setObjectName("checkbox_task3")
        self.grid_layout_tasks.addWidget(self.checkbox_task3, 2, 5, 1, 1)
        self.lbl_subject2 = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setItalic(False)
        self.lbl_subject2.setFont(font)
        self.lbl_subject2.setObjectName("lbl_subject2")
        self.grid_layout_tasks.addWidget(self.lbl_subject2, 1, 2, 1, 1)
        self.lbl_due_date2 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_due_date2.setObjectName("lbl_due_date2")
        self.grid_layout_tasks.addWidget(self.lbl_due_date2, 1, 4, 1, 1)
```

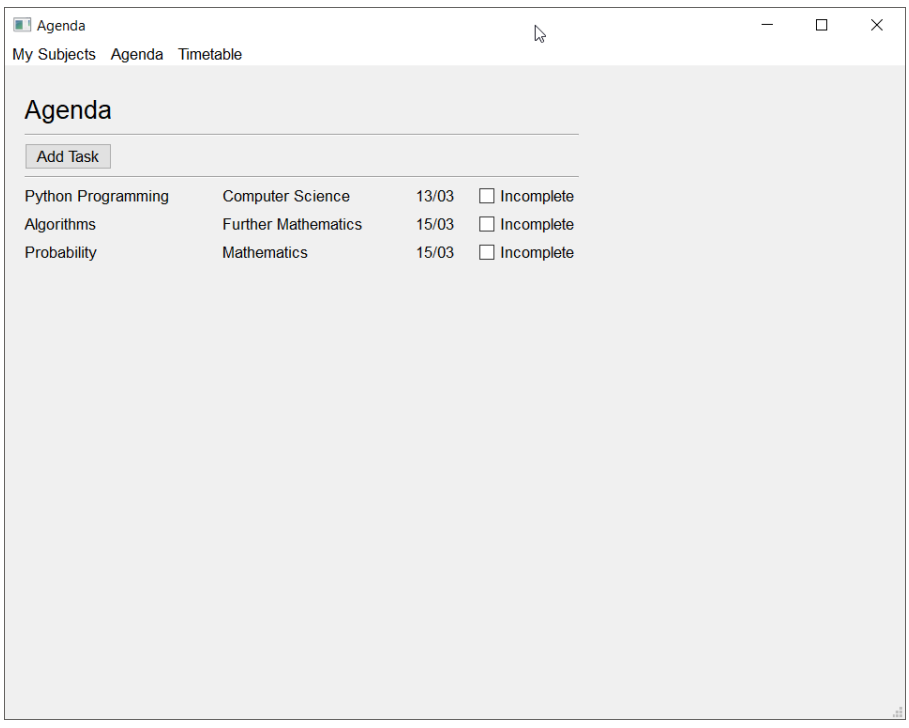
```
        self.lbl_task_title2 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_task_title2.setObjectName("lbl_task_title2")
        self.grid_layout_tasks.addWidget(self.lbl_task_title2, 1, 0, 1, 1)
        self.lbl_task_title3 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_task_title3.setObjectName("lbl_task_title3")
        self.grid_layout_tasks.addWidget(self.lbl_task_title3, 2, 0, 1, 1)
        self.lbl_subject3 = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setItalic(False)
        self.lbl_subject3.setFont(font)
        self.lbl_subject3.setObjectName("lbl_subject3")
        self.grid_layout_tasks.addWidget(self.lbl_subject3, 2, 2, 1, 1)
        self.lbl_subject1 = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setItalic(False)
        self.lbl_subject1.setFont(font)
        self.lbl_subject1.setObjectName("lbl_subject1")
        self.grid_layout_tasks.addWidget(self.lbl_subject1, 0, 2, 1, 1)
        self.checkbox_task1 = QtWidgets.QCheckBox(self.layoutWidget)
        self.checkbox_task1.setEnabled(True)
        font = QtGui.QFont()
        font.setKerning(True)
        self.checkbox_task1.setFont(font)
        self.checkbox_task1.setCheckable(True)
        self.checkbox_task1.setObjectName("checkbox_task1")
        self.grid_layout_tasks.addWidget(self.checkbox_task1, 0, 5, 1, 1)
        self.lbl_due_date3 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_due_date3.setObjectName("lbl_due_date3")
        self.grid_layout_tasks.addWidget(self.lbl_due_date3, 2, 4, 1, 1)
        self.lbl_task_title1 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_task_title1.setObjectName("lbl_task_title1")
        self.grid_layout_tasks.addWidget(self.lbl_task_title1, 0, 0, 1, 1)
        spacerItem = QtWidgets.QSpacerItem(50, 20, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Minimum)
        self.grid_layout_tasks.addItem(spacerItem, 0, 1, 1, 1)
        self.checkbox_task2 = QtWidgets.QCheckBox(self.layoutWidget)
        self.checkbox_task2.setEnabled(True)
        font = QtGui.QFont()
```

```
        font.setKerning(True)
        self.checkbox_task2.setFont(font)
        self.checkbox_task2.setCheckable(True)
        self.checkbox_task2.setObjectName("checkbox_task2")
        self.grid_layout_tasks.addWidget(self.checkbox_task2, 1, 5, 1, 1)
        spacerItem1 = QtWidgets.QSpacerItem(50, 20, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Minimum)
        self.grid_layout_tasks.addItem(spacerItem1, 0, 3, 1, 1)
        self.lbl_due_date1 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_due_date1.setObjectName("lbl_due_date1")
        self.grid_layout_tasks.addWidget(self.lbl_due_date1, 0, 4, 1, 1)
        spacerItem2 = QtWidgets.QSpacerItem(50, 20, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Minimum)
        self.grid_layout_tasks.addItem(spacerItem2, 1, 1, 1, 1)
        spacerItem3 = QtWidgets.QSpacerItem(50, 20, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Minimum)
        self.grid_layout_tasks.addItem(spacerItem3, 1, 3, 1, 1)
        spacerItem4 = QtWidgets.QSpacerItem(50, 20, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Minimum)
        self.grid_layout_tasks.addItem(spacerItem4, 2, 1, 1, 1)
        spacerItem5 = QtWidgets.QSpacerItem(50, 20, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Minimum)
        self.grid_layout_tasks.addItem(spacerItem5, 2, 3, 1, 1)
        self.vert_layout_add_task.addLayout(self.grid_layout_tasks)
        self.vert_layout_agenda.addLayout(self.vert_layout_add_task)
        self.scrl_area_agenda_1.setWidget(self.scrl_area_agenda_2)
        self.gridLayout.addWidget(self.scrl_area_agenda_1, 0, 0, 1, 1)
        mwindow_agenda.setCentralWidget(self.central_widget)
        self.menu_bar = QtWidgets.QMenuBar(mwindow_agenda)
        self.menu_bar.setGeometry(QtCore.QRect(0, 0, 960, 25))
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_bar.setFont(font)
        self.menu_bar.setObjectName("menu_bar")
        self.menu_agenda = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_agenda.setFont(font)
        self.menu_agenda.setObjectName("menu_agenda")
        self.menu_my_subjects = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
```

```
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_my_subjects.setFont(font)
        self.menu_my_subjects.setObjectName("menu_my_subjects")
        self.menu_timetable = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_timetable.setFont(font)
        self.menu_timetable.setObjectName("menu_timetable")
        mwindow_agenda.setMenuBar(self.menu_bar)
        self.status_bar = QtWidgets.QStatusBar(mwindow_agenda)
        self.status_bar.setObjectName("status_bar")
        mwindow_agenda.setStatusBar(self.status_bar)
        self.menu_bar.addAction(self.menu_my_subjects.menuAction())
        self.menu_bar.addAction(self.menu_agenda.menuAction())
        self.menu_bar.addAction(self.menu_timetable.menuAction())

        self.retranslateUi(mwindow_agenda)
        QtCore.QMetaObject.connectSlotsByName(mwindow_agenda)

    def retranslateUi(self, mwindow_agenda):
        _translate = QtCore.QCoreApplication.translate
        mwindow_agenda.setWindowTitle(_translate("mwindow_agenda", "Agenda"))
        self.lbl_agenda.setText(_translate("mwindow_agenda", "Agenda"))
        self.btn_add_task.setText(_translate("mwindow_agenda", "Add Task"))
        self.checkbox_task3.setText(_translate("mwindow_agenda", "Incomplete"))
        self.lbl_subject2.setText(_translate("mwindow_agenda", "Further Mathematics"))
        self.lbl_due_date2.setText(_translate("mwindow_agenda", "15/03"))
        self.lbl_task_title2.setText(_translate("mwindow_agenda", "Algorithms"))
        self.lbl_task_title3.setText(_translate("mwindow_agenda", "Probability"))
        self.lbl_subject3.setText(_translate("mwindow_agenda", "Mathematics"))
        self.lbl_subject1.setText(_translate("mwindow_agenda", "Computer Science"))
        self.checkbox_task1.setText(_translate("mwindow_agenda", "Incomplete"))
        self.lbl_due_date3.setText(_translate("mwindow_agenda", "15/03"))
        self.lbl_task_title1.setText(_translate("mwindow_agenda", "Python Programming"))
        self.checkbox_task2.setText(_translate("mwindow_agenda", "Incomplete"))
        self.lbl_due_date1.setText(_translate("mwindow_agenda", "13/03"))
```

```
        self.menu_agenda.setTitle(_translate("mwindow_agenda", "Agenda"))
        self.menu_my_subjects.setTitle(_translate("mwindow_agenda", "My Subjects"))
        self.menu_timetable.setTitle(_translate("mwindow_agenda", "Timetable"))


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    mwindow_agenda = QtWidgets.QMainWindow()
    ui = Ui_mwindow_agenda()
    ui.setupUi(mwindow_agenda)
    mwindow_agenda.show()
    sys.exit(app.exec_())
```

Testing: Iteration #2 – UI Improvements



Using my test table, I tested the changes I made to see whether using the grid layout would work in practice.

Test	Expected Result	Outcome	Further Actions
Is the user able to access an overview of their tasks?	User should be able to read an overview of key information about their tasks.	User is able to read an overview of key information about their tasks, including task title, subject, due date, and whether it is completed.	N/A.
Is the user able to easily read the agenda?	User should be able to read agenda easily, with data which are related aligned to indicate their relation.	User can differentiate between the individual tasks easily, and details are vertically aligned, so the user knows what type of information in that task is being displayed.	Vertically align the details of the task for clearer distinction about what type of information is being displayed (task title, subject, due date, or completion).

As I thought, the columns expanded to the maximum width required by the entries in that column. For example, if the first row did not exist, the first column would be narrower, as it only needs to fit the word ‘Probability’. All the data in the table is aligned, so the changes I made were successful.

At this point, I also surveyed some of my primary stakeholders, sixth form students, to ask what they thought about the user interface for Agenda, as I made some significant changes compared to the prototype design to adapt to the native design of Windows.

Feedback for the user interface was generally positive, and 100% of my stakeholders who were surveyed stated that they would prefer Agenda without the colour coding at this stage, because adding colour coding would clash with the sleek and minimalistic aesthetic of Agenda.

The sixth form students also explained that adding sound effects would be an unnecessary feature, because they would often be using this program in a school environment, where it would become annoying to other people, meaning they would likely mute the sound if it was added.

Based on this, I decided to cancel my idea to add colour coding for the tasks based on their subject and to add sound effects, as they would detract from the aesthetics and user friendliness of my user interface respectively.

Review

My target for this stage was to create the user interface for the agenda, excluding functionality. The intention was that this user interface would act as the foundations for future development stages on Agenda, by displaying the key information which was mentioned in the success criteria, such as task details, subject, due date, and whether the task was completed.

By the end of this stage, I had created a user interface which is sleek, meaning it can be read easily. It contains all the necessary information, with the header to aid the user with navigation, horizontal lines separating some elements to avoid clutter, a button for students

to add new tasks, and a list of tasks with their task titles, subjects, due dates, and check boxes for task completion.

Next Steps for Module

In later stages, I plan on adding functionality to the 'Add Task' button by programming it to open a dialog window when it is clicked, which will provide the user with a GUI to enter the details of their new task.

Stage 2: My Subjects

Objectives

Now that the initial GUI for Agenda has been completed, this needs also be applied to My Subjects, as the My Subjects screen will be linked to Agenda in later development stages to enable the user to use a dropdown menu to select the subject for their tasks.

Like in stage 1, the objective is only to create a user interface which acts as a foundation for functionality in this part of the program in future development stages. No functionality is expected to be added in this development stage.

Prototype Program

The prototype program I will create for this stage will be very similar to Agenda in stage 1. This is because the concept is the same, but using different variables. In this development stage, I will focus on creating a user interface to display the user's subject names, and it will be developed further in a later stage to enable the user to input additional information about their subject, such as teachers for each individual subject.

Therefore, my prototype program will be focused on code which enables the user to input their subject names, and have these subject names saved to a class of a subject list.

I transferred a lot of the code from the previous stage's prototype program, as the functionality will eventually be very similar.

```
# Defines a class for a list of subjects.
class MySubjects:
    def __init__(self, subject1, subject2, subject3, subject4):
        self.subject1 = subject_name_1
        self.subject2 = subject_name_2
        self.subject3 = subject_name_3
        self.subject4 = subject_name_4

# Validates the entries for subject names.
subject_name_1 = str(input("Subject 1: "))
while len(subject_name_1) > 30:
    print("\nThis subject name is too long.\n")
    subject_name_1 = str(input("Subject 1: "))

subject_name_2 = str(input("Subject 2: "))
while len(subject_name_2) > 30:
    print("\nThis subject name is too long.\n")
    subject_name_2 = str(input("Subject 2: "))

subject_name_3 = str(input("Subject 3: "))
while len(subject_name_3) > 30:
    print("\nThis subject name is too long.\n")
    subject_name_3 = str(input("Subject 3: "))

subject_name_4 = str(input("Subject 4: "))
while len(subject_name_4) > 30:
    print("\nThis subject name is too long.")
    subject_name_4 = str(input("Subject 4: "))

# Adds a list of subject names using the user inputs, and prints a list of the user's subjects.
subjectlist = SubjectList(subject_name_1, subject_name_2, subject_name_3, subject_name_4)
print("\nMy Subjects:\nSubject 1: " + subject_name_1 + "\nSubject 2: " + subject_name_2 + "\nSubject 3: " + subject_name_3 + "\nSubject 4: " + subject_name_4)
```

In this program, I defined a class for the subject list, with the attributes being the individual subject names.

From line 9, the user is prompted to enter their subject names. As with the prototype program in stage 1, validation is performed after each input to ensure that the length of inputs does not exceed 30 characters, as this is the maximum length for subject names. If the user enters a subject name which fails validation, they will be continuously prompted for a valid input until they provide one.

From line 30, the program creates an object for the subject list using the user inputs as attribute entries and the class *MySubjects*. Then, it prints the user's subjects to provide feedback to the user that their subjects have been successfully added.

```
Subject 1: Computer Science
Subject 2: Further Mathematics
Subject 3: History
Subject 4: Mathematics

My Subjects:
Subject 1: Computer Science
Subject 2: Further Mathematics
Subject 3: History
Subject 4: Mathematics
```

When the user enters subject names, which consist of 30 or less characters (as seen above), the program continues until all four subjects have been entered, creates an object for the subject list using these user entries as class attributes, and prints a list of the subjects the user entered.

```
Subject 1: Computer Science
Subject 2: History djsadosajdijsodjioajdioaodjoisajdoijdioajdoiasjdioasjoidjsoaijdiosajdoijajdiosa

This subject name is too long.

Subject 2: History
Subject 3: Further Mathematics
Subject 4: Mathematics djassajdioasjdioasjiaodjioasdoiasodjasidoiajdiodoijsaiodjasjdoasdjasidjoasjdioasodj

This subject name is too long.
Subject 4: Mathematics

My Subjects:
Subject 1: Computer Science
Subject 2: History
Subject 3: Further Mathematics
Subject 4: Mathematics
```

Like in stage 1, the program continually asks for a valid subject name until the user enters one. Once four valid subject names have been entered, the program creates an object for the subject list using the user entries as class attributes, and prints a list of the subjects the user entered.

Development: Iteration #1 – User Interface

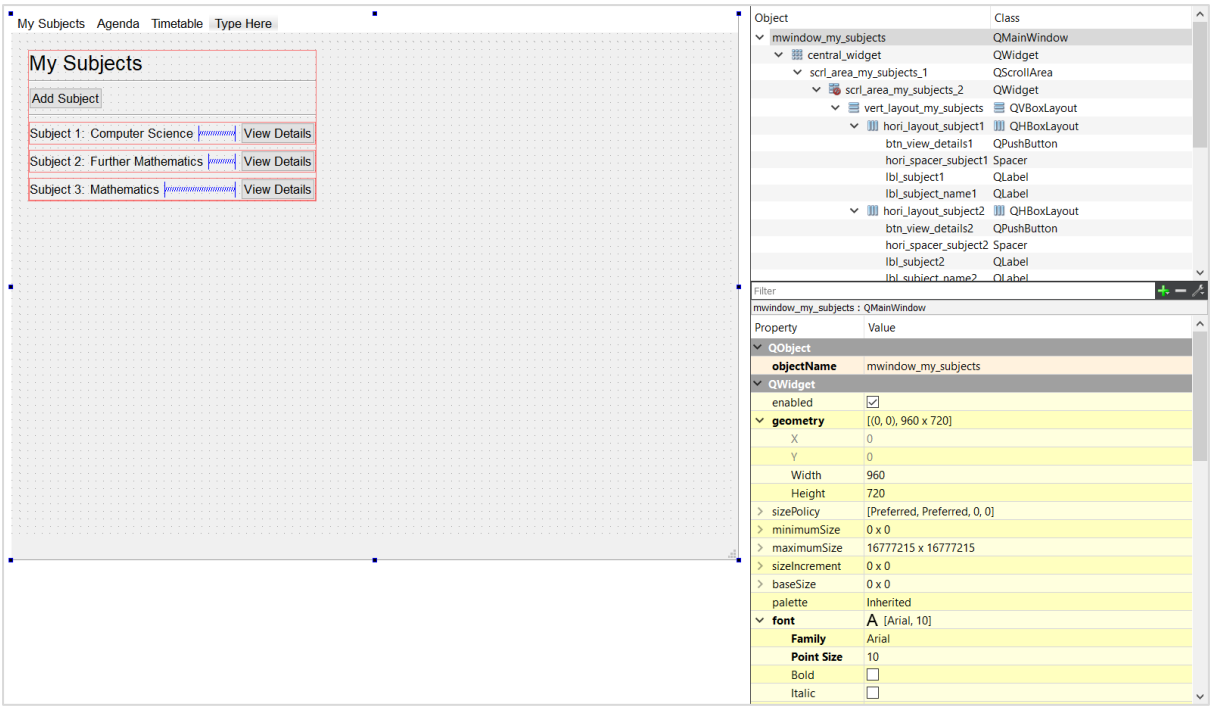
Now that the initial GUI for Agenda has been created, creating a GUI for the user's subjects list is crucial, because this will be the backbone of my program for which basic information such as the list of subjects the user is studying will be collected, stored, and accessed. Without a place for the user to enter their subject details for example, they will not be able to use a dropdown menu to select the subject when adding a new task to the agenda.

I started by creating the same scroll area feature I used in Agenda. This may not be as useful as it was in Agenda, but it gives the user flexibility to add multiple subjects without the text overflowing the window. Once again, a vertical layout is added for the main elements in this window, as this window will likely have a high degree of verticality.

The same menu as Agenda was implemented in this screen, with buttons for ‘My Subjects’, ‘Agenda’, and ‘Timetable’.

A label was added for the ‘My Subjects’ header to aid screen recognition, with an ‘Add Subject’ button to provide the user a way to add new subjects. Horizontal lines were added above and below this button to separate it from the header of the screen and the subjects.

Next, I created a label indicating each subject number, and another label for the subject name on the right side of it, with the labels bound together using horizontal layouts. On the right side of each subject name, I used an expanding horizontal spacer, which ensures that the subject titles are pushed as far to the left as possible, making alignment of the labels easier. Finally, a button for the user to ‘View Details’ of each subject has been added, which will be linked to another window to possibly display additional details about a subject, such as teacher(s) and room.



After I designed this user interface in Qt Creator, I compiled the code using pyuic5.

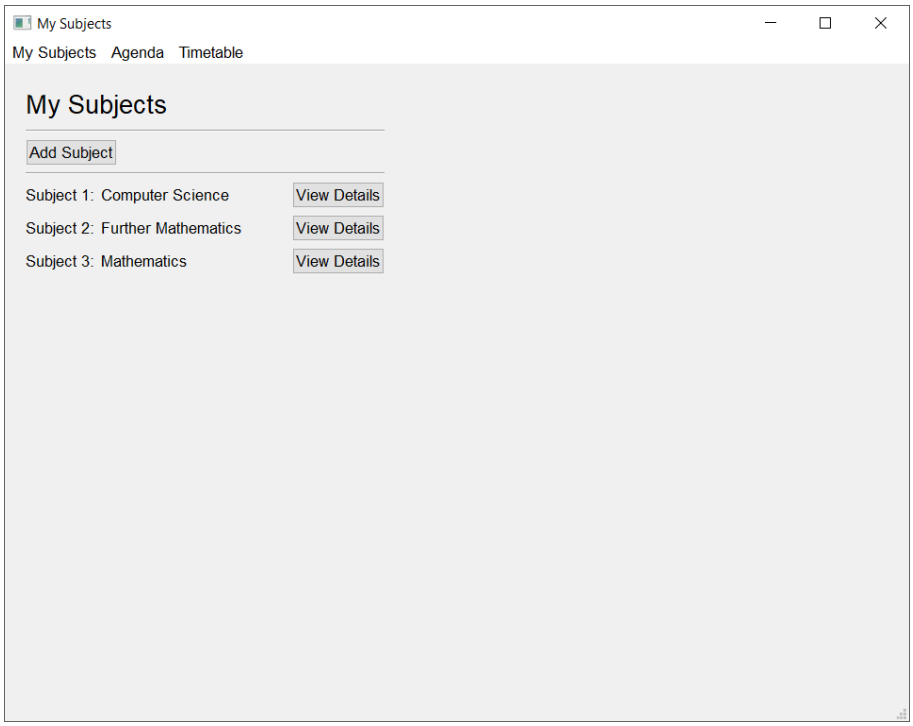
```python
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'my_subjects.ui'
#
# Created by: PyQt5 UI code generator 5.11.3
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_mwindow_my_subjects(object):
    def setupUi(self, mwindow_my_subjects):
        mwindow_my_subjects.setObjectName("mwindow_my_subjects")
        mwindow_my_subjects.resize(960, 720)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        mwindow_my_subjects.setFont(font)
        self.central_widget = QtWidgets.QWidget(mwindow_my_subjects)
        self.central_widget.setObjectName("central_widget")
        self.gridLayout = QtWidgets.QGridLayout(self.central_widget)
        self.gridLayout.setObjectName("gridLayout")
        self.scrl_area_my_subjects_1 = QtWidgets.QScrollArea(self.central_widget)
        self.scrl_area_my_subjects_1.setFrameShape(QtWidgets.QFrame.NoFrame)
        self.scrl_area_my_subjects_1.setWidgetResizable(True)
        self.scrl_area_my_subjects_1.setObjectName("scrl_area_my_subjects_1")
        self.scrl_area_my_subjects_2 = QtWidgets.QWidget()
        self.scrl_area_my_subjects_2.setGeometry(QtCore.QRect(0, 0, 938, 648))
        self.scrl_area_my_subjects_2.setObjectName("scrl_area_my_subjects_2")
        self.layoutWidget = QtWidgets.QWidget(self.scrl_area_my_subjects_2)
        self.layoutWidget.setGeometry(QtCore.QRect(11, 11, 381, 200))
        self.layoutWidget.setObjectName("layoutWidget")
        self.vert_layout_my_subjects = QtWidgets.QVBoxLayout(self.layoutWidget)
        self.vert_layout_my_subjects.setContentsMargins(0, 0, 0, 0)
        self.vert_layout_my_subjects.setObjectName("vert_layout_my_subjects")
        self.lbl_my_subjects = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setPointSize(16)
        self.lbl_my_subjects.setFont(font)
        self.lbl_my_subjects.setObjectName("lbl_my_subjects")
        self.vert_layout_my_subjects.addWidget(self.lbl_my_subjects)
        self.hori_line_my_subjects = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_my_subjects.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_my_subjects.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_my_subjects.setObjectName("hori_line_my_subjects")
        self.vert_layout_my_subjects.addWidget(self.hori_line_my_subjects)
        self.btn_add_subject = QtWidgets.QPushButton(self.layoutWidget)
        self.btn_add_subject.setObjectName("btn_add_subject")
        self.vert_layout_my_subjects.addWidget(self.btn_add_subject, 0, QtCore.Qt.AlignLeft)
        self.hori_line_add_subject = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_add_subject.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_add_subject.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_add_subject.setObjectName("hori_line_add_subject")
        self.vert_layout_my_subjects.addWidget(self.hori_line_add_subject)
        self.hori_layout_subject1 = QtWidgets.QHBoxLayout()
        self.hori_layout_subject1.setObjectName("hori_layout_subject1")
        self.lbl_subject1 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_subject1.setObjectName("lbl_subject1")
        self.hori_layout_subject1.addWidget(self.lbl_subject1, 0, QtCore.Qt.AlignLeft)
        self.lbl_subject_name1 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_subject_name1.setObjectName("lbl_subject_name1")
        self.hori_layout_subject1.addWidget(self.lbl_subject_name1)
        spacerItem = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Minimum)
        self.hori_layout_subject1.addItem(spacerItem)
        self.btn_view_details1 = QtWidgets.QPushButton(self.layoutWidget)
        self.btn_view_details1.setObjectName("btn_view_details1")
        self.hori_layout_subject1.addWidget(self.btn_view_details1)
        self.vert_layout_my_subjects.addLayout(self.hori_layout_subject1)
        self.hori_layout_subject2 = QtWidgets.QHBoxLayout()
        self.hori_layout_subject2.setObjectName("hori_layout_subject2")
        self.lbl_subject2 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_subject2.setObjectName("lbl_subject2")
        self.hori_layout_subject2.addWidget(self.lbl_subject2, 0, QtCore.Qt.AlignLeft)
        self.lbl_subject_name2 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_subject_name2.setObjectName("lbl_subject_name2")
        self.hori_layout_subject2.addWidget(self.lbl_subject_name2)
```

```
        spacerItem1 = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Minimum)
        self.hori_layout_subject2.addItem(spacerItem1)
        self.btn_view_details2 = QtWidgets.QPushButton(self.layoutWidget)
        self.btn_view_details2.setObjectName("btn_view_details2")
        self.hori_layout_subject2.addWidget(self.btn_view_details2)
        self.vert_layout_my_subjects.addLayout(self.hori_layout_subject2)
        self.hori_layout_subject3 = QtWidgets.QHBoxLayout()
        self.hori_layout_subject3.setObjectName("hori_layout_subject3")
        self.lbl_subject3 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_subject3.setObjectName("lbl_subject3")
        self.hori_layout_subject3.addWidget(self.lbl_subject3, 0, QtCore.Qt.AlignLeft)
        self.lbl_subject_name3 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_subject_name3.setObjectName("lbl_subject_name3")
        self.hori_layout_subject3.addWidget(self.lbl_subject_name3)
        spacerItem2 = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Minimum)
        self.hori_layout_subject3.addItem(spacerItem2)
        self.btn_view_details3 = QtWidgets.QPushButton(self.layoutWidget)
        self.btn_view_details3.setObjectName("btn_view_details3")
        self.hori_layout_subject3.addWidget(self.btn_view_details3)
        self.vert_layout_my_subjects.addLayout(self.hori_layout_subject3)
        self.scrl_area_my_subjects_1.setWidget(self.scrl_area_my_subjects_2)
        self.gridLayout.addWidget(self.scrl_area_my_subjects_1, 0, 0, 1, 1)
        mwindow_my_subjects.setCentralWidget(self.central_widget)
        self.menu_bar = QtWidgets.QMenuBar(mwindow_my_subjects)
        self.menu_bar.setGeometry(QtCore.QRect(0, 0, 960, 25))
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_bar.setFont(font)
        self.menu_bar.setObjectName("menu_bar")
        self.menu_my_subjects = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_my_subjects.setFont(font)
        self.menu_my_subjects.setObjectName("menu_my_subjects")
        self.menu_agenda = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
```

```
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_agenda.setFont(font)
        self.menu_agenda.setObjectName("menu_agenda")
        self.menu_timetable = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_timetable.setFont(font)
        self.menu_timetable.setObjectName("menu_timetable")
        mwindow_my_subjects.setMenuBar(self.menu_bar)
        self.status_bar = QtWidgets.QStatusBar(mwindow_my_subjects)
        self.status_bar.setObjectName("status_bar")
        mwindow_my_subjects.setStatusBar(self.status_bar)
        self.actionAgenda = QtWidgets.QAction(mwindow_my_subjects)
        self.actionAgenda.setObjectName("actionAgenda")
        self.menu_bar.addAction(self.menu_my_subjects.menuAction())
        self.menu_bar.addAction(self.menu_agenda.menuAction())
        self.menu_bar.addAction(self.menu_timetable.menuAction())

        self.retranslateUi(mwindow_my_subjects)
        QtCore.QMetaObject.connectSlotsByName(mwindow_my_subjects)

    def retranslateUi(self, mwindow_my_subjects):
        _translate = QtCore.QCoreApplication.translate
        mwindow_my_subjects.setWindowTitle(_translate("mwindow_my_subjects", "My Subjects"))
        self.lbl_my_subjects.setText(_translate("mwindow_my_subjects", "My Subjects"))
        self.btn_add_subject.setText(_translate("mwindow_my_subjects", "Add Subject"))
        self.lbl_subject1.setText(_translate("mwindow_my_subjects", "Subject 1:"))
        self.lbl_subject_name1.setText(_translate("mwindow_my_subjects", "Computer Science"))
        self.btn_view_details1.setText(_translate("mwindow_my_subjects", "View Details"))
        self.lbl_subject2.setText(_translate("mwindow_my_subjects", "Subject 2:"))
        self.lbl_subject_name2.setText(_translate("mwindow_my_subjects", "Further Mathematics"))
        self.btn_view_details2.setText(_translate("mwindow_my_subjects", "View Details"))
        self.lbl_subject3.setText(_translate("mwindow_my_subjects", "Subject 3:"))
        self.lbl_subject_name3.setText(_translate("mwindow_my_subjects", "Mathematics"))
        self.btn_view_details3.setText(_translate("mwindow_my_subjects", "View Details"))
        self.menu_my_subjects.setTitle(_translate("mwindow_my_subjects", "My Subjects"))
```

```
        self.menu_agenda.setTitle(_translate("mwindow_my_subjects", "Agenda"))
        self.menu_timetable.setTitle(_translate("mwindow_my_subjects", "Timetable"))
        self.actionAgenda.setText(_translate("mwindow_my_subjects", "Agenda"))


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    mwindow_my_subjects = QtWidgets.QMainWindow()
    ui = Ui_mwindow_my_subjects()
    ui.setupUi(mwindow_my_subjects)
    mwindow_my_subjects.show()
    sys.exit(app.exec_())
```

Testing: Iteration #1 – User Interface



A test table was used for testing the first iteration of this stage once development for this iteration had been completed.

Test	Expected Result	Outcome	Further Actions
Can the user view their subjects?	User should be able to view of their subject names.	User is able to view a list of their subject names.	N/A.
Can the list of subjects be easily read?	The list of subjects should be easily read, with vertical alignment of the subjects for clearer interpretation by the user.	The list of subjects can be easily read, with vertical alignment of the subjects for clearer interpretation by the user.	N/A.

I tested the developments which I had made, with the expectation that there should be no issues, as the program appeared to be working in good condition using the WYSIWYG Qt Creator interface. This turned out to be true, as all the elements were aligned as I intended; the subject numbers, subject names, and 'View Details' buttons were all perfectly vertically aligned, with the spacers adapting to provide horizontal spacing of varying sizes depending on the length of the subject number and subject name.

No issues were encountered during testing at this stage. None of the buttons performed anything apart from playing the animations when clicked, but this was not expected as I had not programmed any of buttons to perform any actions at this stage.

Review

My target for this stage was to create a user interface for My Subjects, with the idea that it would form the foundation for functionality to be added at a later stage.

This target was achieved by the end of the stage, with only one development iteration being required, as I had learned from the previous stage about how to solve issues with readability. I created a simple user interface which is similar to Agenda, but used for a different purpose. It can be read easily, and the essential parts of the user interface are all included.

Next Steps for Module

Later, I plan on adding functionality to the 'Add Subject' button, so the user can add new subjects, as well as the 'View Details' buttons, so the user can view additional information about each subject in a pop-up window.

Stage 3: Timetable

Objectives

Timetable is the final essential part of my program which will be a high priority for development, so it will be the last thing I will create an initial GUI for at this point, as a grade book would be an optional part of my program depending on time constraints.

The objectives for this stage are to create a user interface which allows the user to view their lesson schedule for each weekday. This should be structured in a typical table format, with columns representing the day of the week, and rows representing lesson slots.

Prototype Program

Part of the functionality of the timetable would be allowing the user to edit the timetable, so they can fill in the timetable slots, allowing them to use this timetable as a reference in case they forget which lesson they have at a specific time and/or day. This development stage will be focused on creating a non-functional GUI for Timetable, but the prototype program I am creating could provide me with some insight on how I could develop some basic functionality for this timetable in the next development stage of Timetable.

```
# Defines a class for information in each cell of a timetable.
class Timetable:
    def __init__(self, lesson_subject, lesson_room):
        self.lesson_subject = lesson_subject
        self.lesson_room = lesson_room

# Asks the user for the subject and room for that lesson.
lesson_subject = str(input("Please enter the subject for this lesson: "))
while len(lesson_subject) > 30:
    print("This subject is too long. Please try again.\n")
    lesson_subject = str(input("Please enter the subject for this lesson: "))
lesson_room = str(input("Please enter the classroom for this lesson: "))
while len(lesson_room) > 15:
    print("This room name is too long. Please try again.\n")
    lesson_room = str(input("Please enter the classroom for this lesson: "))

# Stores the subject and room as one variable which acts as a label.
lesson = Timetable(lesson_subject, lesson_room)
lesson_label = lesson_subject + "\n" + lesson_room
print("\n"+ lesson_label)
```

At the start of this program, I defined the class *Timetable*, with the attributes of *lesson_subject* and *lesson_room*, which represent the subject for that cell in the timetable, and the room the lesson is being held in.

The user is asked to enter the subject and classroom for that cell in the timetable from lines 7 to 15. The program validates these entries to ensure that their character lengths are within a reasonable range (30 and 15 respectively), requesting for the user to re-enter values until they provide a valid input.

Once both inputs have been found to be valid, the program adds this lesson object to the class. Then, the program combines these two string variables into a variable acting as a label, with the contents being spread across separate lines for readability. Once it has combined the two strings, it prints the variable for the user to see as feedback.

The prototype for this stage works similarly to stage 1 and stage 2, but it combines the inputted variables to create a single label, which will be demonstrated as necessary in the development stage due to the cellular nature of tables in Qt Creator.

```
Please enter the subject for this lesson: Computer Science
Please enter the classroom for this lesson: C2

Computer Science
C2
```

When the user enters valid inputs, the program accepts these values for the variables, combines them, and then prints them, as seen above. This was observed in the test performed above.

```
Please enter the subject for this lesson: Computer Science jdoisajdasodjasdosajdiosajodjsaoidjioasjdoiaso
This subject is too long. Please try again.

Please enter the subject for this lesson: Computer Science dsjadoiasjdosadsadoisad
This subject is too long. Please try again.

Please enter the subject for this lesson: Computer Science
Please enter the classroom for this lesson: This classroom name is too long.
This room name is too long. Please try again.

Please enter the classroom for this lesson: C2

Computer Science
C2
```

Meanwhile, the user will be continuously prompted for a valid input until the program receives one which is not greater than 30 characters for the subject name, or 15 characters for the room name. This can be observed above.

Development: Iteration #1 – User Interface

The third part of my program which I am focusing on for the start is the timetable. This is where the user will be able to enter their subjects in their appropriate lesson slots, so they can look at their student planner as a reference in cases where they lose track of the lesson they must be in, and which room it is in.

Many of the same elements as Agenda and My Subjects were used in this GUI to ensure that the user interface is recognisable by the user, making it easier to use. This included the menu at the top, the header (albeit with the text changed to ‘Timetable’), the button (with the text changed to ‘Edit Timetable’, and the horizontal lines above and below the button.

The part of this window which I needed to develop uniquely was the lesson table. To avoid having the same issues as I had experienced in Agenda, I decided to use a grid interface, as

this would especially be suited to creating a table, allowing me to ensure that the rows and columns are aligned horizontally and vertically respectively depending on the maximum heights and widths required for the content.

This provides flexibility for me to add the amount of content I desire in each cell in later stages. To start with, the program will simply add the subject name in each cell when the user adds a timetable slot, but I may develop this further to include the name of the teacher in each cell, which would likely mean the text in each cell would run over multiple lines.

```python
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'timetable.ui'
#
# Created by: PyQt5 UI code generator 5.11.3
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_mwindow_timetable(object):
    def setupUi(self, mwindow_timetable):
        mwindow_timetable.setObjectName("mwindow_timetable")
        mwindow_timetable.resize(960, 720)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        font.setKerning(True)
        mwindow_timetable.setFont(font)
        self.centralwidget = QtWidgets.QWidget(mwindow_timetable)
        self.centralwidget.setObjectName("centralwidget")
        self.gridLayout = QtWidgets.QGridLayout(self.centralwidget)
        self.gridLayout.setObjectName("gridLayout")
        self.scrl_area_timetable_1 = QtWidgets.QScrollArea(self.centralwidget)
        self.scrl_area_timetable_1.setFrameShape(QtWidgets.QFrame.NoFrame)
        self.scrl_area_timetable_1.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.scrl_area_timetable_1.setLineWidth(0)
        self.scrl_area_timetable_1.setWidgetResizable(True)
        self.scrl_area_timetable_1.setObjectName("scrl_area_timetable_1")
        self.scrl_area_timetable_2 = QtWidgets.QWidget()
        self.scrl_area_timetable_2.setGeometry(QtCore.QRect(0, 0, 938, 647))
        font = QtGui.QFont()
        font.setKerning(True)
        self.scrl_area_timetable_2.setFont(font)
        self.scrl_area_timetable_2.setAutoFillBackground(True)
        self.scrl_area_timetable_2.setObjectName("scrl_area_timetable_2")
        self.layoutWidget = QtWidgets.QWidget(self.scrl_area_timetable_2)
        self.layoutWidget.setGeometry(QtCore.QRect(11, 11, 769, 254))
```

```python
        self.layoutWidget.setObjectName("layoutWidget")
        self.vert_layout_timetable = QtWidgets.QVBoxLayout(self.layoutWidget)
        self.vert_layout_timetable.setContentsMargins(0, 0, 0, 0)
        self.vert_layout_timetable.setObjectName("vert_layout_timetable")
        self.lbl_timetable = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setPointSize(16)
        self.lbl_timetable.setFont(font)
        self.lbl_timetable.setObjectName("lbl_timetable")
        self.vert_layout_timetable.addWidget(self.lbl_timetable)
        self.hori_line_timetable = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_timetable.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_timetable.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_timetable.setObjectName("hori_line_timetable")
        self.vert_layout_timetable.addWidget(self.hori_line_timetable)
        self.btn_add_subject = QtWidgets.QPushButton(self.layoutWidget)
        self.btn_add_subject.setObjectName("btn_add_subject")
        self.vert_layout_timetable.addWidget(self.btn_add_subject, 0, QtCore.Qt.AlignLeft)
        self.hori_line_edit_timetable = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_edit_timetable.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_edit_timetable.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_edit_timetable.setObjectName("hori_line_edit_timetable")
        self.vert_layout_timetable.addWidget(self.hori_line_edit_timetable)
        self.grid_layout_lessons = QtWidgets.QGridLayout()
        self.grid_layout_lessons.setObjectName("grid_layout_lessons")
        self.lbl_friday_lesson3 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_friday_lesson3.setObjectName("lbl_friday_lesson3")
        self.grid_layout_lessons.addWidget(self.lbl_friday_lesson3, 4, 5, 1, 1)
        self.lbl_monday_lesson2 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_monday_lesson2.setObjectName("lbl_monday_lesson2")
        self.grid_layout_lessons.addWidget(self.lbl_monday_lesson2, 3, 1, 1, 1)
        self.lbl_wednesday = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_wednesday.setObjectName("lbl_wednesday")
        self.grid_layout_lessons.addWidget(self.lbl_wednesday, 1, 3, 1, 1)
        self.lbl_tuesday_lesson5 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_tuesday_lesson5.setObjectName("lbl_tuesday_lesson5")
        self.grid_layout_lessons.addWidget(self.lbl_tuesday_lesson5, 6, 2, 1, 1)
        self.lbl_wednesday_lesson5 = QtWidgets.QLabel(self.layoutWidget)
```

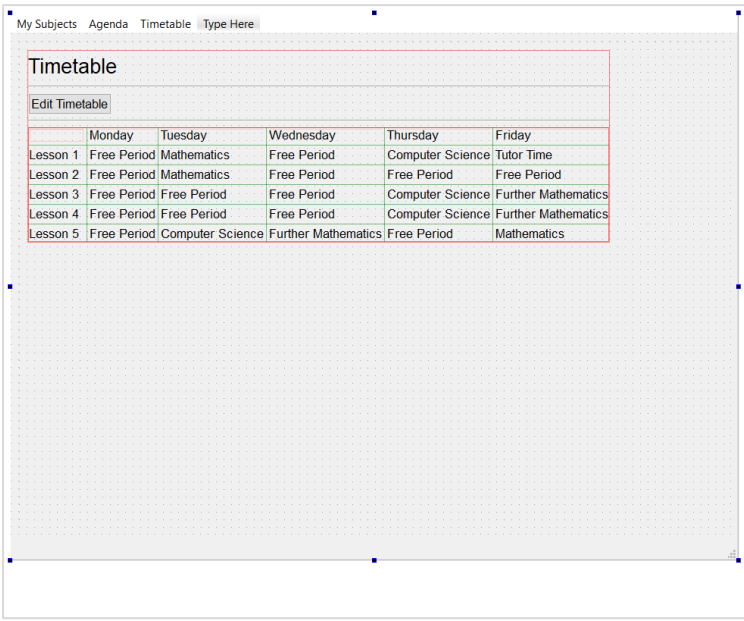
```
        self.lbl_wednesday_lesson5.setObjectName("lbl_wednesday_lesson5")
        self.grid_layout_lessons.addWidget(self.lbl_wednesday_lesson5, 6, 3, 1, 1)
        self.lbl_tuesday_lesson1 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_tuesday_lesson1.setObjectName("lbl_tuesday_lesson1")
        self.grid_layout_lessons.addWidget(self.lbl_tuesday_lesson1, 2, 2, 1, 1)
        self.lbl_monday_lesson5 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_monday_lesson5.setObjectName("lbl_monday_lesson5")
        self.grid_layout_lessons.addWidget(self.lbl_monday_lesson5, 6, 1, 1, 1)
        self.lbl_wednesday_lesson4 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_wednesday_lesson4.setObjectName("lbl_wednesday_lesson4")
        self.grid_layout_lessons.addWidget(self.lbl_wednesday_lesson4, 5, 3, 1, 1)
        self.lbl_wednesday_lesson1 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_wednesday_lesson1.setObjectName("lbl_wednesday_lesson1")
        self.grid_layout_lessons.addWidget(self.lbl_wednesday_lesson1, 2, 3, 1, 1)
        self.lbl_thursday = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_thursday.setObjectName("lbl_thursday")
        self.grid_layout_lessons.addWidget(self.lbl_thursday, 1, 4, 1, 1)
        self.lbl_friday_lesson1 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_friday_lesson1.setObjectName("lbl_friday_lesson1")
        self.grid_layout_lessons.addWidget(self.lbl_friday_lesson1, 2, 5, 1, 1)
        self.lbl_thursday_lesson2 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_thursday_lesson2.setObjectName("lbl_thursday_lesson2")
        self.grid_layout_lessons.addWidget(self.lbl_thursday_lesson2, 3, 4, 1, 1)
        self.lbl_thursday_lesson1 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_thursday_lesson1.setObjectName("lbl_thursday_lesson1")
        self.grid_layout_lessons.addWidget(self.lbl_thursday_lesson1, 2, 4, 1, 1)
        self.lbl_tuesday_lesson4 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_tuesday_lesson4.setObjectName("lbl_tuesday_lesson4")
        self.grid_layout_lessons.addWidget(self.lbl_tuesday_lesson4, 5, 2, 1, 1)
        self.lbl_thursday_lesson4 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_thursday_lesson4.setObjectName("lbl_thursday_lesson4")
        self.grid_layout_lessons.addWidget(self.lbl_thursday_lesson4, 5, 4, 1, 1)
        self.lbl_friday_lesson5 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_friday_lesson5.setObjectName("lbl_friday_lesson5")
        self.grid_layout_lessons.addWidget(self.lbl_friday_lesson5, 6, 5, 1, 1)
        self.lbl_lesson3 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_lesson3.setObjectName("lbl_lesson3")
        self.grid_layout_lessons.addWidget(self.lbl_lesson3, 4, 0, 1, 1)
```

```
        self.lbl_monday_lesson4 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_monday_lesson4.setObjectName("lbl_monday_lesson4")
        self.grid_layout_lessons.addWidget(self.lbl_monday_lesson4, 5, 1, 1, 1)
        self.lbl_tuesday = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_tuesday.setObjectName("lbl_tuesday")
        self.grid_layout_lessons.addWidget(self.lbl_tuesday, 1, 2, 1, 1)
        self.lbl_wednesday_lesson2 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_wednesday_lesson2.setObjectName("lbl_wednesday_lesson2")
        self.grid_layout_lessons.addWidget(self.lbl_wednesday_lesson2, 3, 3, 1, 1)
        self.lbl_thursday_lesson5 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_thursday_lesson5.setObjectName("lbl_thursday_lesson5")
        self.grid_layout_lessons.addWidget(self.lbl_thursday_lesson5, 6, 4, 1, 1)
        self.lbl_tuesday_lesson3 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_tuesday_lesson3.setObjectName("lbl_tuesday_lesson3")
        self.grid_layout_lessons.addWidget(self.lbl_tuesday_lesson3, 4, 2, 1, 1)
        self.lbl_friday_lesson4 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_friday_lesson4.setObjectName("lbl_friday_lesson4")
        self.grid_layout_lessons.addWidget(self.lbl_friday_lesson4, 5, 5, 1, 1)
        self.lbl_monday = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setPointSize(10)
        self.lbl_monday.setFont(font)
        self.lbl_monday.setObjectName("lbl_monday")
        self.grid_layout_lessons.addWidget(self.lbl_monday, 1, 1, 1, 1)
        self.lbl_friday_lesson2 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_friday_lesson2.setObjectName("lbl_friday_lesson2")
        self.grid_layout_lessons.addWidget(self.lbl_friday_lesson2, 3, 5, 1, 1)
        self.lbl_tuesday_lesson2 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_tuesday_lesson2.setObjectName("lbl_tuesday_lesson2")
        self.grid_layout_lessons.addWidget(self.lbl_tuesday_lesson2, 3, 2, 1, 1)
        self.lbl_monday_lesson3 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_monday_lesson3.setObjectName("lbl_monday_lesson3")
        self.grid_layout_lessons.addWidget(self.lbl_monday_lesson3, 4, 1, 1, 1)
        self.lbl_lesson1 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_lesson1.setObjectName("lbl_lesson1")
        self.grid_layout_lessons.addWidget(self.lbl_lesson1, 2, 0, 1, 1)
        self.lbl_lesson5 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_lesson5.setObjectName("lbl_lesson5")
```

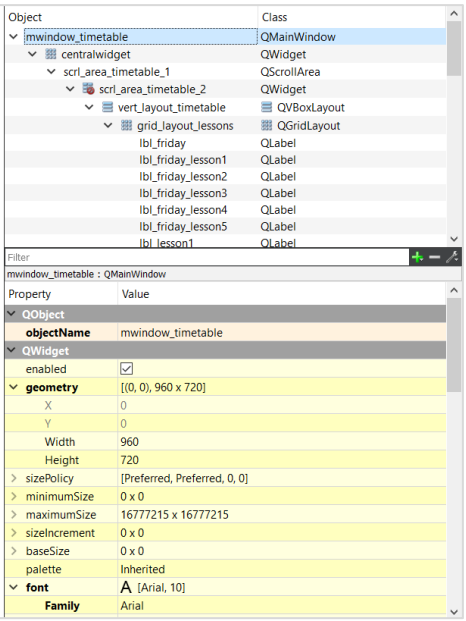
```
        self.grid_layout_lessons.addWidget(self.lbl_lesson5, 6, 0, 1, 1)
        self.lbl_friday = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_friday.setObjectName("lbl_friday")
        self.grid_layout_lessons.addWidget(self.lbl_friday, 1, 5, 1, 1)
        self.lbl_lesson4 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_lesson4.setObjectName("lbl_lesson4")
        self.grid_layout_lessons.addWidget(self.lbl_lesson4, 5, 0, 1, 1)
        self.lbl_wednesday_lesson3 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_wednesday_lesson3.setObjectName("lbl_wednesday_lesson3")
        self.grid_layout_lessons.addWidget(self.lbl_wednesday_lesson3, 4, 3, 1, 1)
        self.lbl_lesson2 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_lesson2.setObjectName("lbl_lesson2")
        self.grid_layout_lessons.addWidget(self.lbl_lesson2, 3, 0, 1, 1)
        self.lbl_thursday_lesson3 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_thursday_lesson3.setObjectName("lbl_thursday_lesson3")
        self.grid_layout_lessons.addWidget(self.lbl_thursday_lesson3, 4, 4, 1, 1)
        self.lbl_monday_lesson1 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_monday_lesson1.setObjectName("lbl_monday_lesson1")
        self.grid_layout_lessons.addWidget(self.lbl_monday_lesson1, 2, 1, 1, 1)
        self.vert_layout_timetable.addLayout(self.grid_layout_lessons)
        self.scrl_area_timetable_1.setWidget(self.scrl_area_timetable_2)
        self.gridLayout.addWidget(self.scrl_area_timetable_1, 0, 0, 1, 1)
        mwindow_timetable.setCentralWidget(self.centralwidget)
        self.menu_bar = QtWidgets.QMenuBar(mwindow_timetable)
        self.menu_bar.setGeometry(QtCore.QRect(0, 0, 960, 26))
        self.menu_bar.setObjectName("menu_bar")
        self.menu_my_subjects = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_my_subjects.setFont(font)
        self.menu_my_subjects.setObjectName("menu_my_subjects")
        self.menu_agenda = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_agenda.setFont(font)
        self.menu_agenda.setObjectName("menu_agenda")
```

```
        self.menu_timetable = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_timetable.setFont(font)
        self.menu_timetable.setObjectName("menu_timetable")
        mwindow_timetable.setMenuBar(self.menu_bar)
        self.status_bar = QtWidgets.QStatusBar(mwindow_timetable)
        self.status_bar.setObjectName("status_bar")
        mwindow_timetable.setStatusBar(self.status_bar)
        self.actionAgenda = QtWidgets.QAction(mwindow_timetable)
        self.actionAgenda.setObjectName("actionAgenda")
        self.menu_bar.addAction(self.menu_my_subjects.menuAction())
        self.menu_bar.addAction(self.menu_agenda.menuAction())
        self.menu_bar.addAction(self.menu_timetable.menuAction())

        self.retranslateUi(mwindow_timetable)
        QtCore.QMetaObject.connectSlotsByName(mwindow_timetable)

    def retranslateUi(self, mwindow_timetable):
        _translate = QtCore.QCoreApplication.translate
        mwindow_timetable.setWindowTitle(_translate("mwindow_timetable", "Timetable"))
        self.lbl_timetable.setText(_translate("mwindow_timetable", "Timetable"))
        self.btn_add_subject.setText(_translate("mwindow_timetable", "Edit Timetable"))
        self.lbl_friday_lesson3.setText(_translate("mwindow_timetable", "Further Mathematics"))
        self.lbl_monday_lesson2.setText(_translate("mwindow_timetable", "Free Period"))
        self.lbl_wednesday.setText(_translate("mwindow_timetable", "Wednesday"))
        self.lbl_tuesday_lesson5.setText(_translate("mwindow_timetable", "Computer Science"))
        self.lbl_wednesday_lesson5.setText(_translate("mwindow_timetable", "Further Mathematics"))
        self.lbl_tuesday_lesson1.setText(_translate("mwindow_timetable", "Mathematics"))
        self.lbl_monday_lesson5.setText(_translate("mwindow_timetable", "Free Period"))
        self.lbl_wednesday_lesson4.setText(_translate("mwindow_timetable", "Free Period"))
        self.lbl_wednesday_lesson1.setText(_translate("mwindow_timetable", "Free Period"))
        self.lbl_thursday.setText(_translate("mwindow_timetable", "Thursday"))
        self.lbl_friday_lesson1.setText(_translate("mwindow_timetable", "Tutor Time"))
        self.lbl_thursday_lesson2.setText(_translate("mwindow_timetable", "Free Period"))
        self.lbl_thursday_lesson1.setText(_translate("mwindow_timetable", "Computer Science"))
        self.lbl_tuesday_lesson4.setText(_translate("mwindow_timetable", "Free Period"))
```

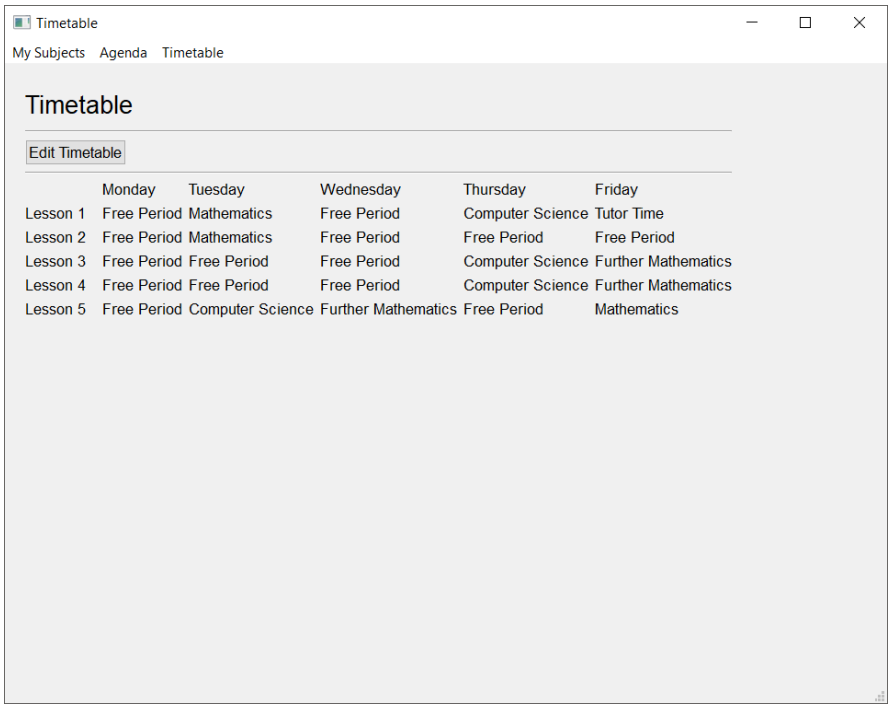
```
        self.lbl_thursday_lesson4.setText(_translate("mwindow_timetable", "Computer Science"))
        self.lbl_friday_lesson5.setText(_translate("mwindow_timetable", "Mathematics"))
        self.lbl_lesson3.setText(_translate("mwindow_timetable", "Lesson 3"))
        self.lbl_monday_lesson4.setText(_translate("mwindow_timetable", "Free Period"))
        self.lbl_tuesday.setText(_translate("mwindow_timetable", "Tuesday"))
        self.lbl_wednesday_lesson2.setText(_translate("mwindow_timetable", "Free Period"))
        self.lbl_thursday_lesson5.setText(_translate("mwindow_timetable", "Free Period"))
        self.lbl_tuesday_lesson3.setText(_translate("mwindow_timetable", "Free Period"))
        self.lbl_friday_lesson4.setText(_translate("mwindow_timetable", "Further Mathematics"))
        self.lbl_monday.setText(_translate("mwindow_timetable", "Monday"))
        self.lbl_friday_lesson2.setText(_translate("mwindow_timetable", "Free Period"))
        self.lbl_tuesday_lesson2.setText(_translate("mwindow_timetable", "Mathematics"))
        self.lbl_monday_lesson3.setText(_translate("mwindow_timetable", "Free Period"))
        self.lbl_lesson1.setText(_translate("mwindow_timetable", "Lesson 1"))
        self.lbl_lesson5.setText(_translate("mwindow_timetable", "Lesson 5"))
        self.lbl_friday.setText(_translate("mwindow_timetable", "Friday"))
        self.lbl_lesson4.setText(_translate("mwindow_timetable", "Lesson 4"))
        self.lbl_wednesday_lesson3.setText(_translate("mwindow_timetable", "Free Period"))
        self.lbl_lesson2.setText(_translate("mwindow_timetable", "Lesson 2"))
        self.lbl_thursday_lesson3.setText(_translate("mwindow_timetable", "Computer Science"))
        self.lbl_monday_lesson1.setText(_translate("mwindow_timetable", "Free Period"))
        self.menu_my_subjects.setTitle(_translate("mwindow_timetable", "My Subjects"))
        self.menu_agenda.setTitle(_translate("mwindow_timetable", "Agenda"))
        self.menu_timetable.setTitle(_translate("mwindow_timetable", "Timetable"))
        self.actionAgenda.setText(_translate("mwindow_timetable", "Agenda"))


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    mwindow_timetable = QtWidgets.QMainWindow()
    ui = Ui_mwindow_timetable()
    ui.setupUi(mwindow_timetable)
    mwindow_timetable.show()
    sys.exit(app.exec_())
```



	Monday	Tuesday	Wednesday	Thursday	Friday
Lesson 1	Free Period	Mathematics	Free Period	Computer Science	Tutor Time
Lesson 2	Free Period	Mathematics	Free Period	Free Period	Free Period
Lesson 3	Free Period	Free Period	Free Period	Computer Science	Further Mathematics
Lesson 4	Free Period	Free Period	Free Period	Computer Science	Further Mathematics
Lesson 5	Free Period	Computer Science	Further Mathematics	Free Period	Mathematics



Testing: Iteration #1 – User Interface



Once again, I used a test table to see whether my development had fulfilled the objectives which were set at the start of the stage.

Test	Expected Result	Outcome	Further Actions
Can the user view their schedule for each weekday?	User should be able to view a timetable which is broken down into the lesson slots of each day.	User is able to view a timetable which is broken down into the lesson slots of each day.	N/A.
Does the timetable have a clear, easy to understand structure?	Timetable should be structured in a table format, with columns representing days, and rows representing lesson slots.	Timetable is structured in a table format, with columns representing days, and rows representing lesson slots.	N/A.

When I tested this, the GUI was working as intended, with the columns adapting to the width of the cells in that column. For example, the column for Monday is a lot narrower than the column for Tuesday, because Tuesday has a longer subject name in the column in 'Computer Science'. This is further demonstrated in the Wednesday column, as it expands even wider than the Tuesday column, due to 'Further Mathematics' being a longer subject name.

There were no issues encountered during testing; the button for editing the timetable did not work, but this was not expected as I had not programmed it to perform any actions at this stage.

Review

My target for this stage was to create a user interface for Timetable. This should have involved creating a table to display the lesson schedule of the user, so students can view which lessons they have in each day.

This target was achieved, with a timetable interface created which is clear to read, and it can be used exactly as intended.

Next Steps for Module

Like stage 1 and 2, the next steps for Timetable will be to add functionality to the 'Edit Timetable' button. When it is clicked, it should open a dialogue which allows them to edit the labels in each cell, potentially using a dropdown menu of their subjects using data linked from My Subjects.

Stage 4: User Navigation

Objectives

Now that the initial GUIs for the most important parts of the program have been created, I need to create a main program which will run these user interfaces as modules. This is because it is good programming practice to make projects modular, as modularity helps with organisation of development and adaptability of code.

My objectives for this stage are to create the main program, Student Planner, which should open a window (Agenda, My Subjects, or Timetable) depending on an input.

Prototype Program

The key part for developing this stage is to import the user interfaces as modules, then running the relevant code from these modules to open the user interfaces. This appeared to be relatively simple at first, but I had little experience with using PyQt5, so I did not know how to start this off.

Hence, I decided to learn more about interacting with modular programs in PyQt5, as well as in Python in general.

I decided to perform some research on what type of code would be used to open new windows from other Python files in PyQt. I found this question on a forum (https://stackoverflow.com/questions/27567208/how-do-i-open-sub-window-after-i-click-on-button-on-main-screen-in-pyqt4) from a user who wanted to launch a new window when a button was pressed.

This is something like what my program will achieve at a later stage, but it is not currently the exact same, as my user interfaces do not have any functionality. However, the premise of how to import the necessary functions from the modules and then use these modules in the main program remains the same.

I looked at each line in the program mentioned in the top answer to that question on the forum, and ensured that I understood how the program worked as a whole. Based on this, I made my own prototype program to open Agenda.

```
import sys

from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QMainWindow

from agenda import Ui_mwindow_agenda


# Set up the window for Agenda.
class AgendaWindow(QMainWindow, Ui_mwindow_agenda):
    def __init__(self):
        super().__init__()
        self.setupUi(self)


app = QtWidgets.QApplication(sys.argv)

window = AgendaWindow()
window.show()

sys.exit(app.exec_())
```

At the start of the program, I needed to import sys, as this allows me to use the code at the end of my program which exits out of Python.

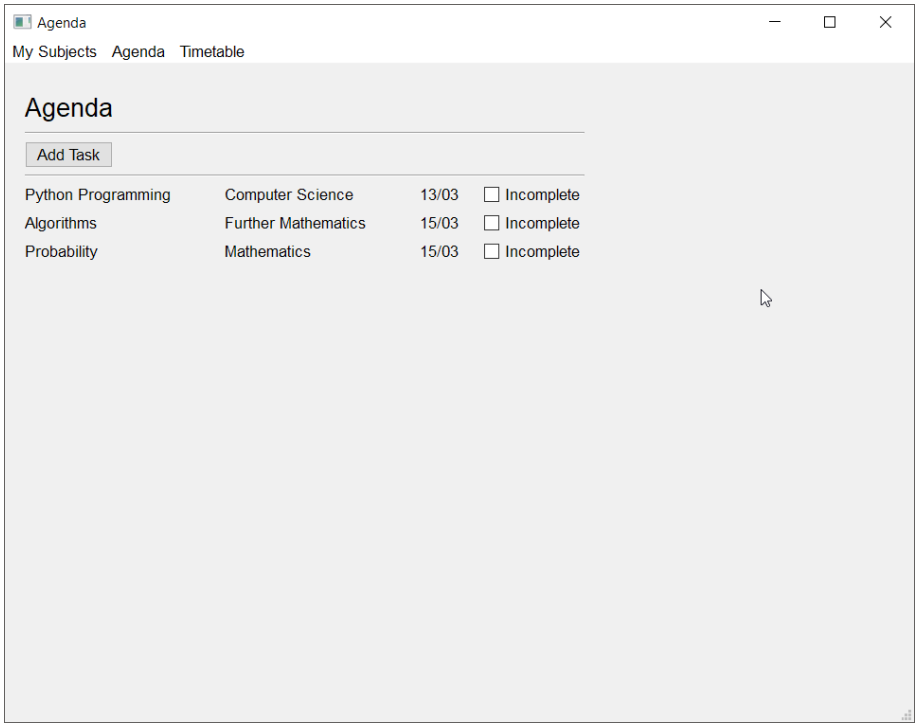
I also needed to import the modules from PyQt5 which are essential for my user interfaces to work properly, such as *QtCore*, *QtGui*, and *QtWidgets*.

Next, I needed to import *QMainWindow* from *PyQt5.QtWidgets*, as this provides a framework for building an application's user interface, as it provides a set of UI elements to create desktop user interfaces. It will enable the classes to function by inheriting from *QMainWindow* and the window it wants to open.

Finally, to gain access to the window I wanted to open, I needed to import the class of the window from that Python module. In this case, I imported from *agenda* to open the agenda window.

To set up the window for the module I want, I had to create a class which inherits from *QMainWindow* and the class from that module. In this case, the class from that module would be *Ui_mwindow_agenda*. The class involves using a constructor for the class, which includes using *super()* to gain access to inherited methods, and performing the *setupUi* method to set up all the parts of the user interface, such as the widgets.

After this, I called upon the class I just created, and used *show()* to display the window from that class. *sys.exit* will close this window if the program is stopped.



I tested this prototype program to see whether it worked, and it did successfully open Agenda.

Development: Iteration #1 – Navigation

My prototype program largely meets the objective set at the start of this stage. This means that a lot of the code from the prototype program can be reused in this development iteration, although it must be adapted to meet the objective of opening different windows depending on a user input.

The same type of imports were used in this new program. However, to gain access to the various user interfaces as modules and then run them depending on a user input, I needed to import the classes for the user interfaces from each separate Python file. This was required to access the functions which set up each user interface, which were all the *setupUi* functions in the case of the code automatically generated using pyuic5.

I created equivalent classes of *AgendaWindow* for the other two windows I had created in stages 2 and 3, and I called these classes *MySubjectsWindow* and *TimetableWindow*, which inherit from *Ui_mwindow_my_subjects* and *Ui_mwindow_timetable* respectively.

One problem I had with the development of this program was that I had no functionality in the user interfaces I had developed, so there was no direct way of obtaining a user input.

To solve this, I created a temporary workaround by allowing the user to manually select the window; the user will be asked to input which window they want to select, and the program will call upon the appropriate class before showing that class. If the user does not select a valid option, they will be notified that this is not a valid option, and asked to try again until they select a valid option.

An additional action I took was to include ‘mysubjects’ as a valid input, as it would likely be a common input from users, meaning this would improve the user experience by reducing the number of validation errors. For the same reason, I also converted the user input to lower case to prevent user inputs from being case sensitive.

```python
import sys

from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QMainWindow

from agenda import Ui_mwindow_agenda
from my_subjects import Ui_mwindow_my_subjects
from timetable import Ui_mwindow_timetable


# Set up the window for Agenda.
class AgendaWindow(QMainWindow, Ui_mwindow_agenda):
    def __init__(self):
        super().__init__()
        self.setupUi(self)


# Set up the window for My Subjects.
class MySubjectsWindow(QMainWindow, Ui_mwindow_my_subjects):
    def __init__(self):
        super().__init__()
        self.setupUi(self)


# Set up the window for Timetable.
class TimetableWindow(QMainWindow, Ui_mwindow_timetable):
    def __init__(self):
        super().__init__()
        self.setupUi(self)


app = QtWidgets.QApplication(sys.argv)

# Workaround for selecting window manually.
while True:
    try:
        select_main_window = str(
            input("Select window (agenda, my_subjects, or timetable): ")).lower()
    except ValueError:
        continue
    else:
        if select_main_window == "agenda":
            main_window = AgendaWindow()
        elif select_main_window == "my_subjects" or select_window == "mysubjects":
            main_window = MySubjectsWindow()
        elif select_main_window == "timetable":
            main_window = TimetableWindow()
        else:
            print("This is not a valid selection. Please try again.\n")
            continue
        break

main_window.show()

sys.exit(app.exec_())
```

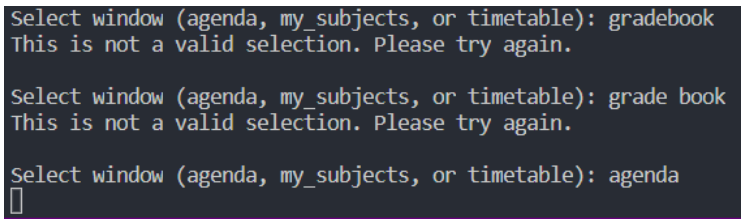
Testing: Iteration #1 – Navigation

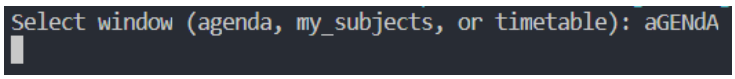
Now that user inputs were involved, I was able to use my test table for various types of data, which was useful for testing that my validation was working properly.

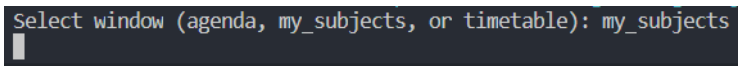
Input	Nature of Data	Data Validation	Is Data Accepted?	Output
Typed ‘agenda’.	Normal	Value check	Yes	Agenda window was opened.
Typed ‘my_subjects’.	Normal	Value check	Yes	My Subjects window was opened.
Typed ‘mysubjects’.	Normal	Value check	Yes	My Subjects window was opened.
Typed ‘timetable’.	Normal	Value check	Yes	Timetable window was opened.
Typed ‘aGENdA’.	Extreme	Value check	Yes	Agenda window was opened.
Typed ‘gradebook’.	Erroneous	Value check	Yes	Displayed ‘This is not a valid selection. Please try again.’

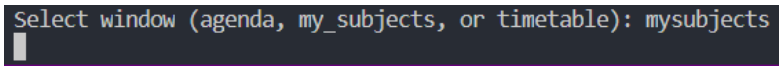
				and asked for reinput.
Typed ‘grade book’.	Erroneous	Value check	Yes	Displayed ‘This is not a valid selection. Please try again.’ and asked for reinput.
Typed an empty input.	Null	Value check	Yes	Displayed ‘This is not a valid selection. Please try again.’ and asked for reinput.

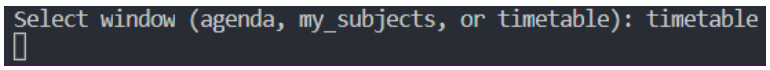
The testing I performed for my validation is shown below:

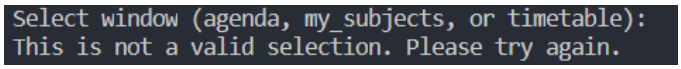












The test table for validation showed that my validation was working as intended. All valid inputs were processed and showed the appropriate window. All invalid inputs were rejected, displayed the error message, and asked for a reinput.

Next, I tested my program more generally by comparing it to the objectives set at the start of this stage.

Test	**Expected Result**	**Outcome**	**Further Actions**
Does the program open the appropriate window based on user input?	The program should open Agenda when ‘agenda’ is entered, My Subjects when ‘my_subjects’ or ‘mysubjects’ is entered, or Timetable when ‘timetable’ is entered (all inputs non-case sensitive).	The program opens Agenda when ‘agenda’ is entered, My Subjects when ‘my_subjects’ or ‘mysubjects’ is entered, or Timetable when ‘timetable’ is entered (all inputs non-case sensitive).	N/A.
How does the program	The program should reject invalid inputs,	The program rejects invalid inputs, display an error	N/A.

handle invalid inputs?	display an error message, and ask for a reinput.	message, and asks for a reinput. (Refer to validation test table for results.)	

My program passed all of the validation tests and general tests with no issues, meaning no further development was required in this stage.

Review

My objective for this stage was to create a main program which handled the opening of windows based on user inputs.

This was successfully created, resulting in more modular software, meaning my software has been made easier to organise and more adaptable. Despite encountering a problem in the lack of button functionality for user inputs, I improvised a solution which can be changed later once better functionality has been achieved in my modules.

Next Steps for Module

As aforementioned, the next steps for this module will be updating the source of user inputs once button functionality has been achieved in other modules, which will enable the user to navigate my software completely within the user interfaces.

Stage 5: Adding Tasks

Objectives

As mentioned in the Next Steps for Module section in stage 1, the next phase of development will be focused towards developing some basic functionality for adding tasks to Agenda using connected buttons to get closer to a fully functioning agenda. This will be useful for further integrating my software into one package.

My objectives for this stage are to create a GUI in the form of a dialog window for Agenda to create a new task, which should give the user an interface to enter the task details of their new task (task title, subject, and due date), and connect this to the ‘Add Task’ button on the Agenda to open this dialog when the button is pressed.

Prototype Program

The prototype program for this stage was the same as the one in the main program in stage 4, as it involves similar programming skills.

I did perform some additional research to build on the skills I learned in stage 4, but this was very brief, and it was part of the formal development process.

Development: Iteration #1 – User Interface

To start with, I needed to create a user interface through Qt Creator to provide the user with the ability to add new tasks to the agenda. As this screen for adding tasks will be a pop-up window rather than an entirely different part of the student planner, I created this window as a dialog box.

The idea is that the user will navigate to the agenda, press the button to add a task, enter the information into this dialog box, and then confirm the changes or cancel to exit this dialog box.

Unlike with the GUIs created in the first three stages, I avoided using a scroll area, as it will not be necessary for the elements I am using here. The only element which could possibly cause scrolling to be necessary would be the textbox for inputting notes if this is added in the future, but this would be an element-exclusive scroll area, where only that text box would be scrolling (as opposed to the whole user interface).

I used a vertical layout for all of the elements of the user interface, as they should mostly be structured vertically. A label is used at the top as a title of the window to indicate what the dialog box is for (adding a new task). This is placed above a horizontal line, which is used to create separation in the user interface, making the user interface easier on the eyes for the user.

A form layout holds the widgets related to the user inputs. The user will be able to add a task by inputting the task title, subject, and due date. These three input methods are all labelled.

To input the task title, a line edit widget has been included, as the task title will only be one line, making this the most appropriate input medium.

Subject will be able to be selected through a dropdown menu (also known as a combo box), because this input is likely to be repeated for many tasks, and it will be faster for most people to use a dropdown menu than manually type the subject for each task.

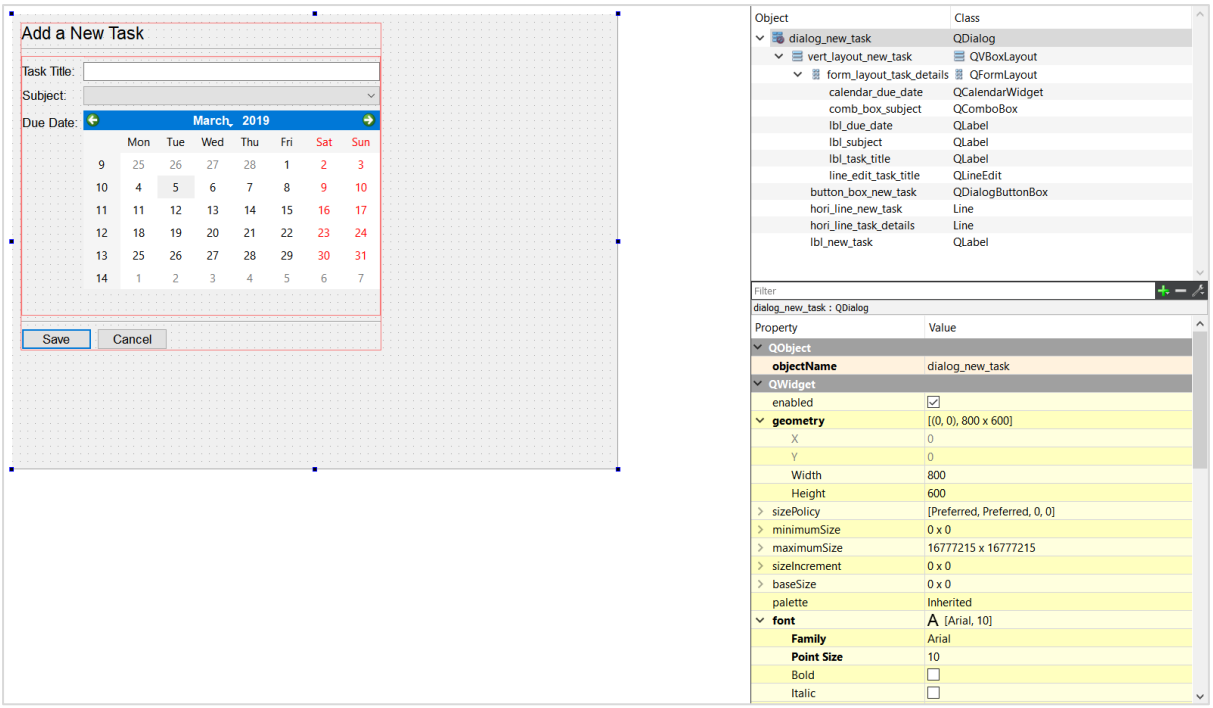
For the due date, a calendar widget has been used to enable the user to pick a due date visually. This is useful because it is easier and quicker to select a date by considering days in a week as opposed to the numerical day of the month, so users will be able to add more tasks in less time. In this particular context, it will also be useful because homework assignments are often due for the same day in the next week, so the user will know which day to look for and select.

At a later stage, a feature may be added to allow the user to add notes about their task to the task details, but this will not be added for now, as the goal is to get basic functionality in the agenda.

I used another horizontal line below the form to create further separation between the form and the buttons.

Below this horizontal line, a button box is used. It contains the two buttons in the user interface which allow the user to exit out of the dialog box. Once I add functionality to the user interface, the ‘Save’ button will add the task to the agenda with the details the user inputted (as long as all required details have been inputted). The ‘Cancel’ button will exit the dialog box without adding the task to the agenda.

After I finished designing the user interface in Qt Creator, I compiled the code using pyuic5, as shown in stage 1.




```python
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'add_task.ui'
#
# Created by: PyQt5 UI code generator 5.11.3
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_dialog_new_task(object):
    def setupUi(self, dialog_new_task):
        dialog_new_task.setObjectName("dialog_new_task")
        dialog_new_task.resize(800, 600)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        dialog_new_task.setFont(font)
        self.layoutWidget = QtWidgets.QWidget(dialog_new_task)
        self.layoutWidget.setGeometry(QtCore.QRect(11, 11, 477, 433))
        self.layoutWidget.setObjectName("layoutWidget")
        self.vert_layout_new_task = QtWidgets.QVBoxLayout(self.layoutWidget)
        self.vert_layout_new_task.setContentsMargins(0, 0, 0, 0)
        self.vert_layout_new_task.setObjectName("vert_layout_new_task")
        self.lbl_new_task = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setPointSize(14)
        self.lbl_new_task.setFont(font)
        self.lbl_new_task.setObjectName("lbl_new_task")
        self.vert_layout_new_task.addWidget(self.lbl_new_task)
        self.hori_line_new_task = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_new_task.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_new_task.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_new_task.setObjectName("hori_line_new_task")
        self.vert_layout_new_task.addWidget(self.hori_line_new_task)
        self.form_layout_task_details = QtWidgets.QFormLayout()
        self.form_layout_task_details.setObjectName("form_layout_task_details")
        self.lbl_task_title = QtWidgets.QLabel(self.layoutWidget)
```

```
        self.lbl_task_title.setObjectName("lbl_task_title")
        self.form_layout_task_details.setWidget(1, QtWidgets.QFormLayout.LabelRole, self.lbl_task_title)
        self.line_edit_task_title = QtWidgets.QLineEdit(self.layoutWidget)
        self.line_edit_task_title.setObjectName("line_edit_task_title")
        self.form_layout_task_details.setWidget(1, QtWidgets.QFormLayout.FieldRole, self.line_edit_task_title)
        self.lbl_subject = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_subject.setObjectName("lbl_subject")
        self.form_layout_task_details.setWidget(2, QtWidgets.QFormLayout.LabelRole, self.lbl_subject)
        self.comb_box_subject = QtWidgets.QComboBox(self.layoutWidget)
        self.comb_box_subject.setFrame(True)
        self.comb_box_subject.setObjectName("comb_box_subject")
        self.form_layout_task_details.setWidget(2, QtWidgets.QFormLayout.FieldRole, self.comb_box_subject)
        self.lbl_due_date = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_due_date.setObjectName("lbl_due_date")
        self.form_layout_task_details.setWidget(3, QtWidgets.QFormLayout.LabelRole, self.lbl_due_date)
        self.calendar_due_date = QtWidgets.QCalendarWidget(self.layoutWidget)
        self.calendar_due_date.setObjectName("calendar_due_date")
        self.form_layout_task_details.setWidget(3, QtWidgets.QFormLayout.FieldRole, self.calendar_due_date)
        self.vert_layout_new_task.addLayout(self.form_layout_task_details)
        self.hori_line_task_details = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_task_details.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_task_details.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_task_details.setObjectName("hori_line_task_details")
        self.vert_layout_new_task.addWidget(self.hori_line_task_details)
        self.button_box_new_task = QtWidgets.QDialogButtonBox(self.layoutWidget)
        self.button_box_new_task.setOrientation(QtCore.Qt.Horizontal)
        self.button_box_new_task.setStandardButtons(QtWidgets.QDialogButtonBox.Cancel|QtWidgets.QDialogButtonBox.Save)
        self.button_box_new_task.setObjectName("button_box_new_task")
        self.vert_layout_new_task.addWidget(self.button_box_new_task, 0, QtCore.Qt.AlignLeft)

        self.retranslateUi(dialog_new_task)
        self.button_box_new_task.rejected.connect(dialog_new_task.reject)
        self.button_box_new_task.accepted.connect(dialog_new_task.accept)
        QtCore.QMetaObject.connectSlotsByName(dialog_new_task)

    def retranslateUi(self, dialog_new_task):
        _translate = QtCore.QCoreApplication.translate
        dialog_new_task.setWindowTitle(_translate("dialog_new_task", "Add Task"))
```

```
        self.lbl_new_task.setText(_translate("dialog_new_task", "Add a New Task"))
        self.lbl_task_title.setText(_translate("dialog_new_task", "Task Title:"))
        self.lbl_subject.setText(_translate("dialog_new_task", "Subject:"))
        self.lbl_due_date.setText(_translate("dialog_new_task", "Due Date:"))


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    dialog_new_task = QtWidgets.QDialog()
    ui = Ui_dialog_new_task()
    ui.setupUi(dialog_new_task)
    dialog_new_task.show()
    sys.exit(app.exec_())
```

Once the user interface had been designed and compiled into Python code, I started development on the code for Agenda to connect the button to the dialog I had just created. This involved editing the code from the Python file for Agenda I created in stage 1.

When I created the main program in the previous stage, it involved importing modules and opening windows from these modules. This meant I was able to apply some of the skills I had learned from that stage to this scenario.

However, I was unsure about how to connect a button to the opening of a new window from another module, so I performed some search on this, and found a question on Stack Overflow which was querying the same action (https://stackoverflow.com/questions/36768033/pyqt-how-to-open-new-window). This showed me that you need to connect the button's click to a function, and program the function of the button to connect to the class which sets up that window.

First, I imported *QDialog* from *PyQt5.QtWidgets* for the same reason as I imported *QMainWindow* in stage 4; it will be used for inheritance in the class which sets up the dialog for adding a new task. The key distinction is that in stage 4, I was setting up a new main window, whereas in this stage, I am setting up a new dialog window instead.

I also imported the class *Ui_dialog_new_task* from *add_task.py* to gain access to that class from the Python file for the dialog. This was important, as it connected the two windows.

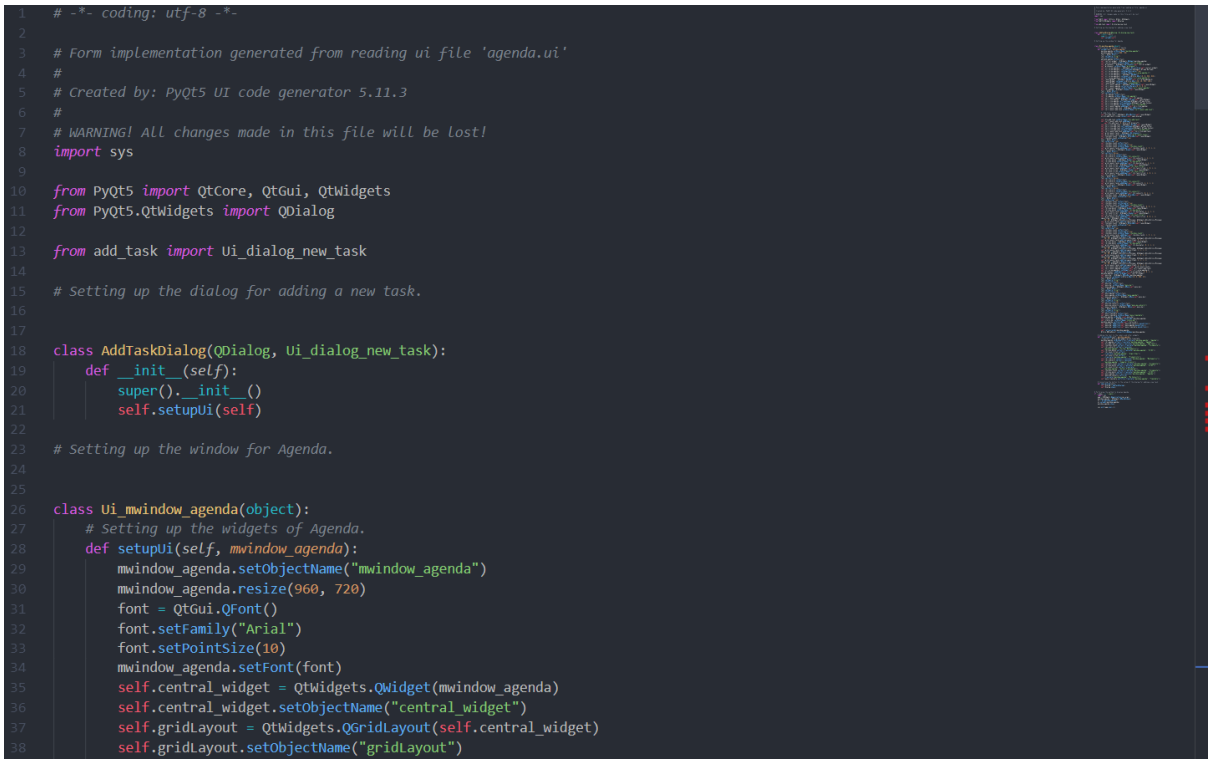
I created a class for setting up the widgets in the window called *AddTaskDialog*. It works by inheriting from *QDialog* and *Ui_dialog_new_task*, and this is the class which will be accessed when the button is pressed.

Next, in the class *Ui_mwindow_agenda*, which is the class for setting up Agenda, I searched the code for the part where the button widget (*btn_add_task*) is set up in the setupUi method. Then, in the next line I programmed it to perform the method *open_dialog* when the button is clicked. For readability, I added new lines above and below this part of code.

Finally, I defined the method *open_dialog* later in the class. In this method, I defined *self.Dialog* to the class which sets up the dialog, and then used the *show()* function to open this window.

It should be noted that I did not add any validation for the user inputs at this stage, as this would occur when tasks are added to the agenda. However, the line edit widget will use text scrolling with the arrow keys when the text cursor is active, so it will still be compatible with inputs which are beyond the maximum size the widget can display at once.

All these changes resulted in the following code for Agenda:



```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'agenda.ui'
#
# Created by: PyQt5 UI code generator 5.11.3
#
# WARNING! All changes made in this file will be lost!
import sys

from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QDialog

from add_task import Ui_dialog_new_task

# Setting up the dialog for adding a new task.


class AddTaskDialog(QDialog, Ui_dialog_new_task):
    def __init__(self):
        super().__init__()
        self.setupUi(self)

# Setting up the window for Agenda.


class Ui_mwindow_agenda(object):
    # Setting up the widgets of Agenda.
    def setupUi(self, mwindow_agenda):
        mwindow_agenda.setObjectName("mwindow_agenda")
        mwindow_agenda.resize(960, 720)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        mwindow_agenda.setFont(font)
        self.central_widget = QtWidgets.QWidget(mwindow_agenda)
        self.central_widget.setObjectName("central_widget")
        self.gridLayout = QtWidgets.QGridLayout(self.central_widget)
        self.gridLayout.setObjectName("gridLayout")
```

```
        self.scrl_area_agenda_1 = QtWidgets.QScrollArea(self.central_widget)
        self.scrl_area_agenda_1.setFrameShape(QtWidgets.QFrame.NoFrame)
        self.scrl_area_agenda_1.setWidgetResizable(True)
        self.scrl_area_agenda_1.setObjectName("scrl_area_agenda_1")
        self.scrl_area_agenda_2 = QtWidgets.QWidget()
        self.scrl_area_agenda_2.setGeometry(QtCore.QRect(0, 0, 938, 648))
        self.scrl_area_agenda_2.setObjectName("scrl_area_agenda_2")
        self.layoutWidget = QtWidgets.QWidget(self.scrl_area_agenda_2)
        self.layoutWidget.setGeometry(QtCore.QRect(10, 16, 591, 183))
        self.layoutWidget.setObjectName("layoutWidget")
        self.vert_layout_agenda = QtWidgets.QVBoxLayout(self.layoutWidget)
        self.vert_layout_agenda.setContentsMargins(0, 0, 0, 0)
        self.vert_layout_agenda.setObjectName("vert_layout_agenda")
        self.lbl_agenda = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setPointSize(16)
        self.lbl_agenda.setFont(font)
        self.lbl_agenda.setObjectName("lbl_agenda")
        self.vert_layout_agenda.addWidget(self.lbl_agenda)
        self.hori_line_agenda = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_agenda.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_agenda.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_agenda.setObjectName("hori_line_agenda")
        self.vert_layout_agenda.addWidget(self.hori_line_agenda)
        self.vert_layout_add_task = QtWidgets.QVBoxLayout()
        self.vert_layout_add_task.setObjectName("vert_layout_add_task")

        # 'Add Task' button.
        self.btn_add_task = QtWidgets.QPushButton(self.layoutWidget)
        self.btn_add_task.setObjectName("btn_add_task")
        self.btn_add_task.clicked.connect(self.open_dialog)

        self.vert_layout_add_task.addWidget(
            self.btn_add_task, 0, QtCore.Qt.AlignLeft)
        self.hori_line_add_task = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_add_task.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_add_task.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_add_task.setObjectName("hori_line_add_task")
```

```
        self.vert_layout_add_task.addWidget(self.hori_line_add_task)
        self.grid_layout_tasks = QtWidgets.QGridLayout()
        self.grid_layout_tasks.setObjectName("grid_layout_tasks")
        self.checkbox_task3 = QtWidgets.QCheckBox(self.layoutWidget)
        self.checkbox_task3.setEnabled(True)
        font = QtGui.QFont()
        font.setKerning(True)
        self.checkbox_task3.setFont(font)
        self.checkbox_task3.setCheckable(True)
        self.checkbox_task3.setObjectName("checkbox_task3")
        self.grid_layout_tasks.addWidget(self.checkbox_task3, 2, 5, 1, 1)
        self.lbl_subject2 = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setItalic(False)
        self.lbl_subject2.setFont(font)
        self.lbl_subject2.setObjectName("lbl_subject2")
        self.grid_layout_tasks.addWidget(self.lbl_subject2, 1, 2, 1, 1)
        self.lbl_due_date2 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_due_date2.setObjectName("lbl_due_date2")
        self.grid_layout_tasks.addWidget(self.lbl_due_date2, 1, 4, 1, 1)
        self.lbl_task_title2 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_task_title2.setObjectName("lbl_task_title2")
        self.grid_layout_tasks.addWidget(self.lbl_task_title2, 1, 0, 1, 1)
        self.lbl_task_title3 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_task_title3.setObjectName("lbl_task_title3")
        self.grid_layout_tasks.addWidget(self.lbl_task_title3, 2, 0, 1, 1)
        self.lbl_subject3 = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setItalic(False)
        self.lbl_subject3.setFont(font)
        self.lbl_subject3.setObjectName("lbl_subject3")
        self.grid_layout_tasks.addWidget(self.lbl_subject3, 2, 2, 1, 1)
        self.lbl_subject1 = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setItalic(False)
        self.lbl_subject1.setFont(font)
        self.lbl_subject1.setObjectName("lbl_subject1")
        self.grid_layout_tasks.addWidget(self.lbl_subject1, 0, 2, 1, 1)
```

```
        self.checkbox_task1 = QtWidgets.QCheckBox(self.layoutWidget)
        self.checkbox_task1.setEnabled(True)
        font = QtGui.QFont()
        font.setKerning(True)
        self.checkbox_task1.setFont(font)
        self.checkbox_task1.setCheckable(True)
        self.checkbox_task1.setObjectName("checkbox_task1")
        self.grid_layout_tasks.addWidget(self.checkbox_task1, 0, 5, 1, 1)
        self.lbl_due_date3 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_due_date3.setObjectName("lbl_due_date3")
        self.grid_layout_tasks.addWidget(self.lbl_due_date3, 2, 4, 1, 1)
        self.lbl_task_title1 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_task_title1.setObjectName("lbl_task_title1")
        self.grid_layout_tasks.addWidget(self.lbl_task_title1, 0, 0, 1, 1)
        spacerItem = QtWidgets.QSpacerItem(
            50, 20, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Minimum)
        self.grid_layout_tasks.addItem(spacerItem, 0, 1, 1, 1)
        self.checkbox_task2 = QtWidgets.QCheckBox(self.layoutWidget)
        self.checkbox_task2.setEnabled(True)
        font = QtGui.QFont()
        font.setKerning(True)
        self.checkbox_task2.setFont(font)
        self.checkbox_task2.setCheckable(True)
        self.checkbox_task2.setObjectName("checkbox_task2")
        self.grid_layout_tasks.addWidget(self.checkbox_task2, 1, 5, 1, 1)
        spacerItem1 = QtWidgets.QSpacerItem(
            50, 20, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Minimum)
        self.grid_layout_tasks.addItem(spacerItem1, 0, 3, 1, 1)
        self.lbl_due_date1 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_due_date1.setObjectName("lbl_due_date1")
        self.grid_layout_tasks.addWidget(self.lbl_due_date1, 0, 4, 1, 1)
        spacerItem2 = QtWidgets.QSpacerItem(
            50, 20, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Minimum)
        self.grid_layout_tasks.addItem(spacerItem2, 1, 1, 1, 1)
        spacerItem3 = QtWidgets.QSpacerItem(
            50, 20, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Minimum)
        self.grid_layout_tasks.addItem(spacerItem3, 1, 3, 1, 1)
        spacerItem4 = QtWidgets.QSpacerItem(
```

```
            50, 20, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Minimum)
        self.grid_layout_tasks.addItem(spacerItem4, 2, 1, 1, 1)
        spacerItem5 = QtWidgets.QSpacerItem(
            50, 20, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Minimum)
        self.grid_layout_tasks.addItem(spacerItem5, 2, 3, 1, 1)
        self.vert_layout_add_task.addLayout(self.grid_layout_tasks)
        self.vert_layout_agenda.addLayout(self.vert_layout_add_task)
        self.scrl_area_agenda_1.setWidget(self.scrl_area_agenda_2)
        self.gridLayout.addWidget(self.scrl_area_agenda_1, 0, 0, 1, 1)
        mwindow_agenda.setCentralWidget(self.central_widget)
        self.menu_bar = QtWidgets.QMenuBar(mwindow_agenda)
        self.menu_bar.setGeometry(QtCore.QRect(0, 0, 960, 25))
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_bar.setFont(font)
        self.menu_bar.setObjectName("menu_bar")
        self.menu_agenda = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_agenda.setFont(font)
        self.menu_agenda.setObjectName("menu_agenda")
        self.menu_my_subjects = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_my_subjects.setFont(font)
        self.menu_my_subjects.setObjectName("menu_my_subjects")
        self.menu_timetable = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_timetable.setFont(font)
        self.menu_timetable.setObjectName("menu_timetable")
        mwindow_agenda.setMenuBar(self.menu_bar)
        self.status_bar = QtWidgets.QStatusBar(mwindow_agenda)
        self.status_bar.setObjectName("status_bar")
```

```
        mwindow_agenda.setStatusBar(self.status_bar)
        self.menu_bar.addAction(self.menu_my_subjects.menuAction())
        self.menu_bar.addAction(self.menu_agenda.menuAction())
        self.menu_bar.addAction(self.menu_timetable.menuAction())

        self.retranslateUi(mwindow_agenda)
        QtCore.QMetaObject.connectSlotsByName(mwindow_agenda)

    # Adding the text to the labels and other widgets.
    def retranslateUi(self, mwindow_agenda):
        _translate = QtCore.QCoreApplication.translate
        mwindow_agenda.setWindowTitle(_translate("mwindow_agenda", "Agenda"))
        self.lbl_agenda.setText(_translate("mwindow_agenda", "Agenda"))
        self.btn_add_task.setText(_translate("mwindow_agenda", "Add Task"))
        self.checkbox_task3.setText(_translate("mwindow_agenda", "Incomplete"))
        self.lbl_subject2.setText(_translate(
            "mwindow_agenda", "Further Mathematics"))
        self.lbl_due_date2.setText(_translate("mwindow_agenda", "15/03"))
        self.lbl_task_title2.setText(
            _translate("mwindow_agenda", "Algorithms"))
        self.lbl_task_title3.setText(
            _translate("mwindow_agenda", "Probability"))
        self.lbl_subject3.setText(_translate("mwindow_agenda", "Mathematics"))
        self.lbl_subject1.setText(_translate(
            "mwindow_agenda", "Computer Science"))
        self.checkbox_task1.setText(_translate("mwindow_agenda", "Incomplete"))
        self.lbl_due_date3.setText(_translate("mwindow_agenda", "15/03"))
        self.lbl_task_title1.setText(_translate(
            "mwindow_agenda", "Python Programming"))
        self.checkbox_task2.setText(_translate("mwindow_agenda", "Incomplete"))
        self.lbl_due_date1.setText(_translate("mwindow_agenda", "13/03"))
        self.menu_agenda.setTitle(_translate("mwindow_agenda", "Agenda"))
        self.menu_my_subjects.setTitle(
            _translate("mwindow_agenda", "My Subjects"))
        self.menu_timetable.setTitle(_translate("mwindow_agenda", "Timetable"))

```

```
    # Connecting the button to the setup of the dialog for adding a new task.
    def open_dialog(self):
        self.Dialog = AddTaskDialog()
        self.Dialog.show()


# Performing the method to display Agenda.
if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    mwindow_agenda = QtWidgets.QMainWindow()
    ui = Ui_mwindow_agenda()
    ui.setupUi(mwindow_agenda)
    mwindow_agenda.show()

    sys.exit(app.exec_())

```

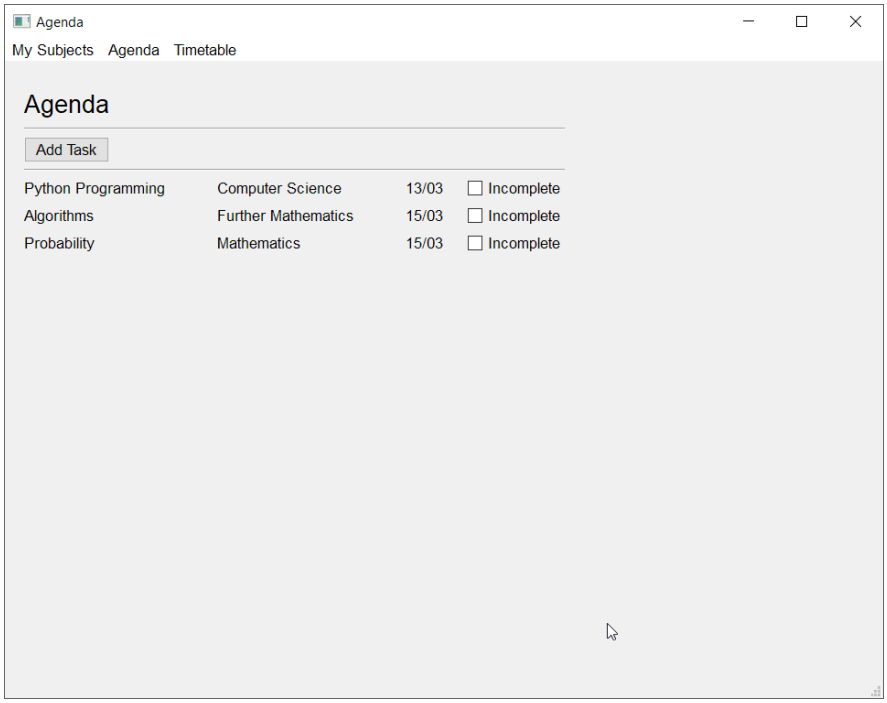
Testing: Iteration #1 – User Interface

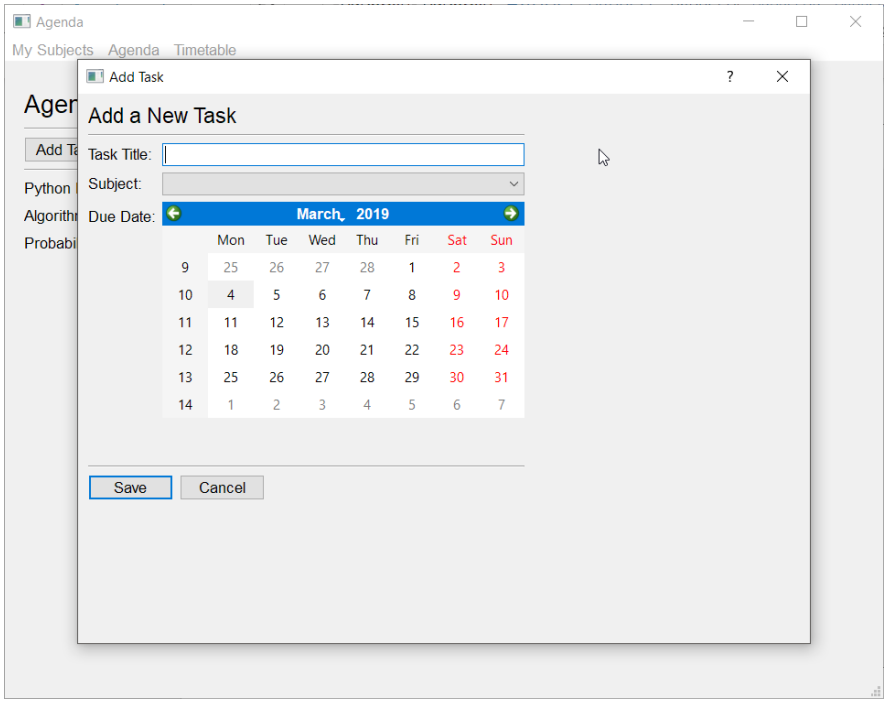
The possible user inputs at this point are limited for Agenda, but I performed tests on these limited range of user inputs to check that Agenda and the dialog within Agenda was working correctly.

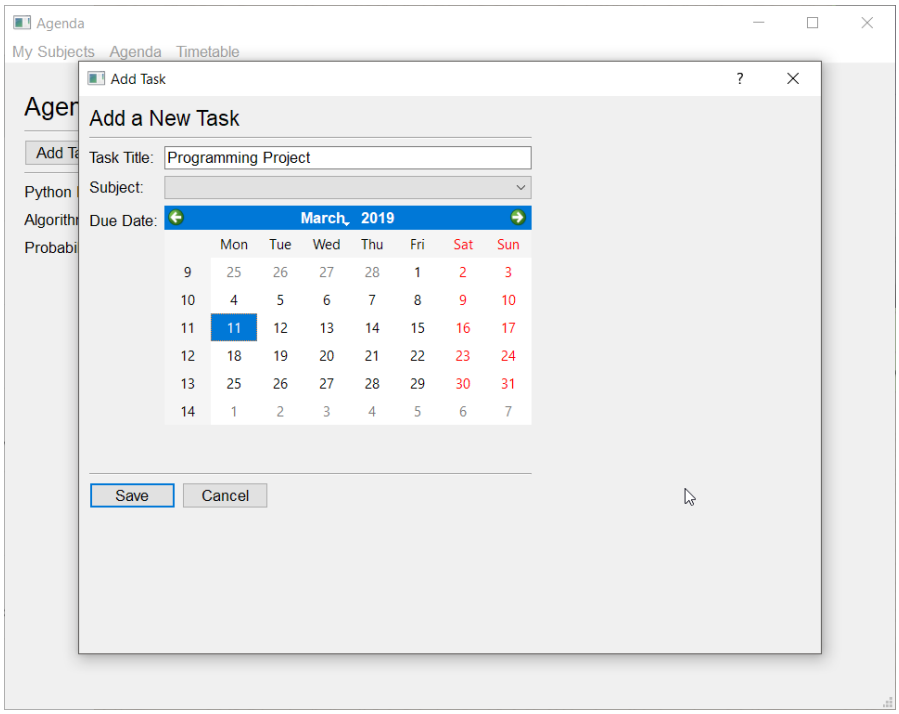
Input	Nature of Data	Data Validation	Is Data Accepted?	Output
Clicked on the 'Add Task' button in Agenda.	Normal	Button pressed	Yes	The Add Task dialog opened on top of Agenda.
Clicked on the 'Save' button in Add Task.	Normal	Button pressed	Yes	The Add Task dialog closed.
Clicked on the 'Cancel' button in Add Task.	Normal	Button pressed	Yes	The Add Task dialog closed.
Typed 'Programming Project' as the task title.	Normal	Button pressed	Yes	The line edit widget displays the full task title, as it fits within the line edit widget.

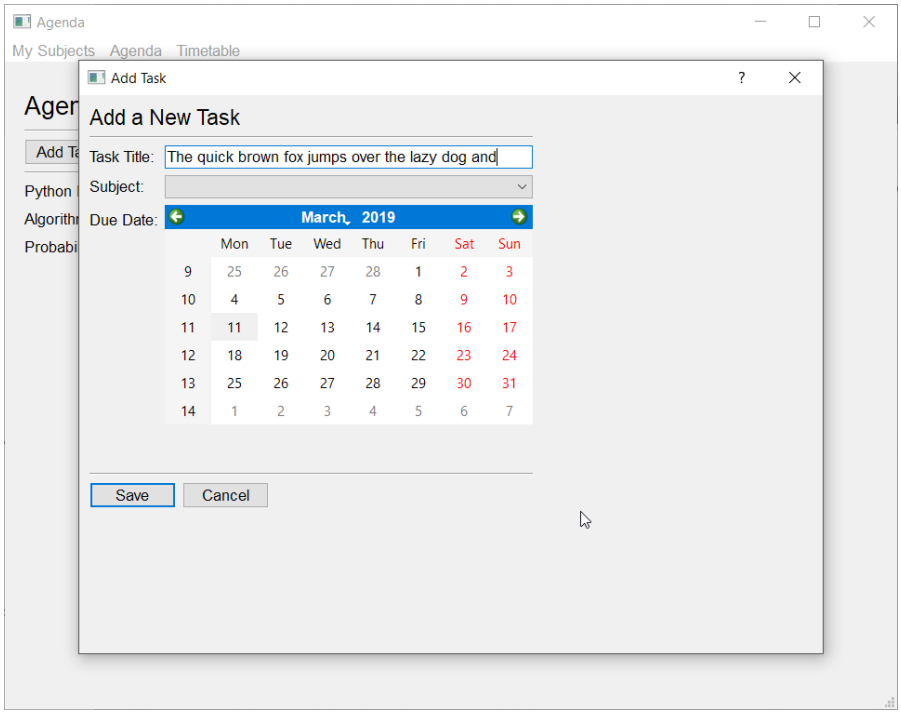
Clicked on the 11th March (next week) for the due date.	Normal	Button pressed	Yes	The calendar widget selects the 11th March as the date.
Typed 'The quick brown fox jumps over the lazy dog and' as the task title.	Extreme	N/A	Yes	The line edit widget displays the full task title, as it fits within the line edit widget, although it is near the maximum size.
Typed 'The quick brown fox jumps over the lazy dog and sprints quickly' as the task title.	Extreme	N/A	Yes	The line edit widget displays part of the task title, as it extends beyond the maximum size of the line edit widget. The rest of the task title can be viewed by using the arrow keys when using the text cursor in the widget.

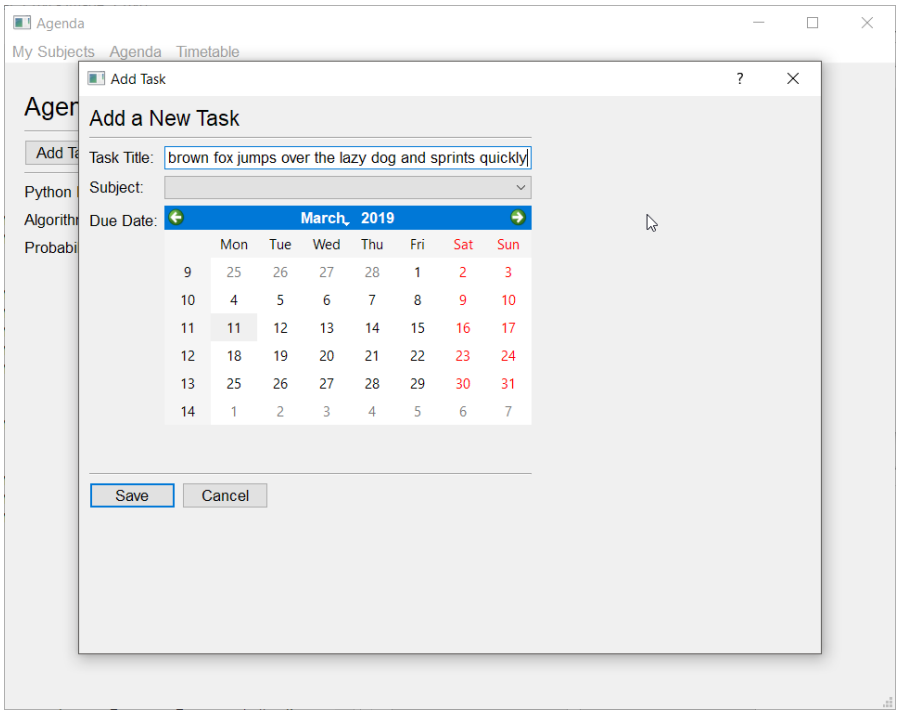
The testing process can be seen below.




Agenda
My Subjects Agenda Timetable
Add Task
Add a New Task
Task Title:
Subject:
Due Date:
March, 2019
Mon Tue Wed Thu Fri Sat Sun
9 25 26 27 28 1 2 3
10 4 5 6 7 8 9 10
11 11 12 13 14 15 16 17
12 18 19 20 21 22 23 24
13 25 26 27 28 29 30 31
14 1 2 3 4 5 6 7
Save
Cancel


Agenda
My Subjects Agenda Timetable
Add Task
Add a New Task
Task Title:
Programming Project
Subject:
Due Date:
March, 2019
Mon Tue Wed Thu Fri Sat Sun
9 25 26 27 28 1 2 3
10 4 5 6 7 8 9 10
11 11 12 13 14 15 16 17
12 18 19 20 21 22 23 24
13 25 26 27 28 29 30 31
14 1 2 3 4 5 6 7
Save
Cancel





It should be noted that although the two extreme task title inputs I used can be considered as erroneous for the adding of a task (as they are both beyond the 30 character limit which I intend to set in place for the task title), it has been determined to be extreme for this case, because no validation is being used at this stage. Both inputs are extreme, one input is near the maximum text the line edit widget can display at once, and the other is beyond this maximum.

Hence, there are no inputs possible which can be considered as invalid at this point, and the program outputted correctly according to the input.

After this, I performed more general tests to check whether I had met my objectives for this stage.

Test	Expected Result	Outcome	Further Actions
Is there a GUI for the user to add a new task?	There should be a GUI in the form of dialog window which allows the user to input the task title, subject, and due date.	There is a user interface in the form of a dialog window which allows the user to input the task title using a line editor, subject using a dropdown menu, and due date using a calendar interface.	N/A.
Does the program have a connected 'Add Task' button?	The 'Add Task' button should cause a dialog window for adding a new task to pop up.	The 'Add Task' button causes a dialog window for adding a new task to pop up.	N/A.

My program passed all the input and general tests, so I stopped development on this stage, as my objectives had been met.

Review

My objectives for this stage were to create a GUI for adding a new task to the Agenda, and connecting this to the 'Add Task' button on Agenda so it would open this dialog window when it is clicked.

I successfully met these objectives, creating a program which is modular and adaptable by using programming skills learned in previous stages, especially stage 4. The connected button links to the dialog window interface for adding a new task, so any changes to this dialog in later stages will not affect Agenda.

Next Steps for Module

The next steps for this module will be to enable the user to add new tasks to Agenda, which will involve performing validation on the task details before adding the task to the agenda if the task details pass this validation.

Stage 6: Adding Subjects

Objectives

Stage 6 will be a continuation from the progress made in step 5, as the user's subjects are required for the dropdown menu in the task adding dialog to work. In many ways, it will be similar, as I do not currently have a user interface for the user to add their subjects. However, I will need to perform research about how to permanently store the user's inputs so that my program saves their subjects after they are entered.

My objectives for this stage are to create a GUI in the form of a dialog window for the user to add subjects to My Subjects, connect the 'Add Subject' button to open this dialog window, and save the user's subject after they enter the correct details for it. All the user's subjects should be saved permanently, so data is retained even after a restart of the application. The user should also be able to delete existing subjects from the subject list.

Prototype Program

To save the user's subjects permanently, the data must be stored in a file. There are multiple mediums of performing this, but the simplest solution for this scenario would be using a .txt file. It is also possible to use a .json file for this, but it would be unnecessary, as the subject list only contains one item per line.

When the data about the user's subjects is stored, it would ideally be sorted alphanumerically in ascending order, because this would make for easier reading and better organisation for the user.

I decided to create a program which takes an input (randomly generated to test the system), save it to a text file, sort the contents of the text file alphabetically in ascending order, and rewrite it to the same text file.

```
import random

with open("test_list.txt", "w") as outfile:
    for i in range(0, 10):
        random_input = str(random.randint(0, 10))
        print(random_input)
        outfile.write(random_input + "\n")

print("---")

with open("test_list.txt", "r") as outfile:
    lines = outfile.readlines()
    lines.sort()
    with open("test_list_temp.txt", "w") as outfile:
        for line in lines:
            outfile.write(line)
        with open("test_list.txt", "w") as outfile:
            for line in lines:
                print(line.replace("\n", ""))
                outfile.write(line)

```

My program writes 10 random numbers from 0 to 10 inclusive to *test_list.txt*. It prints each of these numbers in the console as they are generated and written to the text file, so that testing can be more easily performed.

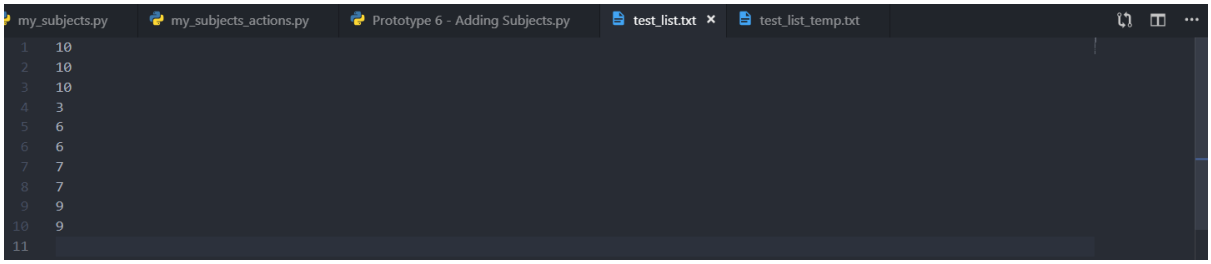
After that, the text file is reopened in read only mode. The lines are read, then sorted alphabetically.

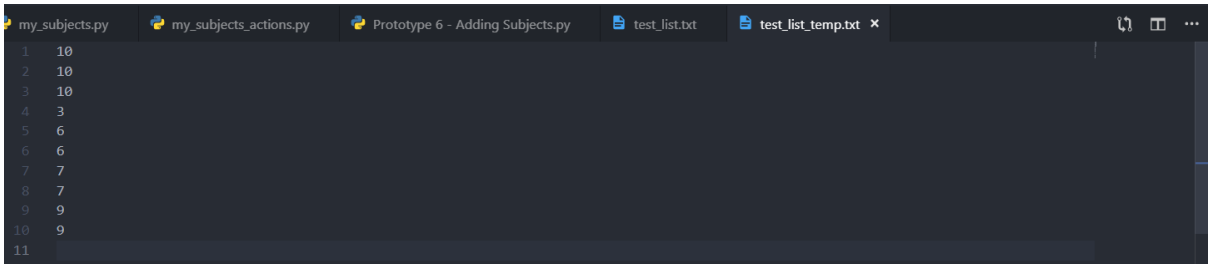
These lines are saved in sequence to a temporary text file, *test_list_temp.txt*. This is performed before overwriting the original text file, as it encourages data atomicity, preventing cases from occurring where data is only partially overwritten, in cases such as an unexpected program crash.

Once the program has finished writing to the temporary text file, it will overwrite the original text file with the sorted data.

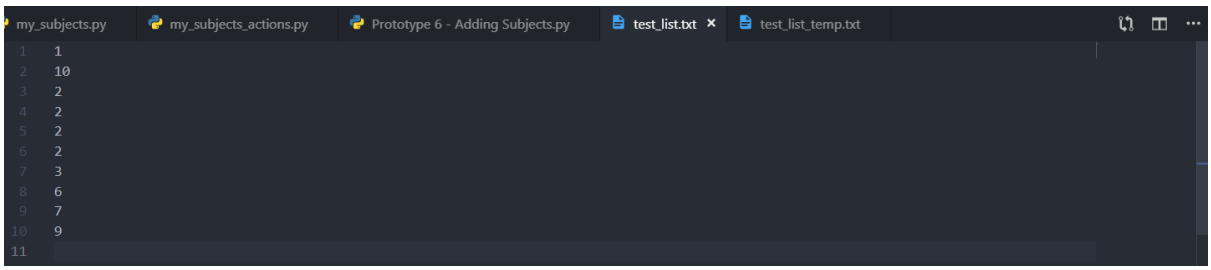
I tested my prototype program with two different automatically generated data sets to see whether it would work.

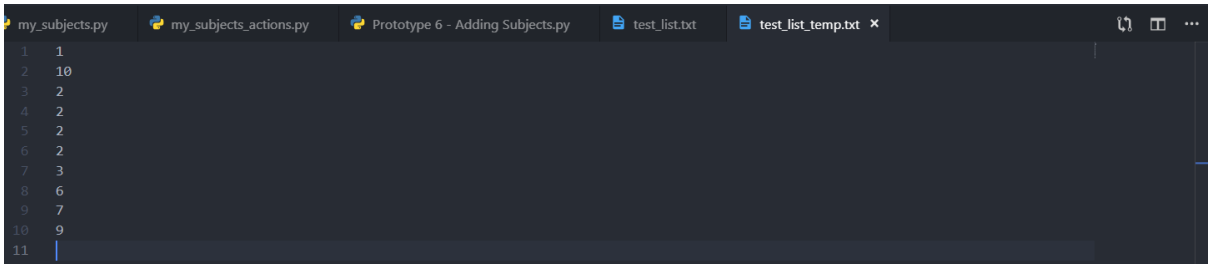
```
10
3
10
6
7
9
10
6
7
9
---
10
10
10
3
6
6
7
7
9
9
```






```
3
2
7
6
2
1
2
2
10
9
---
1
10
2
2
2
2
3
6
7
9
```





The two tests shown above demonstrate that the program works as intended. Both of the original inputs start off as unsorted, then they are sorted, saved to a temporary text file, and then copied over to overwrite the original text file.

Development: Iteration #1 – User Interface

The first thing I needed to do was create a user interface for adding new subjects, like I did for adding new tasks at the start of stage 5. This was performed in Qt Creator, and I created this window as a dialog window as opposed to a main window, because this window is intended to work as a pop-up window rather than a whole new section of the student planner.

In this screen, the plan is that the user will navigate to My Subjects, press the button to add a subject, enter the details in this dialog window, and then either save the changes or cancel the changes to exit this dialog window.

I avoided using a scroll area for this user interface, as it will not be necessary for this use case. There will be no scenarios where scrolling will be required, as the elements in the form will be independently scrollable, meaning scrolling will be enabled within that element without the need for scrolling the whole screen.

For all the elements in my user interface, I used a vertical layout, as the elements will stack on top of each other for the most part. A label is used at the top as a header to inform the user which window, they have navigated to (adding a new subject), and this is placed above a horizontal line, which is used to create separation between the header and the user input elements.

A form layout has been added to provide structure to the elements related to user inputs.

The user is able to input their subject name using a line edit widget. This is the most appropriate input medium because the input is only intended to be one line, and it will provide horizontal scrolling with the text cursor to prevent the page from being stretched.

For now, there is only one user input available, but more user input options may be provided for this dialog window in future stages.

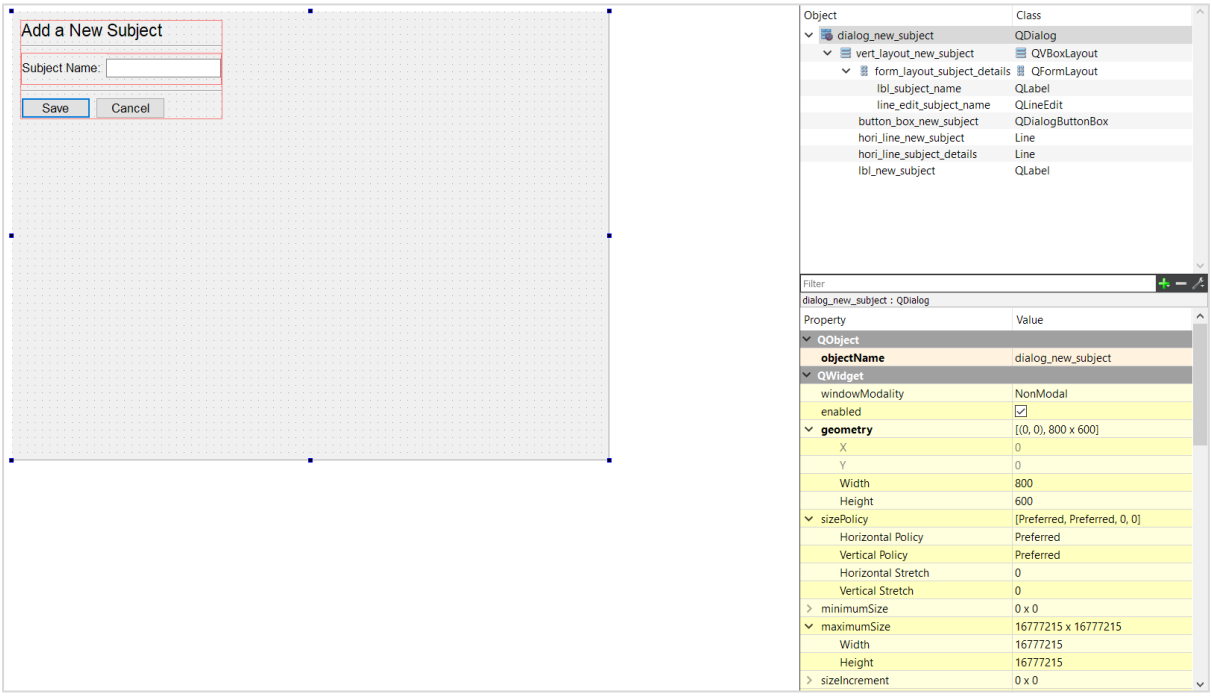
There is another horizontal line below the user inputs to create separation between these elements and the button box below it.

At the bottom of this user interface, there is the same button box used in the previous stage, containing buttons for the user to either save their new subject to My Subjects, or cancel their changes and return back to the My Subjects screen.

The usage of these element shows the similarity with the dialog window I created for adding a new task, as I have taken some of the same ideas from the previous stage and applied them to this dialog window where appropriate. This was a relatively simple user interface

compared to the previous one, because adding extra elements would not serve any purpose at this point.

Once I was happy with the user interface I created, I generated the Python code for it using pyuic5.




```python
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'add_subject.ui'
#
# Created by: PyQt5 UI code generator 5.11.3
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_dialog_new_subject(object):
    def setupUi(self, dialog_new_subject):
        dialog_new_subject.setObjectName("dialog_new_subject")
        dialog_new_subject.resize(800, 600)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        dialog_new_subject.setFont(font)
        self.layoutWidget = QtWidgets.QWidget(dialog_new_subject)
        self.layoutWidget.setGeometry(QtCore.QRect(11, 11, 271, 133))
        self.layoutWidget.setObjectName("layoutWidget")
        self.vert_layout_new_subject = QtWidgets.QVBoxLayout(self.layoutWidget)
        self.vert_layout_new_subject.setContentsMargins(0, 0, 0, 0)
        self.vert_layout_new_subject.setObjectName("vert_layout_new_subject")
        self.lbl_new_subject = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setPointSize(14)
        self.lbl_new_subject.setFont(font)
        self.lbl_new_subject.setObjectName("lbl_new_subject")
        self.vert_layout_new_subject.addWidget(self.lbl_new_subject)
        self.hori_line_new_subject = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_new_subject.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_new_subject.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_new_subject.setObjectName("hori_line_new_subject")
        self.vert_layout_new_subject.addWidget(self.hori_line_new_subject)
        self.form_layout_subject_details = QtWidgets.QFormLayout()
        self.form_layout_subject_details.setObjectName("form_layout_subject_details")
        self.lbl_subject_name = QtWidgets.QLabel(self.layoutWidget)
```

```
        self.lbl_subject_name.setObjectName("lbl_subject_name")
        self.form_layout_subject_details.setWidget(1, QtWidgets.QFormLayout.LabelRole, self.lbl_subject_name)
        self.line_edit_subject_name = QtWidgets.QLineEdit(self.layoutWidget)
        self.line_edit_subject_name.setObjectName("line_edit_subject_name")
        self.form_layout_subject_details.setWidget(1, QtWidgets.QFormLayout.FieldRole, self.line_edit_subject_name)
        self.vert_layout_new_subject.addLayout(self.form_layout_subject_details)
        self.hori_line_subject_details = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_subject_details.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_subject_details.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_subject_details.setObjectName("hori_line_subject_details")
        self.vert_layout_new_subject.addWidget(self.hori_line_subject_details)
        self.button_box_new_subject = QtWidgets.QDialogButtonBox(self.layoutWidget)
        self.button_box_new_subject.setOrientation(QtCore.Qt.Horizontal)
        self.button_box_new_subject.setStandardButtons(QtWidgets.QDialogButtonBox.Cancel|QtWidgets.QDialogButtonBox.Save)
        self.button_box_new_subject.setObjectName("button_box_new_subject")
        self.vert_layout_new_subject.addWidget(self.button_box_new_subject, 0, QtCore.Qt.AlignLeft)

        self.retranslateUi(dialog_new_subject)
        self.button_box_new_subject.rejected.connect(dialog_new_subject.reject)
        self.button_box_new_subject.accepted.connect(dialog_new_subject.accept)
        QtCore.QMetaObject.connectSlotsByName(dialog_new_subject)

    def retranslateUi(self, dialog_new_subject):
        _translate = QtCore.QCoreApplication.translate
        dialog_new_subject.setWindowTitle(_translate("dialog_new_subject", "Add Subject"))
        self.lbl_new_subject.setText(_translate("dialog_new_subject", "Add a New Subject"))
        self.lbl_subject_name.setText(_translate("dialog_new_subject", "Subject Name:"))


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    dialog_new_subject = QtWidgets.QDialog()
    ui = Ui_dialog_new_subject()
    ui.setupUi(dialog_new_subject)
    dialog_new_subject.show()
    sys.exit(app.exec_())
```

The same steps I took in stage 5 to connect a button to open a dialog window were needed in this stage to connect the 'Add Subject' button to open the dialog window for adding a new subject. This involved making changes to *my_subjects.py*, as this is the main window which will be connected to the new dialog window.

Once again, I imported *QDialog* from *PyQt5.QtWidgets*, as this will be used for inheritance in the class which I will use to set up the dialog window for adding a new subject.

I imported the class *Ui_dialog_new_subject* from *add_subject* so I could use that class in the Python file to open the dialog from My Subjects.

Then, I created a class for setting called *AddSubjectDialog*. It inherits *QDialog* and *Ui_dialog_new_subject*, which I had imported earlier. This class will be accessed when the button is pressed.

Like in the previous stage, I searched the code for My Subjects for the part where the button widget (*btn_add_subject*) is set up in the *setupUi* method. I added a new line underneath it to perform the *open_dialog_add_subject method* (changed from *open_dialog*, as My Subjects will have multiple dialog windows available). Then, I added new lines above and below this part of code to increased readability.

After this, I needed to define the method *open_dialog_add_subject* so that the button would have functionality. In this method, I defined *self.Dialog* to the class which sets up the dialog, *AddSubjectDialog*, and then used the *.show()* function to open this dialog window.

At this point, I did not have any validation for the user input, as the most important part was to ensure that the button was functioning correctly. Validation will be added later in this stage.

All these changes resulted in the following code for My Subjects:

```python
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'my_subjects.ui'
#
# Created by: PyQt5 UI code generator 5.11.3
#
# WARNING! All changes made in this file will be lost!
import sys

from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QDialog

from add_subject import Ui_dialog_new_subject

# Setting up the dialog for adding a new subject.


class AddSubjectDialog(QDialog, Ui_dialog_new_subject):
    def __init__(self):
        super().__init__()
        self.setupUi(self)


class Ui_mwindow_my_subjects(object):
    # Setting up the widgets of My Subjects.
    def setupUi(self, mwindow_my_subjects):
        mwindow_my_subjects.setObjectName("mwindow_my_subjects")
        mwindow_my_subjects.resize(960, 720)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        mwindow_my_subjects.setFont(font)
        self.central_widget = QtWidgets.QWidget(mwindow_my_subjects)
        self.central_widget.setObjectName("central_widget")
        self.gridLayout = QtWidgets.QGridLayout(self.central_widget)
        self.gridLayout.setObjectName("gridLayout")
        self.scrl_area_my_subjects_1 = QtWidgets.QScrollArea(
            self.central_widget)
        self.scrl_area_my_subjects_1.setFrameShape(QtWidgets.QFrame.NoFrame)
        self.scrl_area_my_subjects_1.setWidgetResizable(True)
        self.scrl_area_my_subjects_1.setObjectName("scrl_area_my_subjects_1")
        self.scrl_area_my_subjects_2 = QtWidgets.QWidget()
        self.scrl_area_my_subjects_2.setGeometry(QtCore.QRect(0, 0, 938, 648))
        self.scrl_area_my_subjects_2.setObjectName("scrl_area_my_subjects_2")
        self.layoutWidget = QtWidgets.QWidget(self.scrl_area_my_subjects_2)
        self.layoutWidget.setGeometry(QtCore.QRect(11, 11, 381, 200))
        self.layoutWidget.setObjectName("layoutWidget")
        self.vert_layout_my_subjects = QtWidgets.QVBoxLayout(self.layoutWidget)
        self.vert_layout_my_subjects.setContentsMargins(0, 0, 0, 0)
        self.vert_layout_my_subjects.setObjectName("vert_layout_my_subjects")
        self.lbl_my_subjects = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setPointSize(16)
        self.lbl_my_subjects.setFont(font)
        self.lbl_my_subjects.setObjectName("lbl_my_subjects")
        self.vert_layout_my_subjects.addWidget(self.lbl_my_subjects)
        self.hori_line_my_subjects = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_my_subjects.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_my_subjects.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_my_subjects.setObjectName("hori_line_my_subjects")
        self.vert_layout_my_subjects.addWidget(self.hori_line_my_subjects)

        # 'Add Subject' button.
        self.btn_add_subject = QtWidgets.QPushButton(self.layoutWidget)
        self.btn_add_subject.setObjectName("btn_add_subject")
        ui.btn_add_subject.clicked.connect(self.open_dialog_add_subject)

        self.vert_layout_my_subjects.addWidget(
            self.btn_add_subject, 0, QtCore.Qt.AlignLeft)
        self.hori_line_add_subject = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_add_subject.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_add_subject.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_add_subject.setObjectName("hori_line_add_subject")
        self.vert_layout_my_subjects.addWidget(self.hori_line_add_subject)
        self.hori_layout_subject1 = QtWidgets.QHBoxLayout()
        self.hori_layout_subject1.setObjectName("hori_layout_subject1")
```

```
        self.lbl_subject1 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_subject1.setObjectName("lbl_subject1")
        self.hori_layout_subject1.addWidget(
            self.lbl_subject1, 0, QtCore.Qt.AlignLeft)
        self.lbl_subject_name1 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_subject_name1.setObjectName("lbl_subject_name1")
        self.hori_layout_subject1.addWidget(self.lbl_subject_name1)
        spacerItem = QtWidgets.QSpacerItem(
            40, 20, QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Minimum)
        self.hori_layout_subject1.addItem(spacerItem)
        self.btn_view_details1 = QtWidgets.QPushButton(self.layoutWidget)
        self.btn_view_details1.setObjectName("btn_view_details1")
        self.hori_layout_subject1.addWidget(self.btn_view_details1)
        self.vert_layout_my_subjects.addLayout(self.hori_layout_subject1)
        self.hori_layout_subject2 = QtWidgets.QHBoxLayout()
        self.hori_layout_subject2.setObjectName("hori_layout_subject2")
        self.lbl_subject2 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_subject2.setObjectName("lbl_subject2")
        self.hori_layout_subject2.addWidget(
            self.lbl_subject2, 0, QtCore.Qt.AlignLeft)
        self.lbl_subject_name2 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_subject_name2.setObjectName("lbl_subject_name2")
        self.hori_layout_subject2.addWidget(self.lbl_subject_name2)
        spacerItem1 = QtWidgets.QSpacerItem(
            40, 20, QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Minimum)
        self.hori_layout_subject2.addItem(spacerItem1)
        self.btn_view_details2 = QtWidgets.QPushButton(self.layoutWidget)
        self.btn_view_details2.setObjectName("btn_view_details2")
        self.hori_layout_subject2.addWidget(self.btn_view_details2)
        self.vert_layout_my_subjects.addLayout(self.hori_layout_subject2)
        self.hori_layout_subject3 = QtWidgets.QHBoxLayout()
        self.hori_layout_subject3.setObjectName("hori_layout_subject3")
        self.lbl_subject3 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_subject3.setObjectName("lbl_subject3")
        self.hori_layout_subject3.addWidget(
            self.lbl_subject3, 0, QtCore.Qt.AlignLeft)
        self.lbl_subject_name3 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_subject_name3.setObjectName("lbl_subject_name3")
```

```
        self.hori_layout_subject3.addWidget(self.lbl_subject_name3)
        spacerItem2 = QtWidgets.QSpacerItem(
            40, 20, QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Minimum)
        self.hori_layout_subject3.addItem(spacerItem2)
        self.btn_view_details3 = QtWidgets.QPushButton(self.layoutWidget)
        self.btn_view_details3.setObjectName("btn_view_details3")
        self.hori_layout_subject3.addWidget(self.btn_view_details3)
        self.vert_layout_my_subjects.addLayout(self.hori_layout_subject3)
        self.scrl_area_my_subjects_1.setWidget(self.scrl_area_my_subjects_2)
        self.gridLayout.addWidget(self.scrl_area_my_subjects_1, 0, 0, 1, 1)
        mwindow_my_subjects.setCentralWidget(self.central_widget)
        self.menu_bar = QtWidgets.QMenuBar(mwindow_my_subjects)
        self.menu_bar.setGeometry(QtCore.QRect(0, 0, 960, 25))
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_bar.setFont(font)
        self.menu_bar.setObjectName("menu_bar")
        self.menu_my_subjects = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_my_subjects.setFont(font)
        self.menu_my_subjects.setObjectName("menu_my_subjects")
        self.menu_agenda = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_agenda.setFont(font)
        self.menu_agenda.setObjectName("menu_agenda")
        self.menu_timetable = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_timetable.setFont(font)
        self.menu_timetable.setObjectName("menu_timetable")
        mwindow_my_subjects.setMenuBar(self.menu_bar)
        self.status_bar = QtWidgets.QStatusBar(mwindow_my_subjects)
```

```
        self.status_bar.setObjectName("status_bar")
        mwindow_my_subjects.setStatusBar(self.status_bar)
        self.actionAgenda = QtWidgets.QAction(mwindow_my_subjects)
        self.actionAgenda.setObjectName("actionAgenda")
        self.menu_bar.addAction(self.menu_my_subjects.menuAction())
        self.menu_bar.addAction(self.menu_agenda.menuAction())
        self.menu_bar.addAction(self.menu_timetable.menuAction())

        self.retranslateUi(mwindow_my_subjects)
        QtCore.QMetaObject.connectSlotsByName(mwindow_my_subjects)

    # Adding the text to the labels and other widgets.
    def retranslateUi(self, mwindow_my_subjects):
        _translate = QtCore.QCoreApplication.translate
        mwindow_my_subjects.setWindowTitle(
            _translate("mwindow_my_subjects", "My Subjects"))
        self.lbl_my_subjects.setText(_translate(
            "mwindow_my_subjects", "My Subjects"))
        self.btn_add_subject.setText(_translate(
            "mwindow_my_subjects", "Add Subject"))
        self.lbl_subject1.setText(_translate(
            "mwindow_my_subjects", "Subject 1:"))
        self.lbl_subject_name1.setText(_translate(
            "mwindow_my_subjects", "Computer Science"))
        self.btn_view_details1.setText(_translate(
            "mwindow_my_subjects", "View Details"))
        self.lbl_subject2.setText(_translate(
            "mwindow_my_subjects", "Subject 2:"))
        self.lbl_subject_name2.setText(_translate(
            "mwindow_my_subjects", "Further Mathematics"))
        self.btn_view_details2.setText(_translate(
            "mwindow_my_subjects", "View Details"))
        self.lbl_subject3.setText(_translate(
            "mwindow_my_subjects", "Subject 3:"))
        self.lbl_subject_name3.setText(_translate(
            "mwindow_my_subjects", "Mathematics"))
        self.btn_view_details3.setText(_translate(
            "mwindow_my_subjects", "View Details"))
```

```
        self.menu_my_subjects.setTitle(_translate(
            "mwindow_my_subjects", "My Subjects"))
        self.menu_agenda.setTitle(_translate("mwindow_my_subjects", "Agenda"))
        self.menu_timetable.setTitle(_translate(
            "mwindow_my_subjects", "Timetable"))
        self.actionAgenda.setText(_translate("mwindow_my_subjects", "Agenda"))

    # Connecting the button to the setup of the dialog for adding a new subject.
    def open_dialog_add_subject(self):
        self.Dialog = AddSubjectDialog()
        self.Dialog.show()


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    mwindow_my_subjects = QtWidgets.QMainWindow()
    ui = Ui_mwindow_my_subjects()
    ui.setupUi(mwindow_my_subjects)
    mwindow_my_subjects.show()
    sys.exit(app.exec_())

```

Testing: Iteration #1 – User Interface

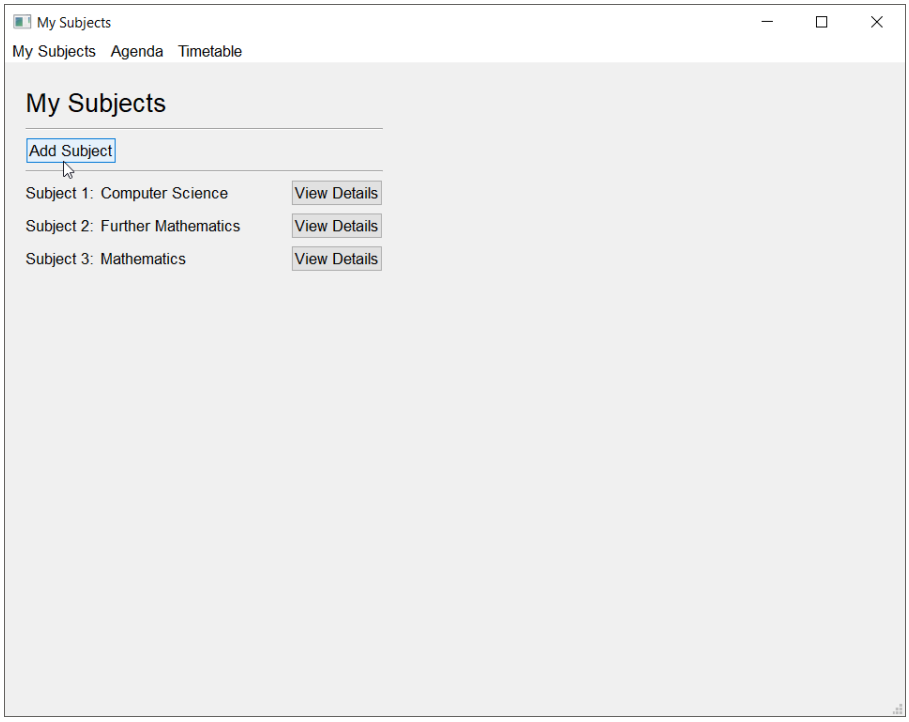
At this point, I had performed a significant amount of development, so I decided it would be a good idea to performing testing on the development even if I had not attempted to fulfil all the objectives yet.

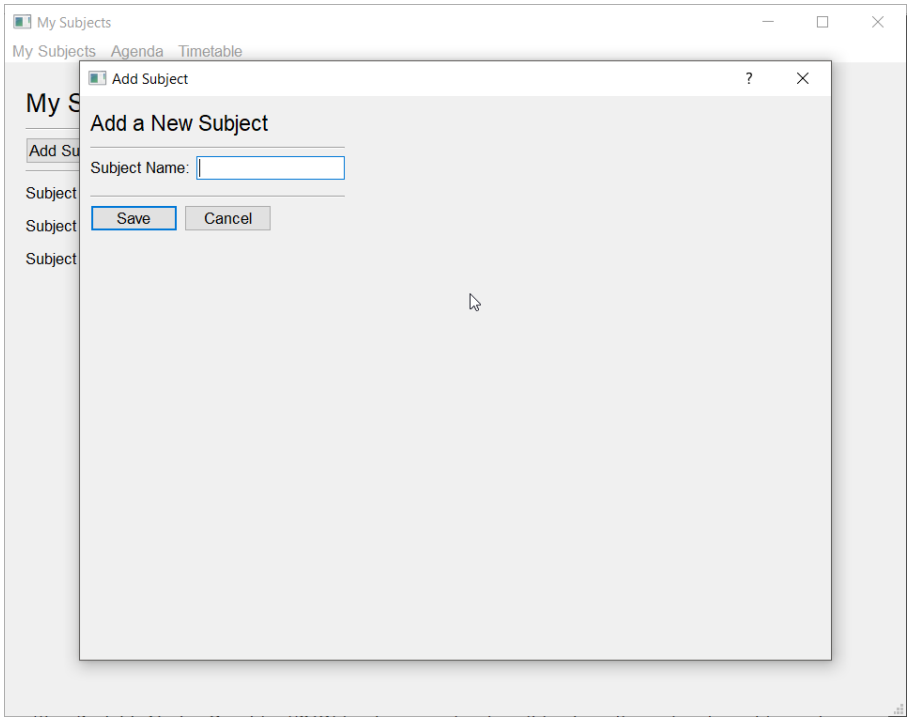
There is currently a limited range of potential user inputs for My Subjects, but I tested the user inputs which are available to check that data was being processed by My Subjects correctly.

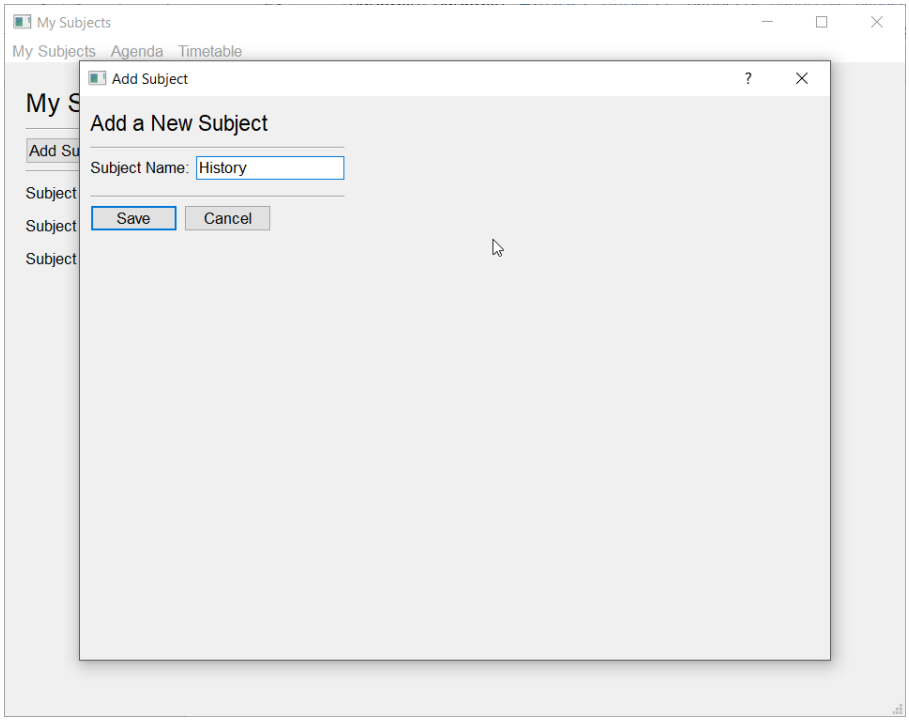
Input	Nature of Data	Data Validation	Is Data Accepted?	Output
Clicked on the 'Add Subject' button in My Subjects.	Normal	Button pressed	Yes	The Add Subject dialog opened on top of My Subjects.
Clicked on the 'Save' button in Add Subject.	Normal	Button pressed	Yes	The Add Subject dialog closed.

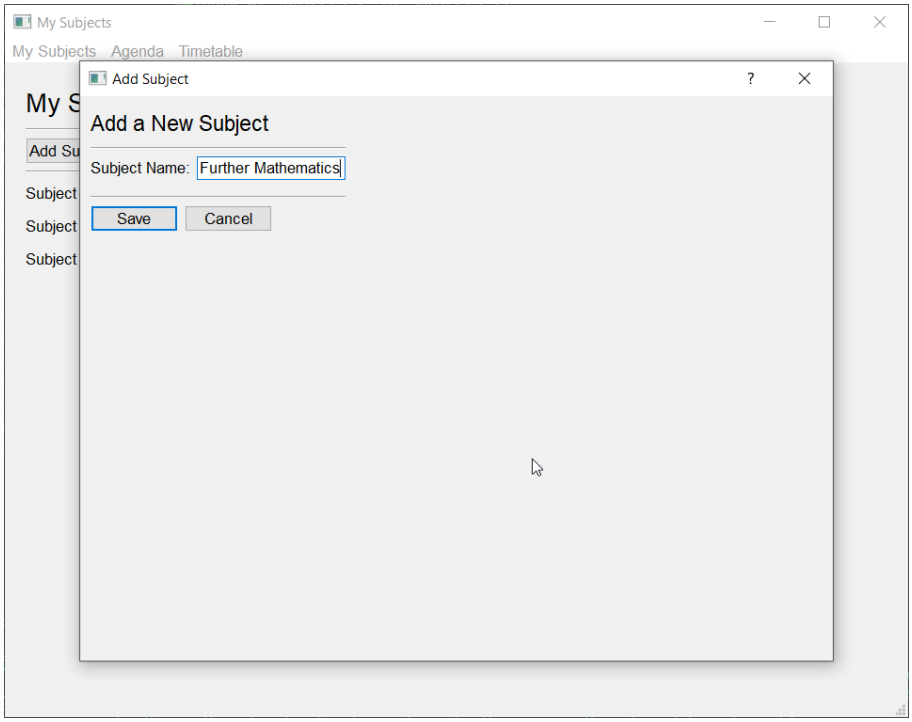
Clicked on the 'Cancel' button in Add Subject.	Normal	Button pressed	Yes	The Add Subject dialog closed.
Typed 'History' as the subject name.	Normal	N/A	Yes	The line edit widget displays the full subject name, as it fits within the line edit widget.
Typed 'Further Mathematics' as the subject name.	Extreme	N/A	Yes	The line edit widget displays the full subject name, as it fits within the line edit widget, although it is near the maximum size.
Typed 'Philosophy and Ethics' as the task title.	Extreme	N/A	Yes	The line edit widget displays part of the subject name, as it extends beyond the maximum size of the line edit widget. The rest of the subject name can be viewed by using the arrow keys when using the text cursor in the widget.

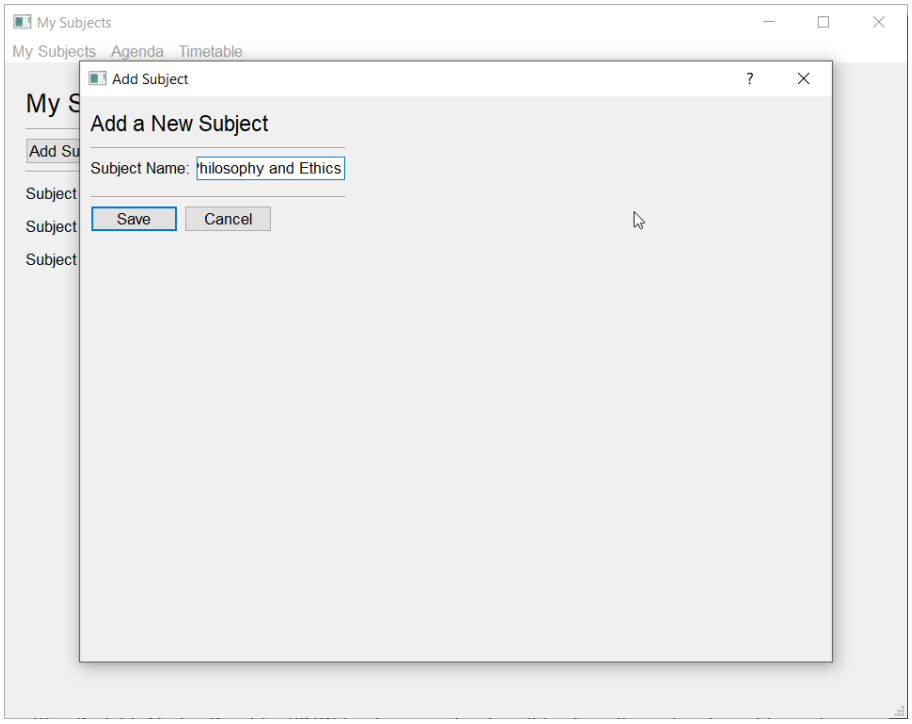
The testing process for the inputs of this development iteration can be seen below:




My Subjects
My Subjects Agenda Timetable
Add Subject
Add a New Subject
Subject Name:
Save
Cancel


My Subjects
My Subjects Agenda Timetable
Add Subject
Add a New Subject
Subject Name: History
Save
Cancel





After this, I performed more general testing by comparing the functions of my program to the objectives I set at the start of the stage. I had not attempted to complete all the objectives, but I compared my progress to the objectives I set so that I could clarify what actions I need to take in later development iterations.

Test	Expected Result	Outcome	Further Actions
Is there a GUI for the user to add a new subject?	There should be a GUI in the form of dialog window which allows the user to	There is a user interface in the form of a dialog window which allows the user to input the subject name using a line editor.	N/A.

	input the subject name.		
Does the program have a connected 'Add Subject' button?	The 'Add Subject' button should cause a dialog window for adding a new subject to pop up.	The 'Add Subject' button causes a dialog window for adding a new subject to pop up.	N/A.
Are subjects added using Add Subject saved to the list in My Subjects?	Subjects entered from the Add Subject dialog and saved should appear in the list of subjects in My Subjects in alphabetical ascending order.	Subjects entered from the Add Subject dialog are not added to the list of subjects in My Subjects.	Fetch the user's input from the line edit widget, and save this to a file for permanent storage.
Is the subject list in My Subjects retained permanently?	The subject list in My Subjects should be stored to a file so that it can be stored permanently and displayed when the program is next run.	The subject list in My Subjects is not stored permanently, so data is not retained.	Research into data storage solutions, so that data can be retained permanently.

In this development iteration, I met the objectives which I had aimed to complete, which were tested in the first and second tests from the test table above.

By considering the expected results for the remaining objectives, I gained a clearer insight into what I could do to fulfil these objectives in the next development iteration.

<u>Development: Iteration #2 – Adding Subjects</u>

To start this development iteration, I separated the operational code (the extra code which I added to the automatically generated code) from the automatically generated code for My Subjects. This resulted in it being separated into two files, *my_subjects_setup.py* for the automatically generated code, and *my_subjects.py* for the operational code.

I had to import *QMainWindow* and *Ui_mwindow_my_subjects* to ensure that the operational code was linking together with the generated code properly. I also changed the code for *if __name__* part to adapt to a different class being used for the main window, *MySubjectsWindow()*.

```
from PyQt5.QtWidgets import QDialog, QMainWindow

from add_subject_setup import Ui_dialog_new_subject
from my_subjects_setup import Ui_mwindow_my_subjects
```

```
class MySubjectsWindow(QMainWindow, Ui_mwindow_my_subjects):
    def __init__(self):
        super().__init__()
        self.setupUi(self)

        # Connects 'Add Subject' button to the Add Subject dialog.
        self.btn_add_subject.clicked.connect(self.open_dialog_add_subject)
```

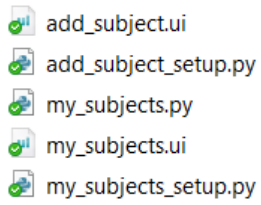
```
36      # Opens the dialog for the user to add a subject.
37      def open_dialog_add_subject(self):
38          self.Dialog = AddSubjectDialog()
39          # Connects 'Save' button to save the user input.
40          self.Dialog.button_box_new_subject.accepted.connect(self.save_subject)
41          self.Dialog.open()
```

```
66  if __name__ == "__main__":
67      import sys
68      app = QtWidgets.QApplication(sys.argv)
69      mwindow_my_subjects = MySubjectsWindow()
70      mwindow_my_subjects.show()
71      sys.exit(app.exec_())
```

I also moved the operational code for Add Subject into the operational code for My Subjects, as this makes the code cleaner. It makes better sense to have code for the dialog integrated with the code for the main window which the dialog comes from. Based on this, I renamed the automatically generated code for Add Subject from *add_subject.py* to *add_subject_setup.py* to make it clear that it only contains code created by pyuic5.

I did not have to create a new file for adding a subject, as the operational code is all contained within the main operational program, *my_subjects.py*.

This will make future development and maintenance easier, and it is also considered better practice in coding conventions, as it makes my application more modular.

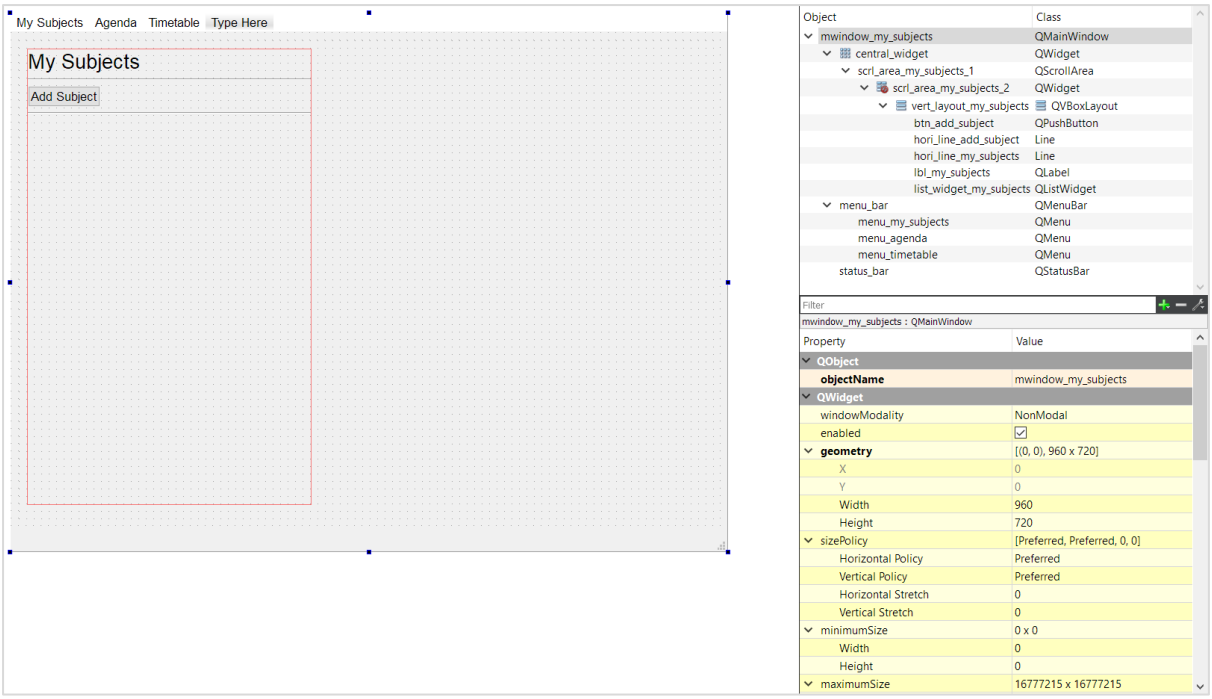


After I had finished the changes to the Add Subject module, I realised that my method of displaying the list of subjects would not be appropriate, as I was using a fixed number of labels. This would cause problems if the user wanted to add more or less than three subjects.

Instead of using labels, I decided that using a *QListWidget* would be better for my needs, as I would be storing a list of subjects. This would also make it easier for me to sort the list alphabetically in ascending order.

To start with, I updated the user interface in Qt Creator to use this *QListWidget* (named *list_widget_my_subjects*). I also removed the 'View Details' buttons as they would cause issues with the implementation of the *QListWidget*; this may be reimplemented in a different form later on.

Once I made the changes, I converted the user interface to a Python file using pyuic5, and then transferred the additional code from the previous development iteration.



Next, I needed a way to save the user inputs for both the user's subjects and the user's details of each new task. For the user to be able to create a new task, they must be able to select a subject from the dropdown menu, so saving the user's added subjects will be prioritised first; adding tasks will not be covered in this stage, but at a later stage.

When the window is initialised, the program will go through the process of populating the list widget. It opens the text file which stores the subject list, *subject_list.txt*, in read mode. My program reads all the lines of the file and prints the lines and length of the subject list in the terminal for easier testing for future development and maintenance.

Then, it goes through each line of the subject list file, and adds each subject as a new item in sequence. It sorts the list widget alphanumerically in ascending order, then calls the method *sort_subject_list()* to sort the subject list text file in alphanumeric ascending order.

```
26          # Populates the list widget on window startup.
27          with open('subject_list.txt', 'r') as data_file:
28              subject_list = data_file.readlines()
29              print(subject_list)
30              print(len(subject_list))
31              for line in subject_list:
32                  self.list_widget_my_subjects.addItem(line.replace("\n", ""))
33              self.list_widget_my_subjects.sortItems()
34              self.sort_subject_list()
```

I implemented some parts from my prototype program at the start of this stage for the method for sorting the subject list text file. The method for sorting the subject list starts by opening *subject_list.txt* in read only mode. It reads all the lines in the text file, then sorts them alphabetically to the variable *lines*.

Then, it opens a temporary subject list, *subject_list_temp.txt* in write mode. It writes the sorted subject list to the file by writing each subject to the text file in sequence line by line. As explained in the prototype program, the program will write the sorted list to this temporary text file first to ensure that data is atomic. This will prevent cases where data would have

otherwise been lost during an unexpected program crash in the middle of data being written. In the case of a program crash, the subject list will not be lost, as it will always exist in either *subject_list.txt* or *subject_list_temp.txt*.

Finally, the original subject list file, *subject_list.txt*, is opened in write mode. It overwrites the text file by writing the sorted subject list to the text file in sequence line by line. This is the

```
43      # Sorts the subject list text file alphanumerically.
44      def sort_subject_list(self):
45          with open("subject_list.txt", "r") as outfile:
46              lines = outfile.readlines()
47              lines.sort()
48              with open("subject_list_temp.txt", "w") as outfile:
49                  for line in lines:
50                      outfile.write(line)
51                  with open("subject_list.txt", "w") as outfile:
52                      for line in lines:
53                          outfile.write(line)
```

To input the name of the new subject, the user uses a line edit widget, which enables them to manually type in a string. My program will need to access the text entered in this line edit widget when they have saved their subject using the 'Save' button to exit the dialog.

First, I had to connect the 'Save' button to a function for saving the subject. To do this, I used the *.connect()* function for this button to connect the click of the button to the function which saves the subject. I called this method *save_subject*.

In this function, I used the *.text()* function to gather the text from the line edit widget used by the user to input the name of their new subject. I assigned this to a variable, then printed it to the console. Printing the new subject name is not necessary, but it will make testing and maintenance of the program easier later.

After the new subject name is printed, the program adds the subject to the subject list, then sorts the subject list widget alphanumerically in ascending order.

I opened the text file which stores the subject list in append mode, and appended the new subject name alongside a new line (for differentiation between various subjects).

At the end of this function, the method for sorting the subject list text file in alphanumerical ascending order, *sort_subject_list()*, is called.

```
23          # Connects 'Add Subject' button to the Add Subject dialog.
24          self.btn_add_subject.clicked.connect(self.open_dialog_add_subject)
```

```
55      # Saves input to a .txt file for the list of subjects.
56      def save_subject(self):
57          new_subject_name = self.Dialog.line_edit_subject_name.text()
58          print(new_subject_name)
59          self.list_widget_my_subjects.addItem(new_subject_name)
60          self.list_widget_my_subjects.sortItems()
61          with open('subject_list.txt', 'a') as outfile:
62              outfile.write(new_subject_name + "\n")
63          self.sort_subject_list()
```

Testing: Iteration #2 – Adding Subjects

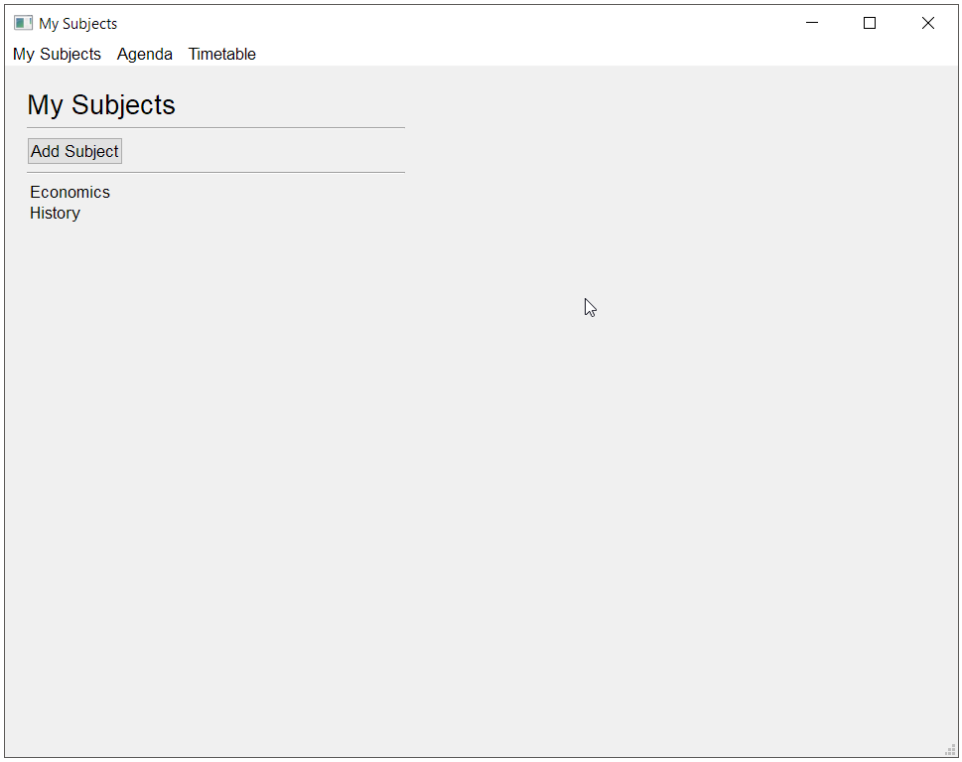
After I finished coding some basic functionality for adding subjects to My Subjects, I decided that it would be a good point to stop and perform testing to ensure that the features I added were implemented correctly.

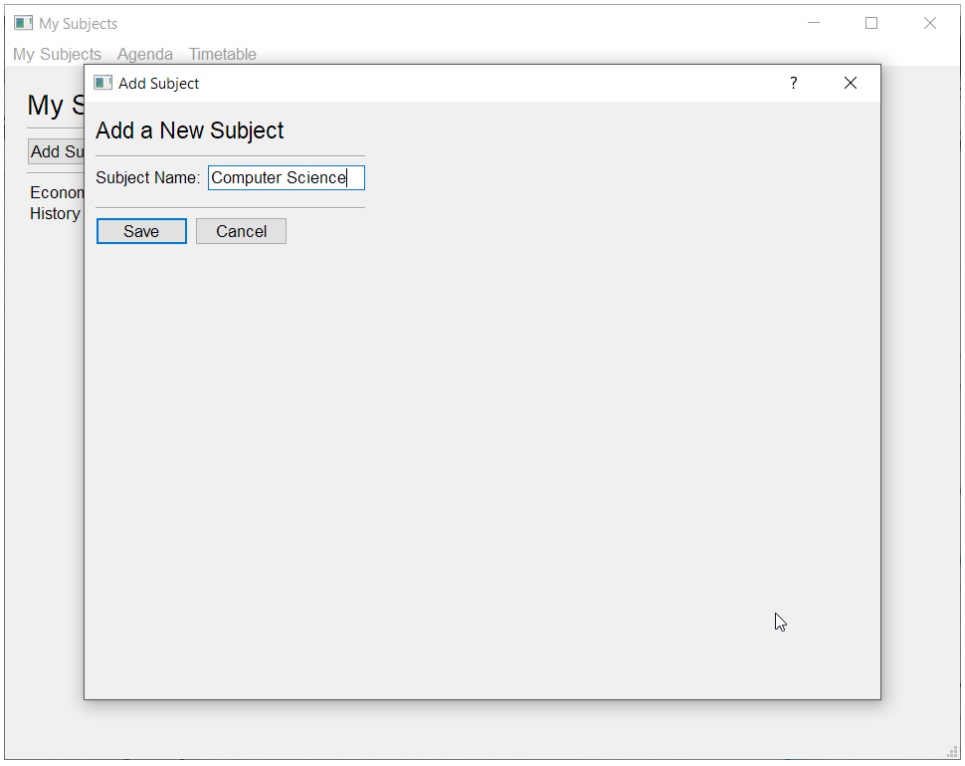
I started by testing the inputs for my program.

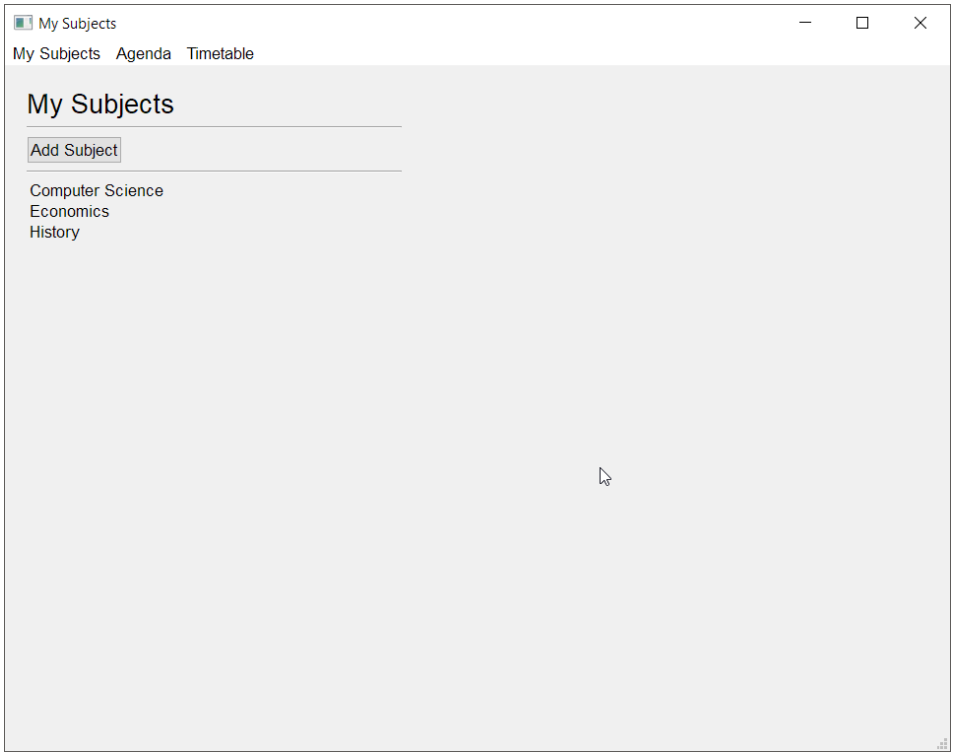
Input	**Nature of Data**	**Data Validation**	**Is Data Accepted?**	**Output**
Typed 'Computer Science' and saved the subject by pressing the 'Save' button.	Normal	Button pressed	Yes	'Computer Science' is added to the subject list in its alphabetical ascending position, and the dialog window is closed.
Typed 'World Philosophy and Ethics' and saved the subject by pressing the 'Save' button.	Extreme	Button pressed	Yes	'World Philosophy and Ethics' is added to the subject list in its alphabetical ascending position, and the dialog window is closed.
Typed 'Computer Science and Further Mathematics' and saved the subject by pressing the 'Save' button.	Erroneous	Button pressed	Yes	'Computer Science and Further Mathematics' is added to the subject list in its alphabetical ascending position, and the dialog window is closed.
Typed nothing and saved the subject by pressing the 'Save' button.	Null	Button pressed	Yes	'' is added to the subject list in its alphabetical ascending position, and the dialog window is closed.

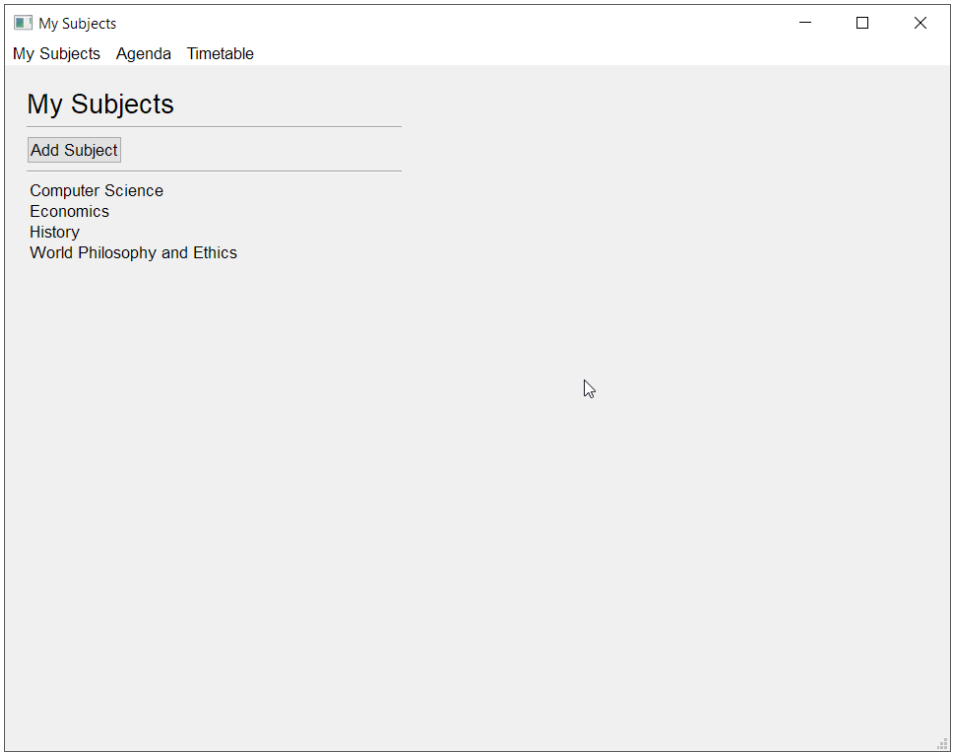
As noted in the table above, there is currently no validation for the data inputs, so the user could have entered anything, including erroneous and null inputs, and it would have been added to the list of subjects.

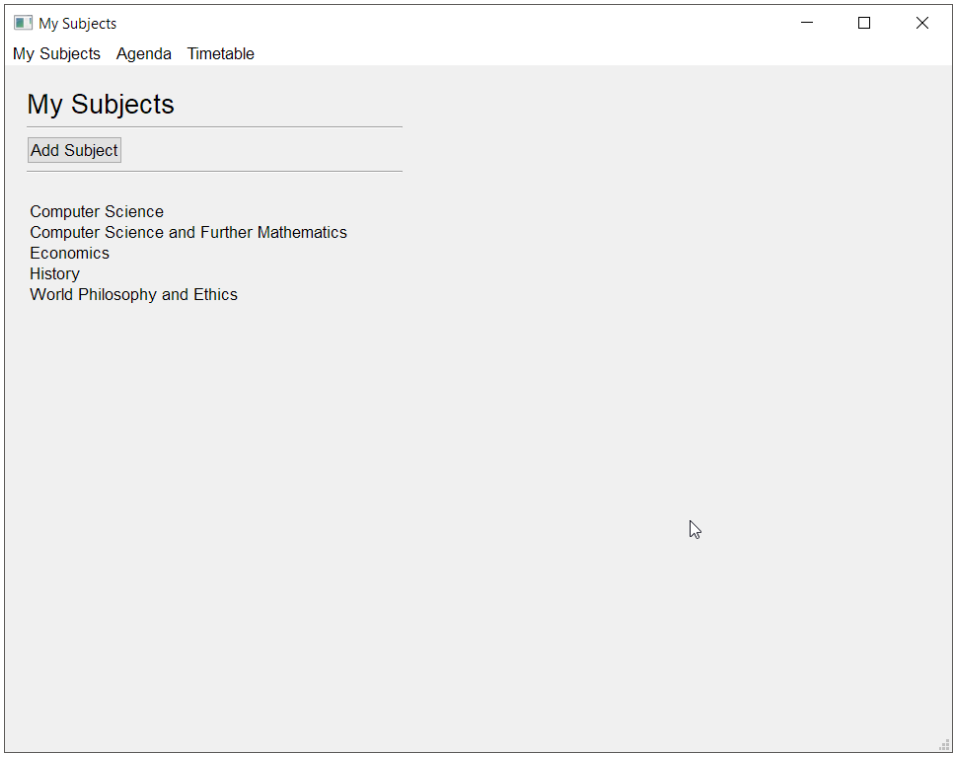
Length checks and presence checks will be implemented in the next development iteration, as the user should be limited from accidental null inputs or typing in subject names which are too long, as this would cause horizontal scrolling of the *QListWidget*. This would prevent erroneous and null inputs such as those shown above to be rejected by the program.


My Subjects
My Subjects Agenda Timetable
My Subjects
Add Subject
Economics
History


My Subjects
My Subjects Agenda Timetable
Add Subject
Add a New Subject
Subject Name: Computer Science
Save
Cancel


My Subjects
My Subjects Agenda Timetable
My Subjects
Add Subject
Computer Science
Economics
History


My Subjects
My Subjects Agenda Timetable
My Subjects
Add Subject
Computer Science
Economics
History
World Philosophy and Ethics



I also performed more general tests to see whether my development in this iteration was successful as a whole. This included two tests from the previous iteration which had been unsuccessful in the previous testing iteration.

Test	Expected Result	Outcome	Further Actions
Are subjects added using Add Subject saved to the list in My Subjects?	Subjects entered from the Add Subject dialog and saved should appear in the list of subjects in My Subjects in alphabetical ascending order.	Subjects entered from the Add Subject dialog are added to the list of subjects in My Subjects in alphabetical ascending order.	N/A.
Is the subject list in My Subjects retained permanently?	The subject list in My Subjects should be stored to a file so that it can be stored permanently and displayed when the program is next run.	The subject list in My Subjects is stored to a .txt file. This file is accessed when the program is started so that it can display the list of subjects.	N/A.
Is there validation to check whether that the user has entered a subject name, and that it contains 30 characters or less?	The program should reject the user's input if either no subject name has been input, or the subject name exceeds 30 characters. It should display an error message to notify them what the issue with their input is.	The program does not validate for the user's input; it accepts all inputs regardless of the nature of the data.	Perform a presence check and a length check on the user's input, and display an error message depending on what is wrong with the input.

In this development iteration, I have addressed the issues exposed by the general testing in the previous iteration, developing basic functionality which enables the user to add subjects which are persisted permanently using .txt files.

The next development iteration will be focused on improving functionality for the user when adding subjects by ensuring that their inputs are validated. This is important because it ensures that the program will display subjects in a clean manner, as well helping prevent new subjects from accidentally being added.

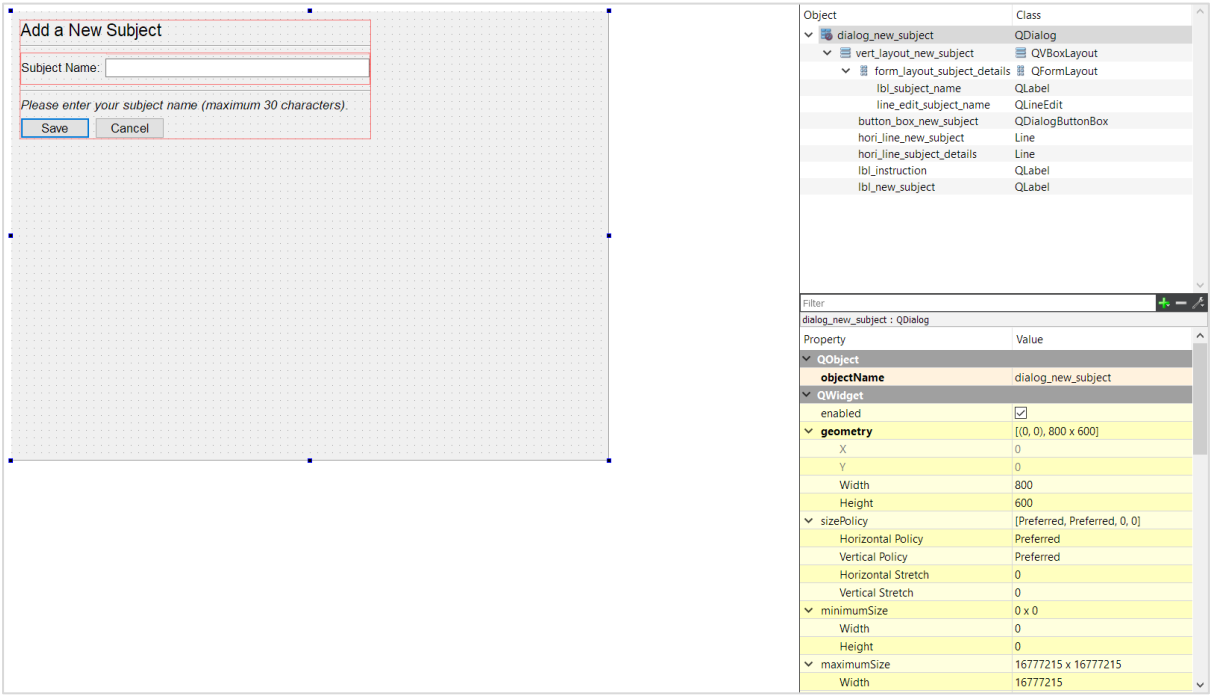
Development: Iteration #3 – Validation and Deletion

The focus of this development iteration was to address the concerns raised by the previous test iteration about validation, as well as adding the ability for the user to delete subjects from their subject list.

To start with, I updated my user interface in Qt Creator by adding a new label to instruct the user about what they need to do. This is necessary because without any instructions, the user will not know that their subject name has a limit of 30 characters or why their input has been rejected, meaning the program will not be intuitive to them, which is crucial for students who need a program which is easy and convenient to use.

I placed the label above the button box, as this is naturally where the user will be reading before they press the 'Save' button. To indicate that it is an instruction and is separate from the rest of the program, I made the font italic.

By default, this label will be telling the user to enter their subject name and that there is a limit of 30 characters. The label will change to tell them that their subject name is too long if they have attempted to save a subject name which exceeds 30 characters, and to try again.



I saved my user interface and then converted it to a Python file using *pyuic5* to automatically generate the code.

```python
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'add_subject.ui'
#
# Created by: PyQt5 UI code generator 5.12.1
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_dialog_new_subject(object):
    def setupUi(self, dialog_new_subject):
        dialog_new_subject.setObjectName("dialog_new_subject")
        dialog_new_subject.resize(800, 600)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        font.setItalic(False)
        dialog_new_subject.setFont(font)
        self.layoutWidget = QtWidgets.QWidget(dialog_new_subject)
        self.layoutWidget.setGeometry(QtCore.QRect(11, 11, 471, 160))
        self.layoutWidget.setObjectName("layoutWidget")
        self.vert_layout_new_subject = QtWidgets.QVBoxLayout(self.layoutWidget)
        self.vert_layout_new_subject.setContentsMargins(0, 0, 0, 0)
        self.vert_layout_new_subject.setObjectName("vert_layout_new_subject")
        self.lbl_new_subject = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setPointSize(14)
        self.lbl_new_subject.setFont(font)
        self.lbl_new_subject.setObjectName("lbl_new_subject")
        self.vert_layout_new_subject.addWidget(self.lbl_new_subject)
        self.hori_line_new_subject = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_new_subject.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_new_subject.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_new_subject.setObjectName("hori_line_new_subject")
        self.vert_layout_new_subject.addWidget(self.hori_line_new_subject)
        self.form_layout_subject_details = QtWidgets.QFormLayout()
        self.form_layout_subject_details.setObjectName(
            "form_layout_subject_details")
        self.lbl_subject_name = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_subject_name.setObjectName("lbl_subject_name")
        self.form_layout_subject_details.setWidget(
            1, QtWidgets.QFormLayout.LabelRole, self.lbl_subject_name)
        self.line_edit_subject_name = QtWidgets.QLineEdit(self.layoutWidget)
        self.line_edit_subject_name.setObjectName("line_edit_subject_name")
        self.form_layout_subject_details.setWidget(
            1, QtWidgets.QFormLayout.FieldRole, self.line_edit_subject_name)
        self.vert_layout_new_subject.addLayout(
            self.form_layout_subject_details)
        self.hori_line_subject_details = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_subject_details.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_subject_details.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_subject_details.setObjectName(
            "hori_line_subject_details")
        self.vert_layout_new_subject.addWidget(self.hori_line_subject_details)
        self.lbl_instruction = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setItalic(True)
        self.lbl_instruction.setFont(font)
        self.lbl_instruction.setObjectName("lbl_instruction")
        self.vert_layout_new_subject.addWidget(self.lbl_instruction)
        self.button_box_new_subject = QtWidgets.QDialogButtonBox(
            self.layoutWidget)
        font = QtGui.QFont()
        font.setItalic(False)
        self.button_box_new_subject.setFont(font)
        self.button_box_new_subject.setOrientation(QtCore.Qt.Horizontal)
        self.button_box_new_subject.setStandardButtons(
            QtWidgets.QDialogButtonBox.Cancel | QtWidgets.QDialogButtonBox.Save)
        self.button_box_new_subject.setObjectName("button_box_new_subject")
        self.vert_layout_new_subject.addWidget(
            self.button_box_new_subject, 0, QtCore.Qt.AlignLeft)

        self.retranslateUi(dialog_new_subject)
        self.button_box_new_subject.rejected.connect(dialog_new_subject.reject)
        self.button_box_new_subject.accepted.connect(dialog_new_subject.accept)
        QtCore.QMetaObject.connectSlotsByName(dialog_new_subject)

    def retranslateUi(self, dialog_new_subject):
        _translate = QtCore.QCoreApplication.translate
        dialog_new_subject.setWindowTitle(
            _translate("dialog_new_subject", "Add Subject"))
        self.lbl_new_subject.setText(_translate(
            "dialog_new_subject", "Add a New Subject"))
        self.lbl_subject_name.setText(_translate(
            "dialog_new_subject", "Subject Name:"))
        self.lbl_instruction.setText(_translate(
            "dialog_new_subject", "Please enter your subject name (maximum 30 characters)."))

```

In the operational code, I needed to perform some validation on the user's subject name input from the line edit widget before the 'Save' button exits the dialog window.

To ensure that the dialog window would perform validation before deciding whether to close the dialog window, I disconnected the 'Save' button from the *.accept()* method which causes the dialog window to be closed when it is pressed.

```
    # Opens the dialog for the user to add a subject.
    def open_dialog_add_subject(self):
        self.Dialog = AddSubjectDialog()
        # Connects 'Save' button to save the user input.
        self.Dialog.button_box_new_subject.accepted.disconnect()
        self.Dialog.button_box_new_subject.accepted.connect(self.save_subject)
        self.Dialog.open()
```

I created a prototype program in stage 2 which included validating the length of the user's new subject input before accepting it as an input. Using this code, I adapted it for this codebase. I used the *len*() function to validate the length of the user's new subject input.

My program checks if the input is greater than or equal to 30 characters, and greater than 0 characters when spaces are removed (using *.replace()*, as this also validates for inputs which only includes whitespace). This performs a length check and a presence check.

Then, the program goes through the usual process. It adds their subject to the *QListWidget*, sorts the list in alphabetical ascending order, saves it to the .txt file, and sorts the .txt file into alphabetical ascending order.

At the end, I closed the dialog window using *self.Dialog.close()*, which was necessary because I removed the code which previously caused the 'Save' button to automatically close the window.

```
        if len(new_subject_name) <= 30 and len((new_subject_name).replace(" ", "")) > 0:
            print(new_subject_name)
            self.list_widget_my_subjects.addItem(new_subject_name)
            self.list_widget_my_subjects.sortItems()
            with open('subject_list.txt', 'a') as outfile:
                outfile.write(new_subject_name + "\n")
            self.sort_subject_list()
            self.Dialog.close()
```

Another scenario is if the user enters no subject name (or only a subject name with whitespace), which I used the *elif* statement for. This means that the user's input failed the presence check. When this validation is failed, the instruction text is changed to inform the user that they have not entered a subject name (and to try again). I coded this using the *setText()* function to change the text label.

```
        elif len((new_subject_name).replace(" ", "")) == 0:
            self.Dialog.lbl_instruction.setText(
                "You have not entered a subject name. Please try again.")
```

The final scenario is where the user enters a subject name which exceeds 30 characters, which I used the *else* statement for. In this case, the input from the user will not be accepted, as their input fails the length check. When this validation is failed, the instruction text is changed to inform the user that their input exceeded the 30 character limit (and to try again).

```
        else:
            self.Dialog.lbl_instruction.setText(
                "Your subject name exceeds 30 characters. Please try again.")
```

These changes were all part of the method *save_subject()*, the changes of which can be seen together below.

```
    # Saves input to a .txt file for the list of subjects.
    def save_subject(self):
        new_subject_name = self.Dialog.line_edit_subject_name.text()
        if len(new_subject_name) <= 30 and len((new_subject_name).replace(" ", "")) > 0:
            print(new_subject_name)
            self.list_widget_my_subjects.addItem(new_subject_name)
            self.list_widget_my_subjects.sortItems()
            with open('subject_list.txt', 'a') as outfile:
                outfile.write(new_subject_name + "\n")
            self.sort_subject_list()
            self.Dialog.close()
        elif len((new_subject_name).replace(" ", "")) == 0:
            self.Dialog.lbl_instruction.setText(
                "You have not entered a subject name. Please try again.")
        else:
            self.Dialog.lbl_instruction.setText(
                "Your subject name exceeds 30 characters. Please try again.")
```

I also fixed the comments about setting up the dialog/main window, as they did not make sense previously. This will make future maintenance and updates to my code easier. You can see the updated comments below (the classes were unchanged).

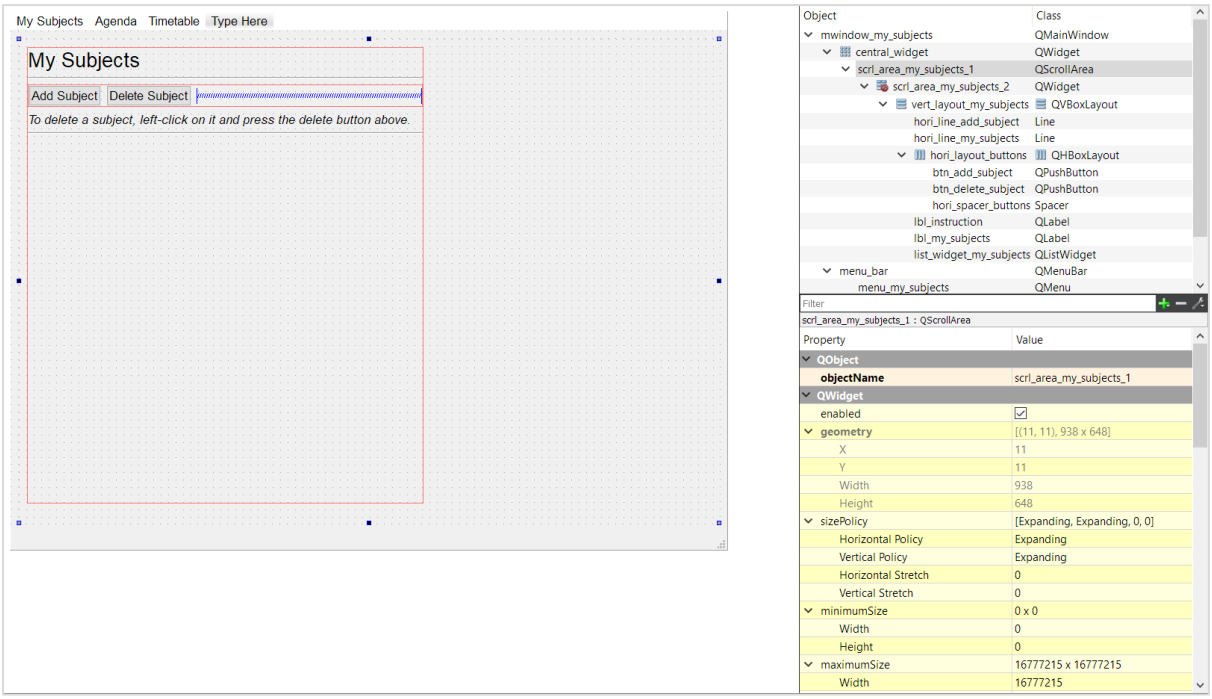
```
# Sets up the Add Subject dialog.


class AddSubjectDialog(QDialog, Ui_dialog_new_subject):
    def __init__(self):
        super().__init__()
        self.setupUi(self)
```

```
# Sets up the My Subjects main window.


class MySubjectsWindow(QMainWindow, Ui_mwindow_my_subjects):
    def __init__(self):
        super().__init__()
        self.setupUi(self)
```

There were many ways I could have chosen to enable the user to delete a subject from My Subjects, but I decided that the most convenient option would be for them to left-click on a subject from the *QListWidget* on My Subjects to select it. From there, the user will be able to left-click on a button above the list of subjects to delete that subject.

I edited the user interface for My Subjects again using Qt Creator. I put the ‘Add Subject’ button in a horizontal layout, then added a button to ‘Delete Subject’ on the right side of it, and a horizontal spacer on the far right side of it so that the buttons would be positioned closely to each other.

Previously, I did not have a label to instruct the user on what to do, because it was already intuitive for the user to navigate. However, it will not be immediately obvious for the user on how to delete a subject, so I also added a text label below the horizontal layout of buttons with instructions in italic about how to delete a subject.



I converted this user interface into a Python file using *pyuic5*.

```python
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'my_subjects.ui'
#
# Created by: PyQt5 UI code generator 5.12.1
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_mwindow_my_subjects(object):
    def setupUi(self, mwindow_my_subjects):
        mwindow_my_subjects.setObjectName("mwindow_my_subjects")
        mwindow_my_subjects.resize(960, 720)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        mwindow_my_subjects.setFont(font)
        self.central_widget = QtWidgets.QWidget(mwindow_my_subjects)
        self.central_widget.setObjectName("central_widget")
        self.gridLayout = QtWidgets.QGridLayout(self.central_widget)
        self.gridLayout.setObjectName("gridLayout")
        self.scrl_area_my_subjects_1 = QtWidgets.QScrollArea(
            self.central_widget)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.scrl_area_my_subjects_1.setFont(font)
        self.scrl_area_my_subjects_1.setFrameShape(QtWidgets.QFrame.NoFrame)
        self.scrl_area_my_subjects_1.setLineWidth(0)
        self.scrl_area_my_subjects_1.setWidgetResizable(True)
        self.scrl_area_my_subjects_1.setObjectName("scrl_area_my_subjects_1")
        self.scrl_area_my_subjects_2 = QtWidgets.QWidget()
        self.scrl_area_my_subjects_2.setGeometry(QtCore.QRect(0, 0, 938, 648))
        self.scrl_area_my_subjects_2.setObjectName("scrl_area_my_subjects_2")
        self.layoutWidget = QtWidgets.QWidget(self.scrl_area_my_subjects_2)
        self.layoutWidget.setGeometry(QtCore.QRect(11, 11, 531, 611))
```

```
        self.layoutWidget.setObjectName("layoutWidget")
        self.vert_layout_my_subjects = QtWidgets.QVBoxLayout(self.layoutWidget)
        self.vert_layout_my_subjects.setContentsMargins(0, 0, 0, 0)
        self.vert_layout_my_subjects.setObjectName("vert_layout_my_subjects")
        self.lbl_my_subjects = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setPointSize(16)
        self.lbl_my_subjects.setFont(font)
        self.lbl_my_subjects.setObjectName("lbl_my_subjects")
        self.vert_layout_my_subjects.addWidget(self.lbl_my_subjects)
        self.hori_line_my_subjects = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_my_subjects.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_my_subjects.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_my_subjects.setObjectName("hori_line_my_subjects")
        self.vert_layout_my_subjects.addWidget(self.hori_line_my_subjects)
        self.hori_layout_buttons = QtWidgets.QHBoxLayout()
        self.hori_layout_buttons.setObjectName("hori_layout_buttons")
        self.btn_add_subject = QtWidgets.QPushButton(self.layoutWidget)
        self.btn_add_subject.setObjectName("btn_add_subject")
        self.hori_layout_buttons.addWidget(
            self.btn_add_subject, 0, QtCore.Qt.AlignLeft)
        self.btn_delete_subject = QtWidgets.QPushButton(self.layoutWidget)
        self.btn_delete_subject.setAutoDefault(False)
        self.btn_delete_subject.setDefault(False)
        self.btn_delete_subject.setFlat(False)
        self.btn_delete_subject.setObjectName("btn_delete_subject")
        self.hori_layout_buttons.addWidget(self.btn_delete_subject)
        spacerItem = QtWidgets.QSpacerItem(
            40, 20, QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Minimum)
        self.hori_layout_buttons.addItem(spacerItem)
        self.vert_layout_my_subjects.addLayout(self.hori_layout_buttons)
        self.lbl_instruction = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setItalic(True)
        self.lbl_instruction.setFont(font)
        self.lbl_instruction.setObjectName("lbl_instruction")
        self.vert_layout_my_subjects.addWidget(self.lbl_instruction)
        self.hori_line_add_subject = QtWidgets.QFrame(self.layoutWidget)
```

```
        self.hori_line_add_subject.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_add_subject.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_add_subject.setObjectName("hori_line_add_subject")
        self.vert_layout_my_subjects.addWidget(self.hori_line_add_subject)
        self.list_widget_my_subjects = QtWidgets.QListWidget(self.layoutWidget)
        palette = QtGui.QPalette()
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active,
                         QtGui.QPalette.WindowText, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 255, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Button, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 255, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Light, brush)
        brush = QtGui.QBrush(QtGui.QColor(247, 247, 247, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Midlight, brush)
        brush = QtGui.QBrush(QtGui.QColor(120, 120, 120, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Dark, brush)
        brush = QtGui.QBrush(QtGui.QColor(160, 160, 160, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Mid, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Text, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 255))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active,
                         QtGui.QPalette.BrightText, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active,
                         QtGui.QPalette.ButtonText, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.NoBrush)
```

```
        palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Base, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 255, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Window, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Shadow, brush)
        brush = QtGui.QBrush(QtGui.QColor(247, 247, 247, 127))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active,
                         QtGui.QPalette.AlternateBase, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 220))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active,
                         QtGui.QPalette.ToolTipBase, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active,
                         QtGui.QPalette.ToolTipText, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0, 128))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active,
                         QtGui.QPalette.PlaceholderText, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive,
                         QtGui.QPalette.WindowText, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 255, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Button, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 255, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Light, brush)
        brush = QtGui.QBrush(QtGui.QColor(247, 247, 247, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive,
                         QtGui.QPalette.Midlight, brush)
        brush = QtGui.QBrush(QtGui.QColor(120, 120, 120, 0))
```

```
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Dark, brush)
        brush = QtGui.QBrush(QtGui.QColor(160, 160, 160, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Mid, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Text, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 255))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive,
                         QtGui.QPalette.BrightText, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive,
                         QtGui.QPalette.ButtonText, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.NoBrush)
        palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Base, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 255, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Window, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Shadow, brush)
        brush = QtGui.QBrush(QtGui.QColor(247, 247, 247, 127))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive,
                         QtGui.QPalette.AlternateBase, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 220))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive,
                         QtGui.QPalette.ToolTipBase, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive,
                         QtGui.QPalette.ToolTipText, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0, 128))
```

```
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive,
                         QtGui.QPalette.PlaceholderText, brush)
        brush = QtGui.QBrush(QtGui.QColor(120, 120, 120, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled,
                         QtGui.QPalette.WindowText, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 255, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Button, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 255, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Light, brush)
        brush = QtGui.QBrush(QtGui.QColor(247, 247, 247, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled,
                         QtGui.QPalette.Midlight, brush)
        brush = QtGui.QBrush(QtGui.QColor(120, 120, 120, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Dark, brush)
        brush = QtGui.QBrush(QtGui.QColor(160, 160, 160, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Mid, brush)
        brush = QtGui.QBrush(QtGui.QColor(120, 120, 120, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Text, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 255))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled,
                         QtGui.QPalette.BrightText, brush)
        brush = QtGui.QBrush(QtGui.QColor(120, 120, 120, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled,
                         QtGui.QPalette.ButtonText, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.NoBrush)
        palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Base, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 255, 0))
```

```
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Window, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Shadow, brush)
        brush = QtGui.QBrush(QtGui.QColor(240, 240, 240, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled,
                         QtGui.QPalette.AlternateBase, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 220))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled,
                         QtGui.QPalette.ToolTipBase, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled,
                         QtGui.QPalette.ToolTipText, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0, 128))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled,
                         QtGui.QPalette.PlaceholderText, brush)
        self.list_widget_my_subjects.setPalette(palette)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.list_widget_my_subjects.setFont(font)
        self.list_widget_my_subjects.setAutoFillBackground(False)
        self.list_widget_my_subjects.setStyleSheet(
            "background-color: rgba(255, 255, 255, 0)")
        self.list_widget_my_subjects.setFrameShape(QtWidgets.QFrame.NoFrame)
        self.list_widget_my_subjects.setFrameShadow(QtWidgets.QFrame.Plain)
        self.list_widget_my_subjects.setObjectName("list_widget_my_subjects")
        self.vert_layout_my_subjects.addWidget(self.list_widget_my_subjects)
        self.scrl_area_my_subjects_1.setWidget(self.scrl_area_my_subjects_2)
        self.gridLayout.addWidget(self.scrl_area_my_subjects_1, 0, 0, 1, 1)
        mwindow_my_subjects.setCentralWidget(self.central_widget)
        self.menu_bar = QtWidgets.QMenuBar(mwindow_my_subjects)
        self.menu_bar.setGeometry(QtCore.QRect(0, 0, 960, 25))
```

```
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_bar.setFont(font)
        self.menu_bar.setObjectName("menu_bar")
        self.menu_my_subjects = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_my_subjects.setFont(font)
        self.menu_my_subjects.setObjectName("menu_my_subjects")
        self.menu_agenda = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_agenda.setFont(font)
        self.menu_agenda.setObjectName("menu_agenda")
        self.menu_timetable = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_timetable.setFont(font)
        self.menu_timetable.setObjectName("menu_timetable")
        mwindow_my_subjects.setMenuBar(self.menu_bar)
        self.status_bar = QtWidgets.QStatusBar(mwindow_my_subjects)
        self.status_bar.setObjectName("status_bar")
        mwindow_my_subjects.setStatusBar(self.status_bar)
        self.actionAgenda = QtWidgets.QAction(mwindow_my_subjects)
        self.actionAgenda.setObjectName("actionAgenda")
        self.menu_bar.addAction(self.menu_my_subjects.menuAction())
        self.menu_bar.addAction(self.menu_agenda.menuAction())
        self.menu_bar.addAction(self.menu_timetable.menuAction())

        self.retranslateUi(mwindow_my_subjects)
        QtCore.QMetaObject.connectSlotsByName(mwindow_my_subjects)
```

```
    def retranslateUi(self, mwindow_my_subjects):
        _translate = QtCore.QCoreApplication.translate
        mwindow_my_subjects.setWindowTitle(
            _translate("mwindow_my_subjects", "My Subjects"))
        self.lbl_my_subjects.setText(_translate(
            "mwindow_my_subjects", "My Subjects"))
        self.btn_add_subject.setText(_translate(
            "mwindow_my_subjects", "Add Subject"))
        self.btn_delete_subject.setText(_translate(
            "mwindow_my_subjects", "Delete Subject"))
        self.lbl_instruction.setText(_translate(
            "mwindow_my_subjects", "To delete a subject, left-click on it and press the delete button above."))
        self.menu_my_subjects.setTitle(_translate(
            "mwindow_my_subjects", "My Subjects"))
        self.menu_agenda.setTitle(_translate("mwindow_my_subjects", "Agenda"))
        self.menu_timetable.setTitle(_translate(
            "mwindow_my_subjects", "Timetable"))
        self.actionAgenda.setText(_translate("mwindow_my_subjects", "Agenda"))

```

Next, I worked on the operational code for deleting the subject from the list when an item is selected and the user presses on the ‘Delete Subject’ button.

To start with, I connected the newly added ‘Delete Subject’ button to a method, *delete_subject()*.

```
        # Connects 'Delete Subject' button to the method for deleting a subject.
        self.btn_delete_subject.clicked.connect(self.delete_subject)
```

Then, I created a new method, *delete_subject()*, to handle the deletion of the subject. In this method, the selected item is assigned to a variable, *selected_item*. I ran a *for* loop to perform *takeItem()* on the row of the selected item, which effectively deletes that subject from the *QListWidget*. After that, it runs the method *save_subject_list()*.

```
    # Deletes the selected subject.
    def delete_subject(self):
        selected_item = self.list_widget_my_subjects.selectedItems()
        for item in selected_item:
            self.list_widget_my_subjects.takeItem(
                self.list_widget_my_subjects.row(item))
        self.save_subject_list()
```

I created the method *save_subject_list()* because it enables the code to be reused if this is required for future features. This method counts the number of items in the list of subjects to decide how long to iterate the *for* loop. In each loop, the method reads the text of a line, then writes it to a text file along with a new line, and iterates to the next line.

As in the previous development iteration, I ensured that data atomicity is present here; the method writes to *subject_list_temp.txt* first to ensure that the data on the file is not lost if the program crashes when it is writing it to the text file. Once it has written to *subject_list_temp.txt*, the program writes the same data to *subject_list.txt*. This way, in the case of a crash, no data will be lost.

```
55      # Saves the subject list.
56      def save_subject_list(self):
57          with open("subject_list_temp.txt", "w") as outfile:
58              for i in range(self.list_widget_my_subjects.count()):
59                  subject = self.list_widget_my_subjects.item(i).text()
60                  outfile.write(subject + "\n")
61              with open("subject_list.txt", "w") as outfile:
62                  for i in range(self.list_widget_my_subjects.count()):
63                      subject = self.list_widget_my_subjects.item(i).text()
64                      outfile.write(subject + "\n")
```

At this point, I also noticed an inefficiency in my existing code. I was using *.replace()* to remove whitespaces when performing validation and when removing new lines to populate the *QListWidget* on start-up. Instead, I used *.strip()* to remove these unwanted characters.

```
78      # Saves input to a .txt file for the list of subjects.
79      def save_subject(self):
80          new_subject_name = self.Dialog.line_edit_subject_name.text()
81          if len(new_subject_name) <= 30 and len((new_subject_name).strip(" ")) > 0:
82              print(new_subject_name)
83              self.list_widget_my_subjects.addItem(new_subject_name)
84              self.list_widget_my_subjects.sortItems()
85              with open('subject_list.txt', 'a') as outfile:
86                  outfile.write(new_subject_name + "\n")
87              self.sort_subject_list()
88              self.Dialog.close()
89          elif len((new_subject_name).strip(" ")) == 0:
90              self.Dialog.lbl_instruction.setText(
91                  "You have not entered a subject name. Please try again.")
92          else:
93              self.Dialog.lbl_instruction.setText(
94                  "Your subject name exceeds 30 characters. Please try again.")
```

```
29          # Populates the list widget on window startup.
30          with open("subject_list.txt", "r") as data_file:
31              subject_list = data_file.readlines()
32              print(subject_list)
33              print(len(subject_list))
34              for line in subject_list:
35                  self.list_widget_my_subjects.addItem(line.strip("\n"))
36              self.list_widget_my_subjects.sortItems()
37              self.sort_subject_list()
```

After all these changes, the updated code for *my_subjects.py* can be seen below.

```python
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QDialog, QMainWindow

from add_subject_setup import Ui_dialog_new_subject
from my_subjects_setup import Ui_mwindow_my_subjects

# Sets up the Add Subject dialog.


class AddSubjectDialog(QDialog, Ui_dialog_new_subject):
    def __init__(self):
        super().__init__()
        self.setupUi(self)

# Sets up the My Subjects main window.


class MySubjectsWindow(QMainWindow, Ui_mwindow_my_subjects):
    def __init__(self):
        super().__init__()
        self.setupUi(self)

        # Connects 'Add Subject' button to the Add Subject dialog.
        self.btn_add_subject.clicked.connect(self.open_dialog_add_subject)

        # Connects 'Delete Subject' button to the method for deleting the selected subject.
        self.btn_delete_subject.clicked.connect(self.delete_subject)

        # Populates the list widget on window startup.
        with open("subject_list.txt", "r") as data_file:
            subject_list = data_file.readlines()
            print(subject_list)
            print(len(subject_list))
            for line in subject_list:
                self.list_widget_my_subjects.addItem(line.strip("\n"))
            self.list_widget_my_subjects.sortItems()
            self.sort_subject_list()

    # Opens the dialog for the user to add a subject.
    def open_dialog_add_subject(self):
        self.Dialog = AddSubjectDialog()
        # Connects 'Save' button to save the user input.
        self.Dialog.button_box_new_subject.accepted.disconnect()
        self.Dialog.button_box_new_subject.accepted.connect(self.save_subject)
        self.Dialog.open()

    # Deletes the selected subject.
    def delete_subject(self):
        selected_item = self.list_widget_my_subjects.selectedItems()
        for item in selected_item:
            self.list_widget_my_subjects.takeItem(
                self.list_widget_my_subjects.row(item))
        self.save_subject_list()

    # Saves the subject list.
    def save_subject_list(self):
        with open("subject_list_temp.txt", "w") as outfile:
            for i in range(self.list_widget_my_subjects.count()):
                subject = self.list_widget_my_subjects.item(i).text()
                outfile.write(subject + "\n")
            with open("subject_list.txt", "w") as outfile:
                for i in range(self.list_widget_my_subjects.count()):
                    subject = self.list_widget_my_subjects.item(i).text()
                    outfile.write(subject + "\n")

    # Sorts the subject list text file alphanumerically.
    def sort_subject_list(self):
        with open("subject_list.txt", "r") as outfile:
            lines = outfile.readlines()
            lines.sort()
            with open("subject_list_temp.txt", "w") as outfile:
                for line in lines:
                    outfile.write(line)
                with open("subject_list.txt", "w") as outfile:
                    for line in lines:
                        outfile.write(line)
```

```
    # Saves input to a .txt file for the list of subjects.
    def save_subject(self):
        new_subject_name = self.Dialog.line_edit_subject_name.text()
        if len(new_subject_name) <= 30 and len((new_subject_name).strip(" ")) > 0:
            print(new_subject_name)
            self.list_widget_my_subjects.addItem(new_subject_name)
            self.list_widget_my_subjects.sortItems()
            with open('subject_list.txt', 'a') as outfile:
                outfile.write(new_subject_name + "\n")
            self.sort_subject_list()
            self.Dialog.close()
        elif len((new_subject_name).strip(" ")) == 0:
            self.Dialog.lbl_instruction.setText(
                "You have not entered a subject name. Please try again.")
        else:
            self.Dialog.lbl_instruction.setText(
                "Your subject name exceeds 30 characters. Please try again.")


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    mwindow_my_subjects = MySubjectsWindow()
    mwindow_my_subjects.show()
    sys.exit(app.exec_())
```

Testing: Iteration #3 – Validation and Deletion

Once my program was coded to implement validation and deletion, I had completed my attempt to meet the objectives at the start of this stage, so I started testing the changes to check that they were implemented successfully.

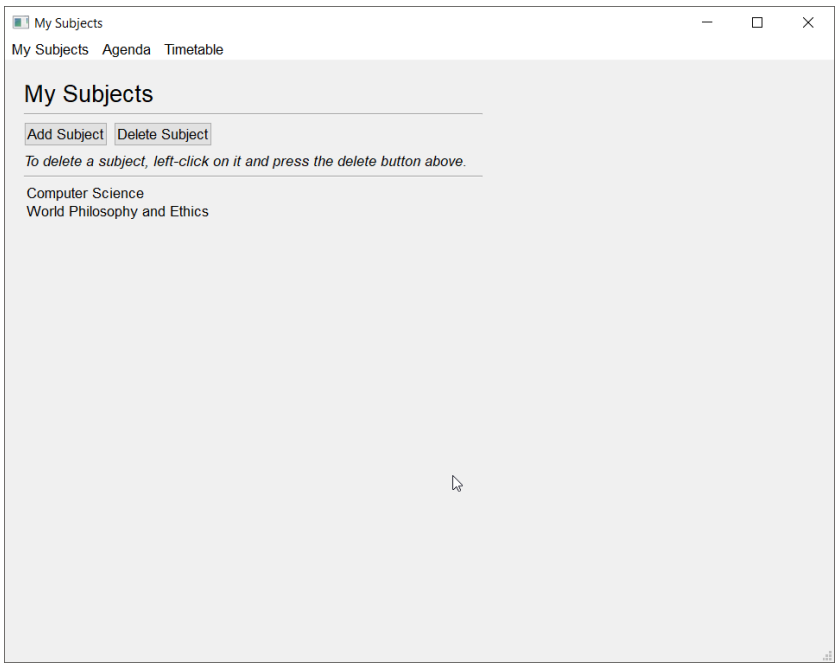
In the previous testing iteration, all types of inputs had the same response from the program, which was adding to the subject to the subject list in its alphabetical ascending position. In this development iteration, the validation features should change this, as erroneous and null subject names should be rejected by the program.

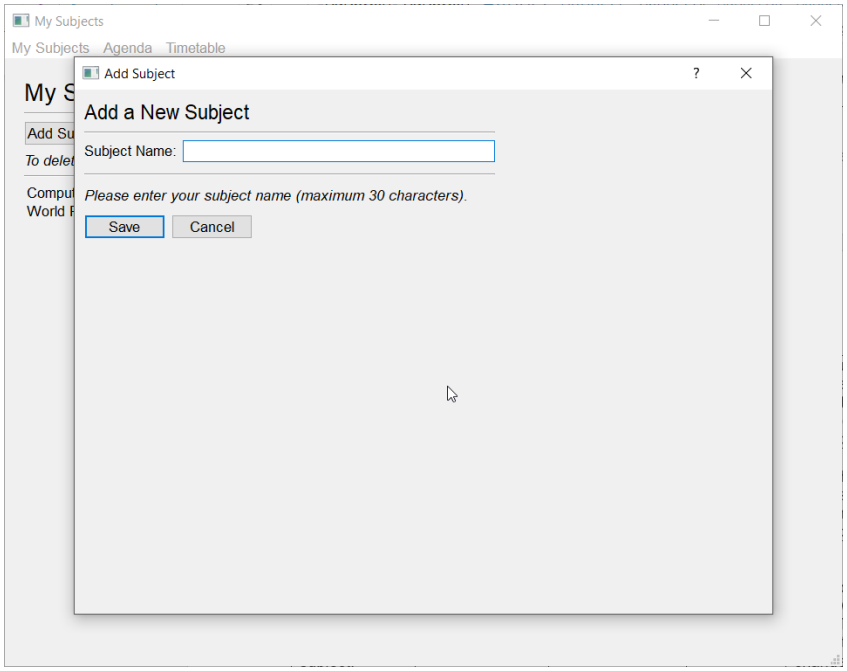
For the beta testing for inputs, I decided to test the same inputs as in the previous testing iteration, and tested the deletion feature I implemented in this development iteration.

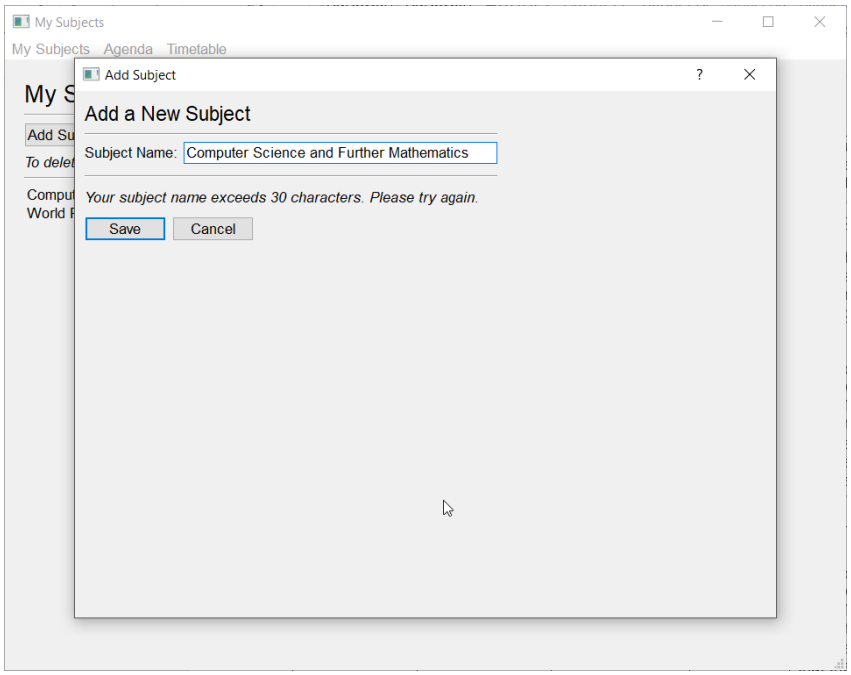
Input	**Nature of Data**	**Data Validation**	**Is Data Accepted?**	**Output**
Typed ‘Computer Science’ and saved the subject by pressing the ‘Save’ button.	Normal	Presence check, length check, button pressed	Yes	‘Computer Science’ is added to the subject list in its alphabetical ascending position, and the dialog window is closed.
Typed ‘World Philosophy and Ethics’ and saved the subject by pressing the ‘Save’ button.	Extreme	Presence check, length check, button pressed	Yes	‘World Philosophy and Ethics’ is added to the subject list in its alphabetical ascending position, and the dialog window is closed.
Typed ‘Computer Science and Further Mathematics’ and saved the subject by	Erroneous	Presence check, length check, button pressed	Yes	The subject is not added to the subject list, and the dialog window remains open. The instruction label changes to specify that their subject

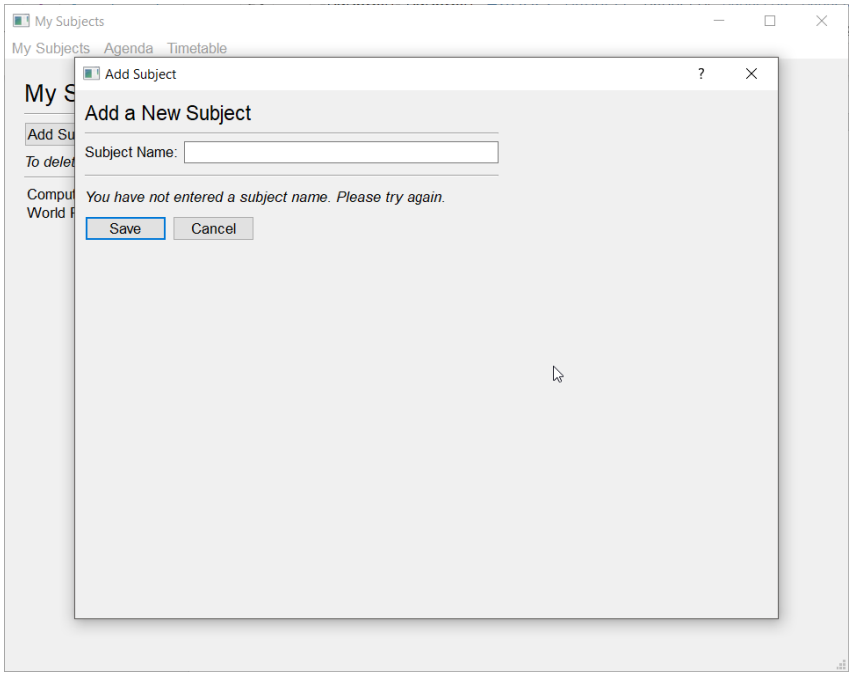
pressing the 'Save' button.				name exceeds 30 characters, and to try again.
Typed nothing and saved the subject by pressing the 'Save' button.	Null	Presence check, length check, button pressed	Yes	The subject is not added to the subject list, and the dialog window remains open. The instruction label changes to specify that the user has not entered a subject name, and to try again.
Typed 'Further Mathematics' and saved the subject by pressing the 'Save' button.	Normal	Presence check, length check, button pressed	Yes	'Further Mathematics' is added to the subject list in its alphabetical ascending position, and the dialog window is closed.
Selected 'World Philosophy and Ethics' and pressed the 'Delete Subject' button.	Normal	Button pressed	Yes	'World Philosophy and Ethics' was removed from the subject list.

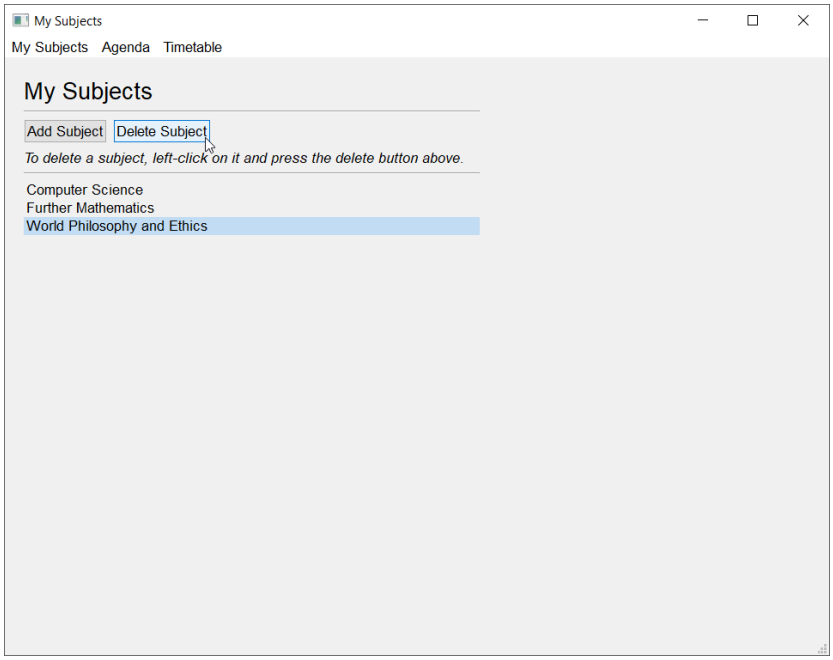
Screenshots demonstrating some of the beta testing I performed can be seen below.

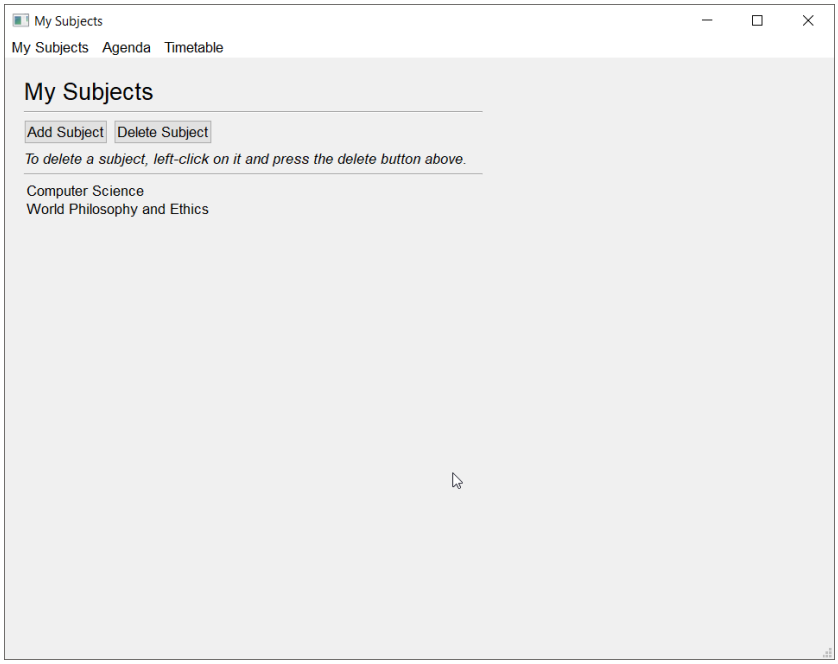



My Subjects
My Subjects Agenda Timetable
Add Subject
Add a New Subject
Subject Name:
Please enter your subject name (maximum 30 characters).
Save
Cancel


My Subjects
My Subjects Agenda Timetable
Add Subject
Add a New Subject
Subject Name: Computer Science and Further Mathematics
Your subject name exceeds 30 characters. Please try again.
Save
Cancel


My Subjects
My Subjects Agenda Timetable
Add Subject
Add a New Subject
Subject Name:
You have not entered a subject name. Please try again.
Save
Cancel


My Subjects
My Subjects Agenda Timetable
My Subjects
Add Subject
Delete Subject
To delete a subject, left-click on it and press the delete button above.
Computer Science
Further Mathematics
World Philosophy and Ethics



Based on the input tests above, the validation and deletion features developed in this iteration were working as intended.

Next, I performed more general tests to compare my program to the objectives set at the start of this stage, including the general test which had been unsuccessful in the previous testing iteration.

Test	Expected Result	Outcome	Further Actions
Is there validation to check whether that the user has entered a subject name, and that it contains 30 characters or less?	The program should reject the user's input if either no subject name has been input, or the subject name exceeds 30 characters. It should display an error message to notify them what the issue with their input is.	The program validates for the user's input. If the subject name contains 30 characters or less, and more than 0 characters (excluding whitespace), the subject will be added to the list, and the dialog window is closed. If the subject name contains 0 characters (excluding whitespace), the dialog window remains open, and the instruction label changes to tell the user that they have not entered a subject name. Otherwise, when the subject name contains more than 30 characters, the dialog window remains open, and the instruction label changes to tell the	N/A.

		user that their subject name exceeds 30 characters.	
Is there a way for the user to delete subjects from the list of subjects?	The program should allow the user to delete subjects from the subject list by selecting a subject from the subject list and pressing a button to delete that subject.	The program allows the user to delete a subject from the subject list by selecting a subject from the subject left (by left-clicking it), and pressing the 'Delete Subject' button.	N/A.

My program passed all the input tests and general tests, with no further action needed, so I decided to end this stage of development.

Review

My objectives for this stage were to create a dialog window for adding subjects, and develop functionality for this dialog window so that users can add subjects which are saved permanently, with the ability of delete them afterwards.

By the end of this stage, I successfully developed a robust system for adding and deleting subjects, with data atomicity to protect against loss of data, and thoroughly developed validation to ensure that the user is using My Subjects as intended.

Next Steps for Module

In the close future, I plan to integrate the list in My Subjects with Agenda for the combo box used when adding tasks. This is a key feature which will be essential to my application functioning properly, so it is likely to be developed in the next stage.

Stage 7: Adding Tasks to Agenda

Objectives

My objectives for this stage are to develop functionality for Agenda which allows the user to add new tasks to the Agenda using the user interface I developed earlier in stage 5.

The program should be able to perform validation through length checks and presence checks, and if the validation is passed, the new task should appear in a chronologically (by due date) sorted list in Agenda. The user should be able to mark tasks as complete/incomplete, delete tasks, and hide/show completed tasks.

Prototype Program

As the primary focus of this stage is developing data persistence into my program, I researched into the ways Python can be used to save user inputs. I came across the idea of using Python JSON, which was explained by W3Schools (https://www.w3schools.com/python/python_json.asp) and Stack Abuse (https://stackabuse.com/reading-and-writing-json-to-a-file-in-python/).

A JSON file is used for permanent storage, meaning data can be retained even after the program is closed. It is a good solution for the storage of tasks in this program, because it allows data to be grouped together. In this case, it would be grouping together the task title, subject, and due date.

I did not have any experience in working with Python JSON at this stage, so I decided that this would be what my prototype program would be about, because it would help me later in this stage once the user interface for adding new subjects was designed and connected to the button for adding a new task in the main window for Agenda.

I created a program which reused some of the code from the prototype I created in stage 1 and the further validation in stage 6 to validate the user inputs. It takes the validated user inputs for task title, subject, and due date, then inputs a Python object using a dictionary for the data.

The program starts by identifying the task list variable, *task_list*, as a dictionary. Then, it starts reading the existing JSON file if it exists. It converts the JSON file to a Python dictionary, *task_list*, and then outputs this to the console for debugging purposes. When outputting this dictionary to the console, I used *ensure_ascii=False* to ensure that special characters such as accented characters will be shown, which may be useful if language is recorded in a foreign language, or if there are foreign words in it due to tasks for subjects like French and Spanish being added. I also used *indent=4* to display the JSON file in pretty print, and *sys.stdout* enables this to be outputted.

```
4   # Identifies the task list as a dictionary variable.
5   task_list = {}
6
7   # Reads the existing JSON file and prints it to console.
8   with open('task_list.json', 'r') as outfile:
9       try:
10          task_list = json.load(outfile)
11          json.dump(task_list, sys.stdout, ensure_ascii=False, indent=4)
12      except ValueError:
13          print("Empty JSON file.")
14          pass
```

Then, the program takes the inputs from the user for the task title, subject, and due date. It asks them for their input until a valid input is given. The characters must be greater than 0 characters (excluding white space) and a maximum of 31 characters for the task title and subject, and greater than 0 characters (excluding white space) and a maximum of 10 characters for the due date (ideally DD-MM-YYYY).

In development later in this stage, the task title will be the only input validated, because the subject would be chosen from the combo box (having already been validated in My Subjects), and the due date will be the selected by the *QCalendar* widget.

```
# Validates the entries of task details.
task_title = str(input("\n\nTask Title: "))
while len(task_title) > 30:
    print("Your task title exceeds 30 characters. Please try again.\n")
    task_title = str(input("Task Title: "))
while len((task_title).strip(" ")) == 0:
    print("You have not entered a task title. Please try again.\n")
    task_title = str(input("Task Title: "))

task_subject = str(input("\nSubject: "))
while len(task_subject) > 30:
    print("Your subject exceeds 30 characters. Please try again.\n")
    task_subject = str(input("Subject: "))
while len((task_subject).strip(" ")) == 0:
    print("You have not entered a subject. Please try again.\n")
    task_subject = str(input("Subject: "))

task_due_date = str(input("\nDue Date: "))
while len(task_due_date) > 10:
    print("Your due date exceeds 10 characters. Please try again.\n")
    task_due_date = str(input("Due Date: "))
while len((task_due_date).strip(" ")) == 0:
    print("You have not entered a due date. Please try again.\n")
    task_due_date = str(input("Due Date: "))
```

After the validated inputs have been obtained, the program adds the task title as a new key in the dictionary, then assigns the subject and due date as its values.

```
# Adds the new task to the dictionary, using the task title as the key.
task_list[task_title] = {"subject": task_subject, "due_date": task_due_date}
```

Once the program has inputted the Python object, my program converts this input from Python to JSON using *json.dump()*, which takes the input from *task_list*, as seen in the brackets. Again, I used *ensure_ascii=False* to cover cases in which special characters are entered, and used *indent=4* to store the JSON in pretty print for easier human reading.

```
# Converts the input into JSON and saves it to a JSON file, then prints to console.
with open('task_list.json', 'w') as outfile:
    json.dump(task_list, outfile, ensure_ascii=False, indent=4)
    print("\n")
    json.dump(task_list, sys.stdout, ensure_ascii=False, indent=4)

```

After this, I printed the JSON string in a new line, as this provides me an easier way to test whether my program has executed successfully.

```
# Converts the input into JSON and saves it to a JSON file.
with open('task_list.json', 'w') as outfile:
    json.dump(task_list, outfile, ensure_ascii=False, indent=4)
    # Prints the added JSON string.
    print("\n")
    json.dump(task_list, sys.stdout, ensure_ascii=False, indent=4)
```

The full code for the prototype program can be seen below.

```python
import json
import sys

# Identifies the task list as a dictionary variable.
task_list = {}

# Reads the existing JSON file and prints it to console.
with open('task_list.json', 'r') as outfile:
    try:
        task_list = json.load(outfile)
        json.dump(task_list, sys.stdout, ensure_ascii=False, indent=4)
    except ValueError:
        print("Empty JSON file.")
        pass

# Validates the entries of task details.
task_title = str(input("\n\nTask Title: "))
while len(task_title) > 30:
    print("Your task title exceeds 30 characters. Please try again.\n")
    task_title = str(input("Task Title: "))
while len((task_title).strip(" ")) == 0:
    print("You have not entered a task title. Please try again.\n")
    task_title = str(input("Task Title: "))

task_subject = str(input("\nSubject: "))
while len(task_subject) > 30:
    print("Your subject exceeds 30 characters. Please try again.\n")
    task_subject = str(input("Subject: "))
while len((task_subject).strip(" ")) == 0:
    print("You have not entered a subject. Please try again.\n")
    task_subject = str(input("Subject: "))

task_due_date = str(input("\nDue Date: "))
while len(task_due_date) > 10:
    print("Your due date exceeds 10 characters. Please try again.\n")
    task_due_date = str(input("Due Date: "))
while len((task_due_date).strip(" ")) == 0:
    print("You have not entered a due date. Please try again.\n")
    task_due_date = str(input("Due Date: "))

# Adds the new task to the dictionary, using the task title as the key.
task_list[task_title] = {"subject": task_subject, "due_date": task_due_date}

# Converts the input into JSON and saves it to a JSON file, then prints to console.
with open('task_list.json', 'w') as outfile:
    json.dump(task_list, outfile, ensure_ascii=False, indent=4)
    print("\n")
    json.dump(task_list, sys.stdout, ensure_ascii=False, indent=4)
```

From the prototype program section in stage 1 and the program in stage 6, I already know that the validation functions properly, so I did not need to test this again, as I reused that code for user inputs.

I tested this prototype program to check that it successfully saves the user inputs to a JSON file. As shown in the screenshots, my prototype program successfully reads from the file, prints it in pretty print JSON if there is existing data, appends the new user input, saves the data to a pretty printed JSON file, and prints the JSON string to the console.

```
Empty JSON file.


Task Title: Programming Project

Subject: Computer Science

Due Date: 15/04/2019


{
    "Programming Project": {
        "subject": "Computer Science",
        "due_date": "15/04/2019"
    }
}
```

```
{
    "Programming Project": {
        "subject": "Computer Science",
        "due_date": "15/04/2019"
    }
}
```

task_list.json 08/04/2019 10:37 ... JSON File 1 KB

```
{
    "Programming Project": {
        "subject": "Computer Science",
        "due_date": "15/04/2019"
    }
}

Task Title: Differentiation

Subject: Mathematics

Due Date: 16/05/2019


{
    "Programming Project": {
        "subject": "Computer Science",
        "due_date": "15/04/2019"
    },
    "Differentiation": {
        "subject": "Mathematics",
        "due_date": "16/05/2019"
    }
}
```

```
{
    "Programming Project": {
        "subject": "Computer Science",
        "due_date": "15/04/2019"
    },
    "Differentiation": {
        "subject": "Mathematics",
        "due_date": "16/05/2019"
    }
}
```

task_list.json 08/04/2019 10:38 ... JSON File 1 KB

Development: Iteration #1 – List of Tasks

I started out by separating the automatically generated code for Agenda and my operational code for Agenda into two separate files, like I did in development iteration #2 of stage 6.

This meant that instead of having just *agenda.py*, I used *agenda.py* for my operational code, and *agenda_setup.py* for my automatically generated user interface code. As aforementioned, this is due to the convention of modular programming, which makes development and reusing code easier.

The code for the automatically generated user interface code can be seen below.

```python
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'agenda.ui'
#
# Created by: PyQt5 UI code generator 5.12.1
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_mwindow_agenda(object):
    def setupUi(self, mwindow_agenda):
        mwindow_agenda.setObjectName("mwindow_agenda")
        mwindow_agenda.resize(960, 720)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        mwindow_agenda.setFont(font)
        self.central_widget = QtWidgets.QWidget(mwindow_agenda)
        self.central_widget.setObjectName("central_widget")
        self.gridLayout = QtWidgets.QGridLayout(self.central_widget)
        self.gridLayout.setObjectName("gridLayout")
        self.scrl_area_agenda_1 = QtWidgets.QScrollArea(self.central_widget)
        self.scrl_area_agenda_1.setFrameShape(QtWidgets.QFrame.NoFrame)
        self.scrl_area_agenda_1.setWidgetResizable(True)
        self.scrl_area_agenda_1.setObjectName("scrl_area_agenda_1")
        self.scrl_area_agenda_2 = QtWidgets.QWidget()
        self.scrl_area_agenda_2.setGeometry(QtCore.QRect(0, 0, 938, 648))
        self.scrl_area_agenda_2.setObjectName("scrl_area_agenda_2")
        self.layoutWidget = QtWidgets.QWidget(self.scrl_area_agenda_2)
        self.layoutWidget.setGeometry(QtCore.QRect(10, 16, 591, 183))
        self.layoutWidget.setObjectName("layoutWidget")
        self.vert_layout_agenda = QtWidgets.QVBoxLayout(self.layoutWidget)
        self.vert_layout_agenda.setContentsMargins(0, 0, 0, 0)
        self.vert_layout_agenda.setObjectName("vert_layout_agenda")
        self.lbl_agenda = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setPointSize(16)
        self.lbl_agenda.setFont(font)
        self.lbl_agenda.setObjectName("lbl_agenda")
        self.vert_layout_agenda.addWidget(self.lbl_agenda)
        self.hori_line_agenda = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_agenda.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_agenda.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_agenda.setObjectName("hori_line_agenda")
        self.vert_layout_agenda.addWidget(self.hori_line_agenda)
        self.vert_layout_add_task = QtWidgets.QVBoxLayout()
        self.vert_layout_add_task.setObjectName("vert_layout_add_task")
        self.btn_add_task = QtWidgets.QPushButton(self.layoutWidget)
        self.btn_add_task.setObjectName("btn_add_task")
        self.vert_layout_add_task.addWidget(
            self.btn_add_task, 0, QtCore.Qt.AlignLeft)
        self.hori_line_add_task = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_add_task.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_add_task.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_add_task.setObjectName("hori_line_add_task")
        self.vert_layout_add_task.addWidget(self.hori_line_add_task)
        self.grid_layout_tasks = QtWidgets.QGridLayout()
        self.grid_layout_tasks.setObjectName("grid_layout_tasks")
        self.checkbox_task3 = QtWidgets.QCheckBox(self.layoutWidget)
        self.checkbox_task3.setEnabled(True)
        font = QtGui.QFont()
        font.setKerning(True)
        self.checkbox_task3.setFont(font)
        self.checkbox_task3.setCheckable(True)
        self.checkbox_task3.setObjectName("checkbox_task3")
        self.grid_layout_tasks.addWidget(self.checkbox_task3, 2, 5, 1, 1)
        self.lbl_subject2 = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setItalic(False)
        self.lbl_subject2.setFont(font)
        self.lbl_subject2.setObjectName("lbl_subject2")
        self.grid_layout_tasks.addWidget(self.lbl_subject2, 1, 2, 1, 1)
        self.lbl_due_date2 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_due_date2.setObjectName("lbl_due_date2")
```

```
        self.grid_layout_tasks.addWidget(self.lbl_due_date2, 1, 4, 1, 1)
        self.lbl_task_title2 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_task_title2.setObjectName("lbl_task_title2")
        self.grid_layout_tasks.addWidget(self.lbl_task_title2, 1, 0, 1, 1)
        self.lbl_task_title3 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_task_title3.setObjectName("lbl_task_title3")
        self.grid_layout_tasks.addWidget(self.lbl_task_title3, 2, 0, 1, 1)
        self.lbl_subject3 = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setItalic(False)
        self.lbl_subject3.setFont(font)
        self.lbl_subject3.setObjectName("lbl_subject3")
        self.grid_layout_tasks.addWidget(self.lbl_subject3, 2, 2, 1, 1)
        self.lbl_subject1 = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setItalic(False)
        self.lbl_subject1.setFont(font)
        self.lbl_subject1.setObjectName("lbl_subject1")
        self.grid_layout_tasks.addWidget(self.lbl_subject1, 0, 2, 1, 1)
        self.checkbox_task1 = QtWidgets.QCheckBox(self.layoutWidget)
        self.checkbox_task1.setEnabled(True)
        font = QtGui.QFont()
        font.setKerning(True)
        self.checkbox_task1.setFont(font)
        self.checkbox_task1.setCheckable(True)
        self.checkbox_task1.setObjectName("checkbox_task1")
        self.grid_layout_tasks.addWidget(self.checkbox_task1, 0, 5, 1, 1)
        self.lbl_due_date3 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_due_date3.setObjectName("lbl_due_date3")
        self.grid_layout_tasks.addWidget(self.lbl_due_date3, 2, 4, 1, 1)
        self.lbl_task_title1 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_task_title1.setObjectName("lbl_task_title1")
        self.grid_layout_tasks.addWidget(self.lbl_task_title1, 0, 0, 1, 1)
        spacerItem = QtWidgets.QSpacerItem(
            50, 20, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Minimum)
        self.grid_layout_tasks.addItem(spacerItem, 0, 1, 1, 1)
        self.checkbox_task2 = QtWidgets.QCheckBox(self.layoutWidget)
        self.checkbox_task2.setEnabled(True)
```

```
        font = QtGui.QFont()
        font.setKerning(True)
        self.checkbox_task2.setFont(font)
        self.checkbox_task2.setCheckable(True)
        self.checkbox_task2.setObjectName("checkbox_task2")
        self.grid_layout_tasks.addWidget(self.checkbox_task2, 1, 5, 1, 1)
        spacerItem1 = QtWidgets.QSpacerItem(
            50, 20, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Minimum)
        self.grid_layout_tasks.addItem(spacerItem1, 0, 3, 1, 1)
        self.lbl_due_date1 = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_due_date1.setObjectName("lbl_due_date1")
        self.grid_layout_tasks.addWidget(self.lbl_due_date1, 0, 4, 1, 1)
        spacerItem2 = QtWidgets.QSpacerItem(
            50, 20, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Minimum)
        self.grid_layout_tasks.addItem(spacerItem2, 1, 1, 1, 1)
        spacerItem3 = QtWidgets.QSpacerItem(
            50, 20, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Minimum)
        self.grid_layout_tasks.addItem(spacerItem3, 1, 3, 1, 1)
        spacerItem4 = QtWidgets.QSpacerItem(
            50, 20, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Minimum)
        self.grid_layout_tasks.addItem(spacerItem4, 2, 1, 1, 1)
        spacerItem5 = QtWidgets.QSpacerItem(
            50, 20, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Minimum)
        self.grid_layout_tasks.addItem(spacerItem5, 2, 3, 1, 1)
        self.vert_layout_add_task.addLayout(self.grid_layout_tasks)
        self.vert_layout_agenda.addLayout(self.vert_layout_add_task)
        self.scrl_area_agenda_1.setWidget(self.scrl_area_agenda_2)
        self.gridLayout.addWidget(self.scrl_area_agenda_1, 0, 0, 1, 1)
        mwindow_agenda.setCentralWidget(self.central_widget)
        self.menu_bar = QtWidgets.QMenuBar(mwindow_agenda)
        self.menu_bar.setGeometry(QtCore.QRect(0, 0, 960, 25))
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_bar.setFont(font)
        self.menu_bar.setObjectName("menu_bar")
        self.menu_agenda = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
```

```
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_agenda.setFont(font)
        self.menu_agenda.setObjectName("menu_agenda")
        self.menu_my_subjects = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_my_subjects.setFont(font)
        self.menu_my_subjects.setObjectName("menu_my_subjects")
        self.menu_timetable = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_timetable.setFont(font)
        self.menu_timetable.setObjectName("menu_timetable")
        mwindow_agenda.setMenuBar(self.menu_bar)
        self.status_bar = QtWidgets.QStatusBar(mwindow_agenda)
        self.status_bar.setObjectName("status_bar")
        mwindow_agenda.setStatusBar(self.status_bar)
        self.menu_bar.addAction(self.menu_my_subjects.menuAction())
        self.menu_bar.addAction(self.menu_agenda.menuAction())
        self.menu_bar.addAction(self.menu_timetable.menuAction())

        self.retranslateUi(mwindow_agenda)
        QtCore.QMetaObject.connectSlotsByName(mwindow_agenda)

    def retranslateUi(self, mwindow_agenda):
        _translate = QtCore.QCoreApplication.translate
        mwindow_agenda.setWindowTitle(_translate("mwindow_agenda", "Agenda"))
        self.lbl_agenda.setText(_translate("mwindow_agenda", "Agenda"))
        self.btn_add_task.setText(_translate("mwindow_agenda", "Add Task"))
        self.checkbox_task3.setText(_translate("mwindow_agenda", "Incomplete"))
        self.lbl_subject2.setText(_translate(
            "mwindow_agenda", "Further Mathematics"))
        self.lbl_due_date2.setText(_translate("mwindow_agenda", "15/03"))
        self.lbl_task_title2.setText(
            _translate("mwindow_agenda", "Algorithms"))
        self.lbl_task_title3.setText(
            _translate("mwindow_agenda", "Probability"))
        self.lbl_subject3.setText(_translate("mwindow_agenda", "Mathematics"))
        self.lbl_subject1.setText(_translate(
            "mwindow_agenda", "Computer Science"))
        self.checkbox_task1.setText(_translate("mwindow_agenda", "Incomplete"))
        self.lbl_due_date3.setText(_translate("mwindow_agenda", "15/03"))
        self.lbl_task_title1.setText(_translate(
            "mwindow_agenda", "Python Programming"))
        self.checkbox_task2.setText(_translate("mwindow_agenda", "Incomplete"))
        self.lbl_due_date1.setText(_translate("mwindow_agenda", "13/03"))
        self.menu_agenda.setTitle(_translate("mwindow_agenda", "Agenda"))
        self.menu_my_subjects.setTitle(
            _translate("mwindow_agenda", "My Subjects"))
        self.menu_timetable.setTitle(_translate("mwindow_agenda", "Timetable"))
```

To ensure that my operational code worked with the generated code, I used the *from … import …* function to import the file which contains the generated code, *agenda_setup.py*.

```
from add_task_setup import Ui_dialog_new_task
from agenda_setup import Ui_mwindow_agenda
```

I imported the *Ui_dialog_new_task* class from *add_task_setup.py* so that the operational program can initialise the widgets from that dialog window, and I also imported the *Ui_mwindow_agenda* class from *agenda_setup.py*, which allows the operational program to initialise the widgets from the main window.

```
# Setting up the Add Task dialog.


class AddTaskDialog(QDialog, Ui_dialog_new_task):
    def __init__(self):
        super().__init__()
        self.setupUi(self)

# Setting up the Agenda main window.


class AgendaWindow(QMainWindow, Ui_mwindow_agenda):
    def __init__(self):
        super().__init__()
        self.setupUi(self)

        # Connects 'Add Task' button to the Add Task dialog.
        self.btn_add_task.clicked.connect(self.open_dialog_add_task)

    # Opens the dialog for the user to add a task.
    def open_dialog_add_task(self):
        self.Dialog = AddTaskDialog()
        self.Dialog.open()
```

I created new classes in the operational code which inherit from the classes I imported. It initialises the widgets from these classes, so that the elements of the user interface can be displayed.

AddTaskDialog inherits from *QDialog* because the window for that class is a dialog window (as opposed to a main window), and it also inherits from *Ui_dialog_new_task*, as this is the class which initialises the widgets for adding a new task.

AgendaWindow inherits from *QMainWindow* because the window for that class is a main window, and it also inherits from *Ui_mwindow_agenda*, as this is the class which initialises the widgets for the main window of Agenda.

The full operational code in *agenda.py* can be seen below.

```python
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QDialog, QMainWindow

from add_task_setup import Ui_dialog_new_task
from agenda_setup import Ui_mwindow_agenda

# Setting up the Add Task dialog.


class AddTaskDialog(QDialog, Ui_dialog_new_task):
    def __init__(self):
        super().__init__()
        self.setupUi(self)

# Setting up the Agenda main window.


class AgendaWindow(QMainWindow, Ui_mwindow_agenda):
    def __init__(self):
        super().__init__()
        self.setupUi(self)

        # Connects 'Add Task' button to the Add Task dialog.
        self.btn_add_task.clicked.connect(self.open_dialog_add_task)

    # Opens the dialog for the user to add a task.
    def open_dialog_add_task(self):
        self.Dialog = AddTaskDialog()
        self.Dialog.open()


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    mwindow_agenda = AgendaWindow()
    mwindow_agenda.show()
    sys.exit(app.exec_())

```

As seen above, I also renamed *add_task.py* to *add_task_setup.py* to indicate that the Python code inside that file only consists of automatically generated code, and no operational code. The code itself did not change, and I did not have to create an additional Python file to contain the operational code for this dialog window, as it will be added to the operational code for the main window, *agenda.py*, instead.

After my experiences from stage 6, I realised that the method I was planning on using to store the list of tasks in Agenda would not be appropriate, as it would be difficult to scale depending on the number of tasks in the list. Hence, I updated the user interface in Qt Creator to use a *QTableWidget* rather than using a lot of labels as I had been previously.

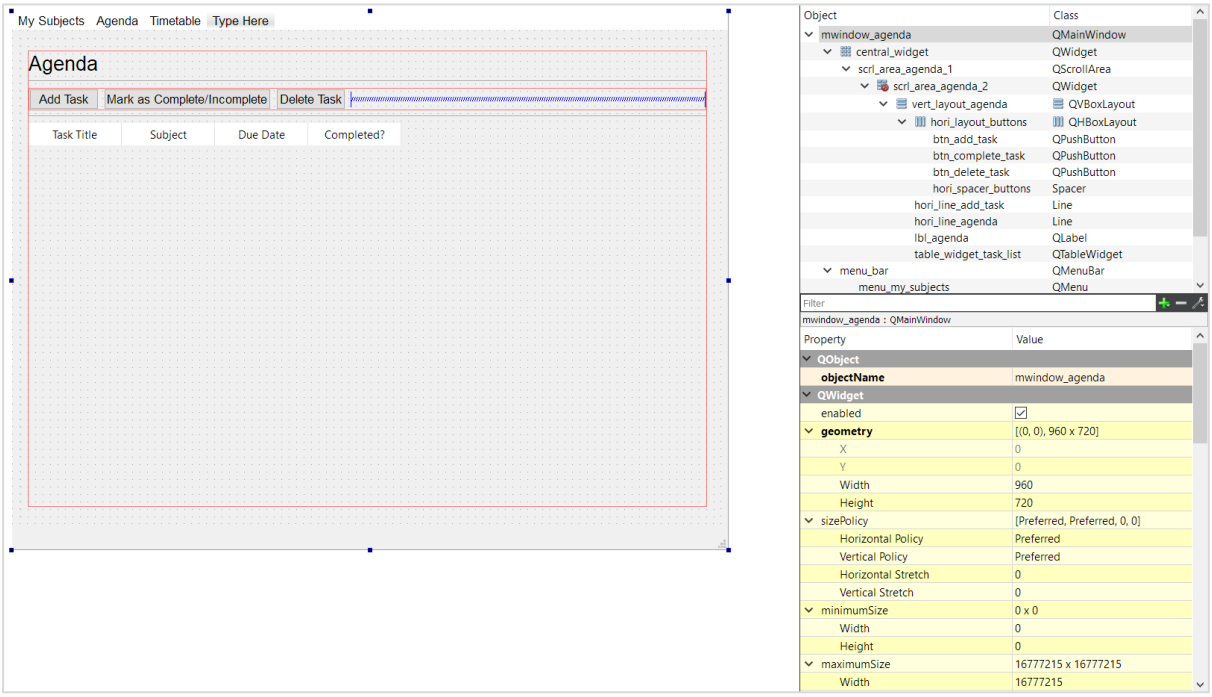
As I was updating the user interface to include the *QTableWidget*, I noticed that I was using a vertical layout, *vert_layout_add_task*, inside another vertical layout, *vert_layout_agenda*. This was unnecessary and simply added more code to my program with no effect, so I removed *vert_layout_add_task*, and moved all the widgets inside that vertical layout to the bottom of *vert_layout_agenda*. The aesthetic of my user interface remained the same after I decluttered it from the extra vertical layout.

In addition, I added a button for marking a task as complete/incomplete and for deleting a task. I called these buttons *btn_complete_task* and *btn_delete_task*. For marking the task as complete/incomplete, I had the option of using two separate buttons, but I decided that this would be a waste of space in the user interface, and that it would be more user friendly to have the button act as a toggle for marking the task as complete or incomplete.

To add these buttons, I placed a horizontal layout where *btn_add_task* was, and placed the button in this layout. Then, I placed *btn_complete_task* and *btn_delete_task* on the right of

the existing button, and added a horizontal spacer on the far right side of these buttons to position the buttons next to each other with left alignment.

The functionality for these buttons will not be coded in this development iteration, but the buttons have been added for use in later development iterations.



Once again, I converted this user interface to a Python file by using *pyuic5* to automatically generate the code.

```python
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'agenda.ui'
#
# Created by: PyQt5 UI code generator 5.12.1
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_mwindow_agenda(object):
    def setupUi(self, mwindow_agenda):
        mwindow_agenda.setObjectName("mwindow_agenda")
        mwindow_agenda.resize(960, 720)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        mwindow_agenda.setFont(font)
        self.central_widget = QtWidgets.QWidget(mwindow_agenda)
        self.central_widget.setObjectName("central_widget")
        self.gridLayout = QtWidgets.QGridLayout(self.central_widget)
        self.gridLayout.setObjectName("gridLayout")
        self.scrl_area_agenda_1 = QtWidgets.QScrollArea(self.central_widget)
        self.scrl_area_agenda_1.setFrameShape(QtWidgets.QFrame.NoFrame)
        self.scrl_area_agenda_1.setWidgetResizable(True)
        self.scrl_area_agenda_1.setObjectName("scrl_area_agenda_1")
        self.scrl_area_agenda_2 = QtWidgets.QWidget()
        self.scrl_area_agenda_2.setGeometry(QtCore.QRect(0, 0, 938, 648))
        self.scrl_area_agenda_2.setObjectName("scrl_area_agenda_2")
        self.layoutWidget = QtWidgets.QWidget(self.scrl_area_agenda_2)
        self.layoutWidget.setGeometry(QtCore.QRect(10, 16, 911, 611))
        self.layoutWidget.setObjectName("layoutWidget")
        self.vert_layout_agenda = QtWidgets.QVBoxLayout(self.layoutWidget)
        self.vert_layout_agenda.setContentsMargins(0, 0, 0, 0)
        self.vert_layout_agenda.setObjectName("vert_layout_agenda")
        self.lbl_agenda = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
```

```
        font.setPointSize(16)
        self.lbl_agenda.setFont(font)
        self.lbl_agenda.setObjectName("lbl_agenda")
        self.vert_layout_agenda.addWidget(self.lbl_agenda)
        self.hori_line_agenda = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_agenda.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_agenda.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_agenda.setObjectName("hori_line_agenda")
        self.vert_layout_agenda.addWidget(self.hori_line_agenda)
        self.hori_layout_buttons = QtWidgets.QHBoxLayout()
        self.hori_layout_buttons.setObjectName("hori_layout_buttons")
        self.btn_add_task = QtWidgets.QPushButton(self.layoutWidget)
        self.btn_add_task.setObjectName("btn_add_task")
        self.hori_layout_buttons.addWidget(
            self.btn_add_task, 0, QtCore.Qt.AlignLeft)
        self.btn_complete_task = QtWidgets.QPushButton(self.layoutWidget)
        self.btn_complete_task.setObjectName("btn_complete_task")
        self.hori_layout_buttons.addWidget(self.btn_complete_task)
        self.btn_delete_task = QtWidgets.QPushButton(self.layoutWidget)
        self.btn_delete_task.setObjectName("btn_delete_task")
        self.hori_layout_buttons.addWidget(self.btn_delete_task)
        spacerItem = QtWidgets.QSpacerItem(
            40, 20, QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Minimum)
        self.hori_layout_buttons.addItem(spacerItem)
        self.vert_layout_agenda.addLayout(self.hori_layout_buttons)
        self.hori_line_add_task = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_add_task.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_add_task.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_add_task.setObjectName("hori_line_add_task")
        self.vert_layout_agenda.addWidget(self.hori_line_add_task)
        self.table_widget_task_list = QtWidgets.QTableWidget(self.layoutWidget)
        palette = QtGui.QPalette()
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Button, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.NoBrush)
        palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Base, brush)
```

```
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Window, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Button, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.NoBrush)
        palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Base, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Window, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Button, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.NoBrush)
        palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Base, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Window, brush)
        self.table_widget_task_list.setPalette(palette)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.table_widget_task_list.setFont(font)
        self.table_widget_task_list.setAutoFillBackground(False)
        self.table_widget_task_list.setStyleSheet(
            "background-color: transparent")
        self.table_widget_task_list.setFrameShape(QtWidgets.QFrame.NoFrame)
        self.table_widget_task_list.setLineWidth(0)
        self.table_widget_task_list.setSizeAdjustPolicy(
            QtWidgets.QAbstractScrollArea.AdjustToContents)
        self.table_widget_task_list.setSelectionBehavior(
            QtWidgets.QAbstractItemView.SelectRows)
        self.table_widget_task_list.setShowGrid(False)
        self.table_widget_task_list.setObjectName("table_widget_task_list")
        self.table_widget_task_list.setColumnCount(4)
```

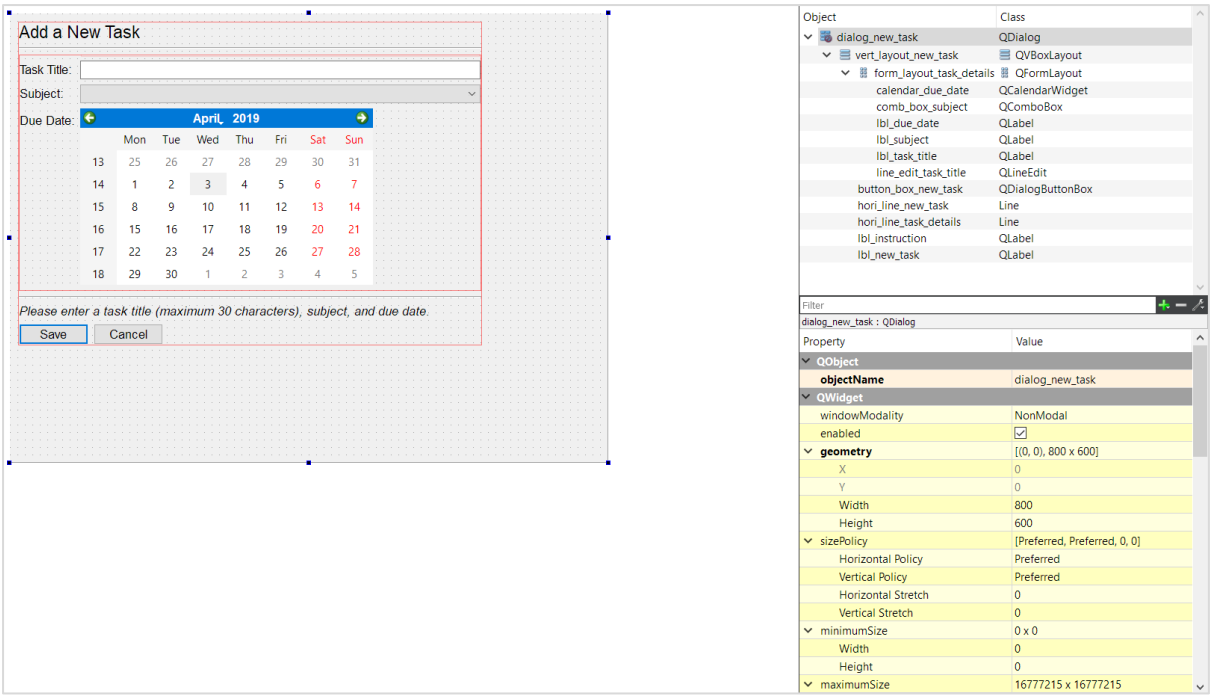
```
        self.table_widget_task_list.setRowCount(0)
        item = QtWidgets.QTableWidgetItem()
        item.setBackground(QtGui.QColor(0, 0, 0, 0))
        self.table_widget_task_list.setHorizontalHeaderItem(0, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_task_list.setHorizontalHeaderItem(1, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_task_list.setHorizontalHeaderItem(2, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_task_list.setHorizontalHeaderItem(3, item)
        self.table_widget_task_list.horizontalHeader().setVisible(True)
        self.table_widget_task_list.horizontalHeader().setHighlightSections(True)
        self.table_widget_task_list.horizontalHeader().setStretchLastSection(False)
        self.table_widget_task_list.verticalHeader().setVisible(True)
        self.vert_layout_agenda.addWidget(self.table_widget_task_list)
        self.scrl_area_agenda_1.setWidget(self.scrl_area_agenda_2)
        self.gridLayout.addWidget(self.scrl_area_agenda_1, 0, 0, 1, 1)
        mwindow_agenda.setCentralWidget(self.central_widget)
        self.menu_bar = QtWidgets.QMenuBar(mwindow_agenda)
        self.menu_bar.setGeometry(QtCore.QRect(0, 0, 960, 25))
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_bar.setFont(font)
        self.menu_bar.setObjectName("menu_bar")
        self.menu_agenda = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_agenda.setFont(font)
        self.menu_agenda.setObjectName("menu_agenda")
        self.menu_my_subjects = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_my_subjects.setFont(font)
        self.menu_my_subjects.setObjectName("menu_my_subjects")
        self.menu_timetable = QtWidgets.QMenu(self.menu_bar)
```

```
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_timetable.setFont(font)
        self.menu_timetable.setObjectName("menu_timetable")
        mwindow_agenda.setMenuBar(self.menu_bar)
        self.status_bar = QtWidgets.QStatusBar(mwindow_agenda)
        self.status_bar.setObjectName("status_bar")
        mwindow_agenda.setStatusBar(self.status_bar)
        self.menu_bar.addAction(self.menu_my_subjects.menuAction())
        self.menu_bar.addAction(self.menu_agenda.menuAction())
        self.menu_bar.addAction(self.menu_timetable.menuAction())

        self.retranslateUi(mwindow_agenda)
        QtCore.QMetaObject.connectSlotsByName(mwindow_agenda)
```

```

    def retranslateUi(self, mwindow_agenda):
        _translate = QtCore.QCoreApplication.translate
        mwindow_agenda.setWindowTitle(_translate("mwindow_agenda", "Agenda"))
        self.lbl_agenda.setText(_translate("mwindow_agenda", "Agenda"))
        self.btn_add_task.setText(_translate("mwindow_agenda", "Add Task"))
        self.btn_complete_task.setText(_translate(
            "mwindow_agenda", "Mark as Complete/Incomplete"))
        self.btn_delete_task.setText(
            _translate("mwindow_agenda", "Delete Task"))
        self.table_widget_task_list.setSortingEnabled(True)
        item = self.table_widget_task_list.horizontalHeaderItem(0)
        item.setText(_translate("mwindow_agenda", "Task Title"))
        item = self.table_widget_task_list.horizontalHeaderItem(1)
        item.setText(_translate("mwindow_agenda", "Subject"))
        item = self.table_widget_task_list.horizontalHeaderItem(2)
        item.setText(_translate("mwindow_agenda", "Due Date"))
        item = self.table_widget_task_list.horizontalHeaderItem(3)
        item.setText(_translate("mwindow_agenda", "Completed?"))
        self.menu_agenda.setTitle(_translate("mwindow_agenda", "Agenda"))
        self.menu_my_subjects.setTitle(
            _translate("mwindow_agenda", "My Subjects"))
        self.menu_timetable.setTitle(_translate("mwindow_agenda", "Timetable"))

```

I also edited the user interface for adding a task slightly to include an instruction label for the user. This will be useful when validation is performed.

By default, the label asks the user to enter a task title (maximum 30 characters), subject, and due date. If validation is failed, it will inform them which input failed validation, why it failed, and it will ask them to try again.



I was happy with the change, so I converted the updated user interface into Python with *pyuic5*.

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'add_task.ui'
#
# Created by: PyQt5 UI code generator 5.12.1
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_dialog_new_task(object):
    def setupUi(self, dialog_new_task):
        dialog_new_task.setObjectName("dialog_new_task")
        dialog_new_task.resize(800, 600)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        dialog_new_task.setFont(font)
        self.layoutWidget = QtWidgets.QWidget(dialog_new_task)
        self.layoutWidget.setGeometry(QtCore.QRect(11, 11, 621, 433))
        self.layoutWidget.setObjectName("layoutWidget")
        self.vert_layout_new_task = QtWidgets.QVBoxLayout(self.layoutWidget)
        self.vert_layout_new_task.setContentsMargins(0, 0, 0, 0)
        self.vert_layout_new_task.setObjectName("vert_layout_new_task")
        self.lbl_new_task = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setPointSize(14)
        self.lbl_new_task.setFont(font)
        self.lbl_new_task.setObjectName("lbl_new_task")
        self.vert_layout_new_task.addWidget(self.lbl_new_task)
        self.hori_line_new_task = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_new_task.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_new_task.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_new_task.setObjectName("hori_line_new_task")
        self.vert_layout_new_task.addWidget(self.hori_line_new_task)
        self.form_layout_task_details = QtWidgets.QFormLayout()
        self.form_layout_task_details.setObjectName("form_layout_task_details")
```

```
        self.lbl_task_title = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_task_title.setObjectName("lbl_task_title")
        self.form_layout_task_details.setWidget(
            1, QtWidgets.QFormLayout.LabelRole, self.lbl_task_title)
        self.line_edit_task_title = QtWidgets.QLineEdit(self.layoutWidget)
        self.line_edit_task_title.setObjectName("line_edit_task_title")
        self.form_layout_task_details.setWidget(
            1, QtWidgets.QFormLayout.FieldRole, self.line_edit_task_title)
        self.lbl_subject = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_subject.setObjectName("lbl_subject")
        self.form_layout_task_details.setWidget(
            2, QtWidgets.QFormLayout.LabelRole, self.lbl_subject)
        self.comb_box_subject = QtWidgets.QComboBox(self.layoutWidget)
        self.comb_box_subject.setFrame(True)
        self.comb_box_subject.setObjectName("comb_box_subject")
        self.form_layout_task_details.setWidget(
            2, QtWidgets.QFormLayout.FieldRole, self.comb_box_subject)
        self.lbl_due_date = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_due_date.setObjectName("lbl_due_date")
        self.form_layout_task_details.setWidget(
            3, QtWidgets.QFormLayout.LabelRole, self.lbl_due_date)
        self.calendar_due_date = QtWidgets.QCalendarWidget(self.layoutWidget)
        sizePolicy = QtWidgets.QSizePolicy(
            QtWidgets.QSizePolicy.Fixed, QtWidgets.QSizePolicy.Preferred)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(
            self.calendar_due_date.sizePolicy().hasHeightForWidth())
        self.calendar_due_date.setSizePolicy(sizePolicy)
        self.calendar_due_date.setObjectName("calendar_due_date")
        self.form_layout_task_details.setWidget(
            3, QtWidgets.QFormLayout.FieldRole, self.calendar_due_date)
        self.vert_layout_new_task.addLayout(self.form_layout_task_details)
        self.hori_line_task_details = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_task_details.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_task_details.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_task_details.setObjectName("hori_line_task_details")
        self.vert_layout_new_task.addWidget(self.hori_line_task_details)
```

```
        self.lbl_instruction = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setItalic(True)
        self.lbl_instruction.setFont(font)
        self.lbl_instruction.setObjectName("lbl_instruction")
        self.vert_layout_new_task.addWidget(self.lbl_instruction)
        self.button_box_new_task = QtWidgets.QDialogButtonBox(
            self.layoutWidget)
        self.button_box_new_task.setOrientation(QtCore.Qt.Horizontal)
        self.button_box_new_task.setStandardButtons(
            QtWidgets.QDialogButtonBox.Cancel | QtWidgets.QDialogButtonBox.Save)
        self.button_box_new_task.setObjectName("button_box_new_task")
        self.vert_layout_new_task.addWidget(
            self.button_box_new_task, 0, QtCore.Qt.AlignLeft)

        self.retranslateUi(dialog_new_task)
        self.button_box_new_task.rejected.connect(dialog_new_task.reject)
        self.button_box_new_task.accepted.connect(dialog_new_task.accept)
        QtCore.QMetaObject.connectSlotsByName(dialog_new_task)

    def retranslateUi(self, dialog_new_task):
        _translate = QtCore.QCoreApplication.translate
        dialog_new_task.setWindowTitle(
            _translate("dialog_new_task", "Add Task"))
        self.lbl_new_task.setText(_translate(
            "dialog_new_task", "Add a New Task"))
        self.lbl_task_title.setText(
            _translate("dialog_new_task", "Task Title:"))
        self.lbl_subject.setText(_translate("dialog_new_task", "Subject:"))
        self.lbl_due_date.setText(_translate("dialog_new_task", "Due Date:"))
        self.lbl_instruction.setText(_translate(
            "dialog_new_task", "Please enter a task title (maximum 30 characters), subject, and due date."))

```

After I made changes to the user interfaces, I started to develop some functionality within Agenda by working on *agenda.py*.

First, I defined *task_list* as a dictionary variable so that the program is able to work with this variable to gather data about the tasks in the agenda. This needed to be coded outside of the classes, as other functions which use *task_list* will not work if this variable is undefined.

```
# Identifies the task list as a dictionary variable.
task_list = {}
```

Outside of the classes, I also coded the program to read *task_list.json*.

If there is data in the file, the program loads the dictionary into the variable *task_list*, then prints the dictionary in pretty print format to the console for debugging purposes using *json.dump*. I used *sys.stdout* to output the contents to the console, *ensure_ascii=False* to ensure that JSON does not change non-ASCII characters (which may be used when students input characters for foreign language subjects such as Chinese), and *indent=4* to format the dictionary in pretty print for easier reading.

In cases where there is no data in the JSON file, such as when all tasks have been deleted from the agenda, the program prints 'Empty JSON file' to the console for debugging purposes, as this will help show that the program has not found any data in the file.

```
13  # Reads existing JSON file for list of tasks.
14  with open('task_list.json', 'r') as outfile:
15      try:
16          task_list = json.load(outfile)
17          json.dump(task_list, sys.stdout, ensure_ascii=False, indent=4)
18      except ValueError:
19          print("Empty JSON file.")
```

In the *AgendaWindow* class, I made a lot of changes.

First, I used the command *.setStyleSheet()* to change the background colour of the *QTableWidget* manually, as I was unable to change the background colour of the header of the table to be transparent in Qt Creator. The use of the style sheet meant that I was able to make the entire table transparent, as opposed to only the bottom part of the table which contains the data. This code is run when the window is initialised, as it is important to format the table.

```
37          # Sets the list of tasks to have a transparent background.
38          self.setStyleSheet("""QTableWidget {background-color: transparent;}
39              QHeaderView::section {background-color: transparent;}
40              QHeaderView {background-color: transparent;}""")
```

Then, I connected the 'Add Task' button to the method *open_dialog_add_task()*, which will contain the code which is used to start the dialog window for adding a new task.

```
42          # Connects 'Add Task' button to the Add Task dialog.
43          self.btn_add_task.clicked.connect(self.open_dialog_add_task)
```

Finally, I called the *update_list()* method, which populates the list on start-up using the existing JSON file data.

```
45          # Populates list on start-up.
46          self.update_list()
```

The method *update_list()* is responsible for populating the table widget by reading the contents of the JSON file for the data.

```
    # Reads the existing JSON file and populates the table widget.
    def update_list(self):
        # Sets number of rows, column size, and row size.
        self.table_widget_task_list.setRowCount(len(task_list))
        self.table_widget_task_list.horizontalHeader().setSectionResizeMode(
            QtWidgets.QHeaderView.ResizeToContents)
        self.table_widget_task_list.verticalHeader().setSectionResizeMode(
            QtWidgets.QHeaderView.Fixed)

        # Adds task titles to the agenda.
        current_row = 0
        for task, details in task_list.items():
            current_column = 1
            self.table_widget_task_list.setItem(
                current_row, 0, QtWidgets.QTableWidgetItem(task))

            # Adds task details to the agenda.
            for detail in details.values():
                self.table_widget_task_list.setItem(
                    current_row, current_column,
                    QtWidgets.QTableWidgetItem(detail))
                current_column += 1

            current_row += 1
```

At the start of the method, the program sets the number of rows in the table widget, *table_widget_task_list*, to be the same number of keys as in *task_list*. This is because there must be one row for each task, with the task details displayed across the columns.

The method resizes the columns to fit the contents using *.ResizeToContents* in the *.setSectionResizeMode()* for the horizontal header, which together with the character limits for the task titles, prevents the table widget from having to display more than one line per row. This ensures a clean aesthetic for the user interface, making it easier to read information.

For the vertical header, *Fixed* is used with *.setSectionResizeMode()*, as the table will only need enough vertical space for one line of text per row due to the restrictions explained above.

```
    def update_list(self):
        # Sets number of rows, column size, and row size.
        self.table_widget_task_list.setRowCount(len(task_list))
        self.table_widget_task_list.horizontalHeader().setSectionResizeMode(
            QtWidgets.QHeaderView.ResizeToContents)
        self.table_widget_task_list.verticalHeader().setSectionResizeMode(
            QtWidgets.QHeaderView.Fixed)
```

I performed an iteration to add the task titles and details to the agenda's table widget, with a *for* loop nested inside another *for* loop.

The current row is set to 0 at the start, so that the program will add items starting from the top to bottom of the table. It iterates from the items of *task_list* using the function *.items()*, and it uses *.setItem()* to add each task title to the current row on the first column (0 representing the first column).

In that row, the method iterates between the columns. It starts off with column 1, which represents the second column, for the task details. Then, it adds the task details to each cell with this iteration. The code for this is adaptable, meaning that if more details are added in the future (as opposed to just subject, due date, and completion), the code will still work, as it reads from the file to see how many values there are.

At the end of the nested *for* loop, the *current_column* variable is incremented by one using *current_column += 1*. The same is performed for the *current_row* variable at the end of the first *for* loop, using *current_row += 1.*

```
57          # Adds task titles to the agenda.
58          current_row = 0
59          for task, details in task_list.items():
60              current_column = 1
61              self.table_widget_task_list.setItem(
62                  current_row, 0, QtWidgets.QTableWidgetItem(task))
63
64              # Adds task details to the agenda.
65              for detail in details.values():
66                  self.table_widget_task_list.setItem(
67                      current_row, current_column,
68                      QtWidgets.QTableWidgetItem(detail))
69                  current_column += 1
70
71              current_row += 1
```

The method *open_dialog_add_task()* initialises some of the functionality for the dialog window, then displays it to the user.

```
73      # Opens the dialog for the user to add a task.
74      def open_dialog_add_task(self):
75          self.Dialog = AddTaskDialog()
76
77          # Connects 'Save' button to save the user's task to Agenda.
78          self.Dialog.button_box_new_task.accepted.disconnect()
79          self.Dialog.button_box_new_task.accepted.connect(self.save_task)
80
81          # Populates the combo box with subject options.
82          with open("subject_list.txt", "r") as data_file:
83              subject_list = data_file.readlines()
84              for line in subject_list:
85                  self.Dialog.comb_box_subject.addItem(line.strip("\n"))
86
87          self.Dialog.open()
```

I disconnected the ‘Save’ button from the built-in method which causes the dialog to close whenever it is pressed, as validation will need to be performed first. Then, I connected it to the method *save_task()*, which will be explained later.

```
77          # Connects 'Save' button to save the user's task to Agenda.
78          self.Dialog.button_box_new_task.accepted.disconnect()
79          self.Dialog.button_box_new_task.accepted.connect(self.save_task)
```

The combo box is populated with the subject options by reading from *subject_list.txt*, the file created to store the user’s subjects from My Subjects. The code reads the lines from the text file, then adds each item to the combo box in sequence, stripping the new lines (as these were included to separate the items in the text file).

```
81          # Populates the combo box with subject options.
82          with open("subject_list.txt", "r") as data_file:
83              subject_list = data_file.readlines()
84              for line in subject_list:
85                  self.Dialog.comb_box_subject.addItem(line.strip("\n"))
```

After the combo box has been populated, the dialog is displayed to the user using the function *open()*.

```
87          self.Dialog.open()
```

The method *save_task()* has the purpose of gathering the task details, validating them, and saving them if validation is passed.

```
89      # Saves new tasks to the agenda and JSON file.
90      def save_task(self):
91          # Gets user inputs for subject and due date, and assigns task as incomplete.
92          new_subject = self.Dialog.comb_box_subject.currentText()
93          new_due_date = self.Dialog.calendar_due_date.selectedDate().toString("dd-MM-yyyy")
94          new_completed = ("No")
95
96          # Validates task title input and adds task if validation passed.
97          new_task_title = self.Dialog.line_edit_task_title.text()
98          if len(new_task_title) <= 30 and len((new_task_title).strip(" ")) > 0:
99              # Adds new task as a key with task details as values.
100             task_list[new_task_title] = {
101                 "subject": new_subject,
102                 "due_date": new_due_date,
103                 "completed": new_completed}
104             # Updates the JSON file with the updated task list dictionary.
105             with open('task_list.json', 'w') as outfile:
106                 json.dump(task_list, outfile, ensure_ascii=False, indent=4)
107             # Closes the dialog and updates the task list.
108             self.Dialog.close()
109             self.update_list()
110         # Rejects input if no task title entered.
111         elif len((new_task_title).strip(" ")) == 0:
112             self.Dialog.lbl_instruction.setText(
113                 "You have not entered a task title. Please try again.")
114         # Rejects input if task title exceeds 30 characters.
115         else:
116             self.Dialog.lbl_instruction.setText(
117                 "Your task title exceeds 30 characters. Please try again.")
```


```
89      # Saves new tasks to the agenda and JSON file.
90      def save_task(self):
91          # Gets user inputs for subject and due date, and assigns task as incomplete.
92          new_subject = self.Dialog.comb_box_subject.currentText()
93          new_due_date = self.Dialog.calendar_due_date.selectedDate().toString("dd-MM-yyyy")
94          new_completed = ("No")
95
96          # Validates task title input and adds task if validation passed.
97          new_task_title = self.Dialog.line_edit_task_title.text()
98          if len(new_task_title) <= 30 and len((new_task_title).strip(" ")) > 0:
99              # Adds new task as a key with task details as values.
100             task_list[new_task_title] = {
101                 "subject": new_subject,
102                 "due_date": new_due_date,
103                 "completed": new_completed}
104             # Updates the JSON file with the updated task list dictionary.
105             with open('task_list.json', 'w') as outfile:
106                 json.dump(task_list, outfile, ensure_ascii=False, indent=4)
107             # Closes the dialog and updates the task list.
108             self.Dialog.close()
109             self.update_list()
110         # Rejects input if no task title entered.
111         elif len((new_task_title).strip(" ")) == 0:
112             self.Dialog.lbl_instruction.setText(
113                 "You have not entered a task title. Please try again.")
114         # Rejects input if task title exceeds 30 characters.
115         else:
116             self.Dialog.lbl_instruction.setText(
117                 "Your task title exceeds 30 characters. Please try again.")
```

To obtain the subject name, I used the command *.currentText()*, which obtains the current selection from the combo box. In the design section, I mentioned that this would be validated, but the combo box chooses the first option as the default to select, so there would be no cases in which there is no option selected, which is the case a presence check would cover.

```
89      # Saves new tasks to the agenda and JSON file.
90      def save_task(self):
91          # Gets user inputs for subject and due date, and assigns task as incomplete.
92          new_subject = self.Dialog.comb_box_subject.currentText()
```

The due date is obtained by using the commands *.selectedDate()* followed by *.toString()*. The first of these commands identifies the date which the user selected, and the second command converts the data into a string. By default, this string is formatted in a more verbose style, stating the day of the week, month, day of the month, and the year. Instead of using the default format, I used *.toString("dd-MM-yyyy")*, which will display the date by the day, month, and year (with leading zeroes). For example, the 4th December 2019 would be displayed as '04-12-2019', for example. This is because it will be easier for the user to compare the due dates of various tasks with this format.

```
93          new_due_date = self.Dialog.calendar_due_date.selectedDate().toString("dd-MM-yyyy")
```

The final detail of each task at this point is the completion. As the user will have just added this task to the agenda, the task will automatically be marked as uncompleted.

```
94          new_completed = ("No")
```

Before these task detail variables are saved to the JSON file, validation is performed on the line edit widget. Similar validation occurs for the task title as in development iteration #3 of stage 6.

```
96          # Validates task title input and adds task if validation passed.
97          new_task_title = self.Dialog.line_edit_task_title.text()
98          if len(new_task_title) <= 30 and len((new_task_title).strip(" ")) > 0:
99              # Adds new task as a key with task details as values.
100             task_list[new_task_title] = {
101                 "subject": new_subject,
102                 "due_date": new_due_date,
103                 "completed": new_completed}
104             # Updates the JSON file with the updated task list dictionary.
105             with open('task_list.json', 'w') as outfile:
106                 json.dump(task_list, outfile, ensure_ascii=False, indent=4)
107             # Closes the dialog and updates the task list.
108             self.Dialog.close()
109             self.update_list()
110         # Rejects input if no task title entered.
111         elif len((new_task_title).strip(" ")) == 0:
112             self.Dialog.lbl_instruction.setText(
113                 "You have not entered a task title. Please try again.")
114         # Rejects input if task title exceeds 30 characters.
115         else:
116             self.Dialog.lbl_instruction.setText(
117                 "Your task title exceeds 30 characters. Please try again.")
```

As stated in the key variables and data structures in the design section, the task title limit will have a character limit of 30, so this method validates whether the user has entered a task title which is less than or equal to 30 characters, and greater than 0 characters when white space is removed. If this is the case, it will add the task title to the dictionary as a key, and the task details as the values for that key. Then, it writes the task details to a JSON file to update it, before closing the dialog.

```
98          if len(new_task_title) <= 30 and len((new_task_title).strip(" ")) > 0:
99              # Adds new task as a key with task details as values.
100             task_list[new_task_title] = {
101                 "subject": new_subject,
102                 "due_date": new_due_date,
103                 "completed": new_completed}
104             # Updates the JSON file with the updated task list dictionary.
105             with open('task_list.json', 'w') as outfile:
106                 json.dump(task_list, outfile, ensure_ascii=False, indent=4)
107             # Closes the dialog and updates the task list.
108             self.Dialog.close()
109             self.update_list()
```

I used the *elif* statement for cases where the user has not entered a task title, which will be validated by checking whether the task title with the white space removed has 0 characters. This will change the instruction label to inform the user that they have not entered a task title, and to try again.

```
110         # Rejects input if no task title entered.
111         elif len((new_task_title).strip(" ")) == 0:
112             self.Dialog.lbl_instruction.setText(
113                 "You have not entered a task title. Please try again.")
```

The other cases are where the user has entered a task title which exceeds 30 characters. I used the *else* statement to handle these cases, and it will change the instruction label to notify the user that their task title exceeds 30 characters, and to try again.

```
114         # Rejects input if task title exceeds 30 characters.
115         else:
116             self.Dialog.lbl_instruction.setText(
117                 "Your task title exceeds 30 characters. Please try again.")
```

Throughout the program, I ensured that I paid special attention to documenting the code using comments, as I had made an especially large number of changes in this development iteration.

Once I had made the changes to the operational code for Agenda, the full code in *agenda.py* looked like the following.

```python
import json
import sys

from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QDialog, QMainWindow

from add_task_setup import Ui_dialog_new_task
from agenda_setup import Ui_mwindow_agenda

# Identifies the task list as a dictionary variable.
task_list = {}

# Reads existing JSON file for list of tasks.
with open('task_list.json', 'r') as outfile:
    try:
        task_list = json.load(outfile)
        json.dump(task_list, sys.stdout, ensure_ascii=False, indent=4)
    except ValueError:
        print("Empty JSON file.")

# Sets up the Add Task dialog.


class AddTaskDialog(QDialog, Ui_dialog_new_task):
    def __init__(self):
        super().__init__()
        self.setupUi(self)

# Sets up the Agenda main window.


class AgendaWindow(QMainWindow, Ui_mwindow_agenda):
    def __init__(self):
        super().__init__()
        self.setupUi(self)

        # Sets the list of tasks to have a transparent background.
        self.setStyleSheet("""QTableWidget {background-color: transparent;}
            QHeaderView::section {background-color: transparent;}
            QHeaderView {background-color: transparent;}""")

        # Connects 'Add Task' button to the Add Task dialog.
        self.btn_add_task.clicked.connect(self.open_dialog_add_task)

        # Populates list on start-up.
        self.update_list()

    # Reads the existing JSON file and populates the table widget.
    def update_list(self):
        # Sets number of rows, column size, and row size.
        self.table_widget_task_list.setRowCount(len(task_list))
        self.table_widget_task_list.horizontalHeader().setSectionResizeMode(
            QtWidgets.QHeaderView.ResizeToContents)
        self.table_widget_task_list.verticalHeader().setSectionResizeMode(
            QtWidgets.QHeaderView.Fixed)

        # Adds task titles to the agenda.
        current_row = 0
        for task, details in task_list.items():
            current_column = 1
            self.table_widget_task_list.setItem(
                current_row, 0, QtWidgets.QTableWidgetItem(task))

            # Adds task details to the agenda.
            for detail in details.values():
                self.table_widget_task_list.setItem(
                    current_row, current_column,
                    QtWidgets.QTableWidgetItem(detail))
                current_column += 1

            current_row += 1

    # Opens the dialog for the user to add a task.
    def open_dialog_add_task(self):
        self.Dialog = AddTaskDialog()
```

```
        # Connects 'Save' button to save the user's task to Agenda.
        self.Dialog.button_box_new_task.accepted.disconnect()
        self.Dialog.button_box_new_task.accepted.connect(self.save_task)

        # Populates the combo box with subject options.
        with open("subject_list.txt", "r") as data_file:
            subject_list = data_file.readlines()
            for line in subject_list:
                self.Dialog.comb_box_subject.addItem(line.strip("\n"))

        self.Dialog.open()

    # Saves new tasks to the agenda and JSON file.
    def save_task(self):
        # Gets user inputs for subject and due date, and assigns task as incomplete.
        new_subject = self.Dialog.comb_box_subject.currentText()
        new_due_date = self.Dialog.calendar_due_date.selectedDate().toString("dd-MM-yyyy")
        new_completed = ("No")

        # Validates task title input and adds task if validation passed.
        new_task_title = self.Dialog.line_edit_task_title.text()
        if len(new_task_title) <= 30 and len((new_task_title).strip(" ")) > 0:
            # Adds new task as a key with task details as values.
            task_list[new_task_title] = {
                "subject": new_subject,
                "due_date": new_due_date,
                "completed": new_completed}
            # Updates the JSON file with the updated task list dictionary.
            with open('task_list.json', 'w') as outfile:
                json.dump(task_list, outfile, ensure_ascii=False, indent=4)
            # Closes the dialog and updates the task list.
            self.Dialog.close()
            self.update_list()
        # Rejects input if no task title entered.
        elif len((new_task_title).strip(" ")) == 0:
            self.Dialog.lbl_instruction.setText(
                "You have not entered a task title. Please try again.")
        # Rejects input if task title exceeds 30 characters.
        else:
            self.Dialog.lbl_instruction.setText(
                "Your task title exceeds 30 characters. Please try again.")


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    mwindow_agenda = AgendaWindow()
    mwindow_agenda.show()
    sys.exit(app.exec_())
```

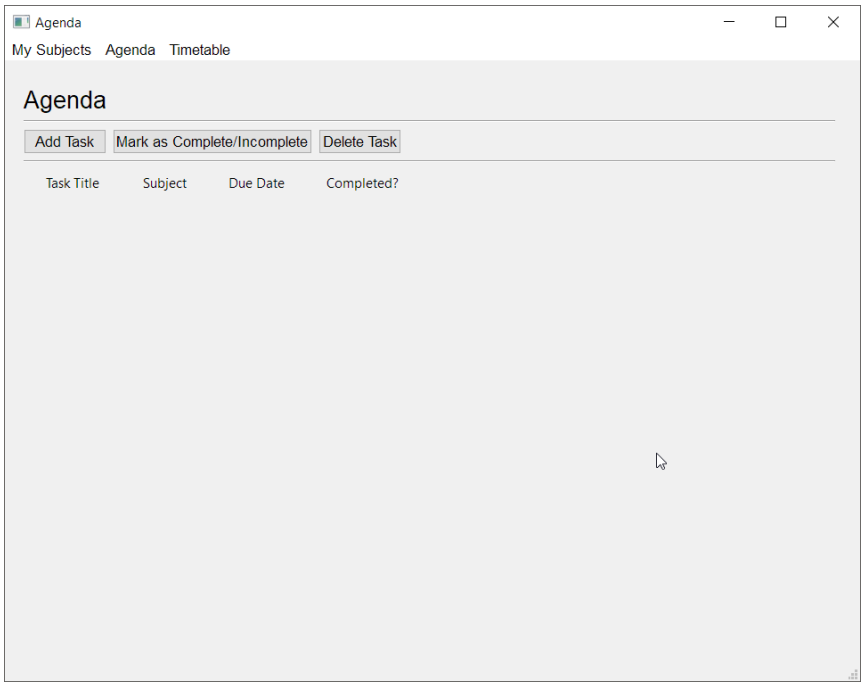
Testing: Iteration #1 – List of Tasks

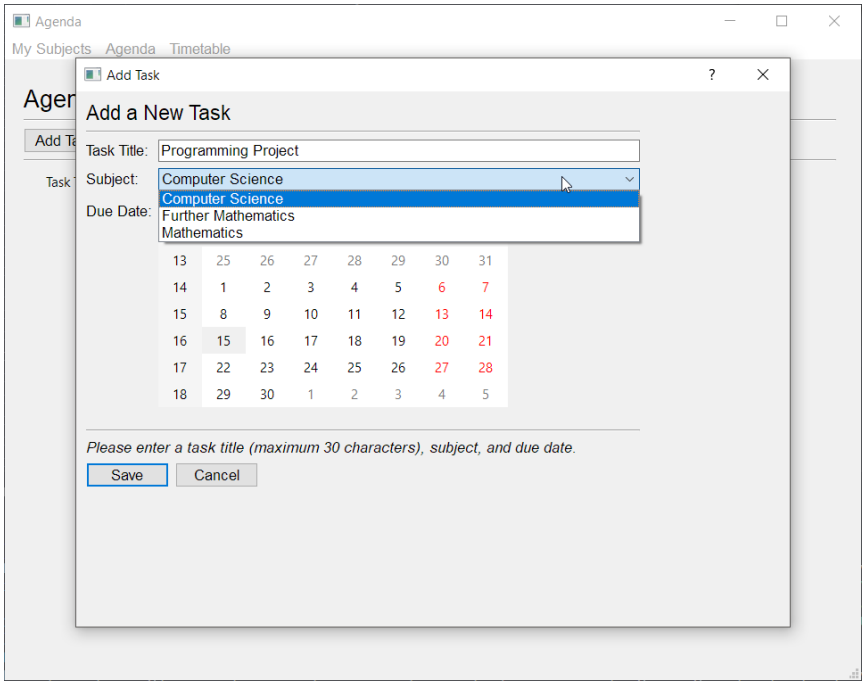
After I made these changes, I started the formal testing process for this development iteration, as there were a lot of changes to check thoroughly. I started by testing the inputs, as shown below by the table and screenshot examples.

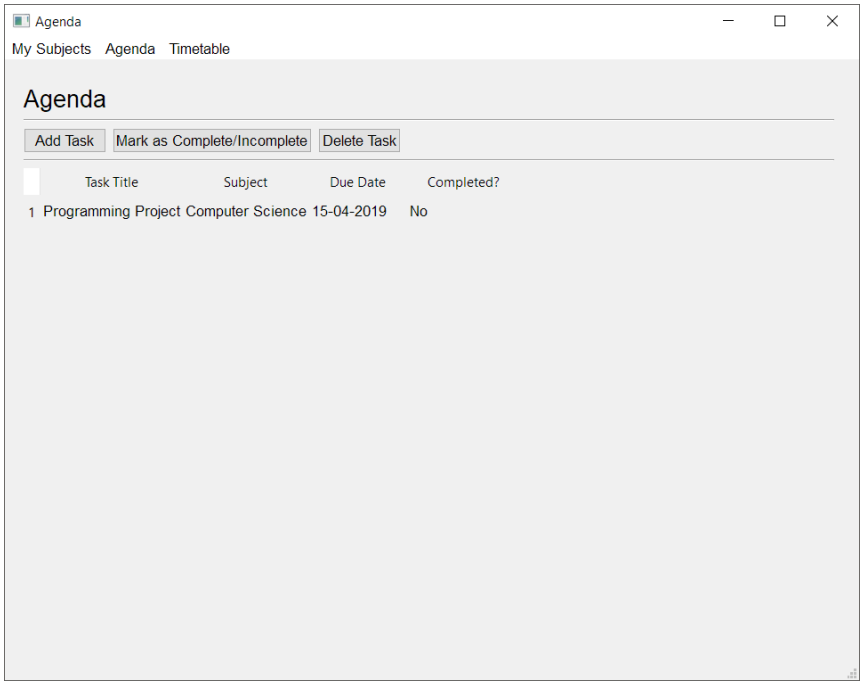
Input	Nature of Data	Data Validation	Is Data Accepted?	Output
Typed 'Programming Project' for the task title, selected 'Computer Science' from the dropdown menu as the subject, and the 15th April 2019 as the due date, then saved the task by pressing the 'Save' button.	Normal	Presence check, length check, button pressed	Yes	The task was added to the agenda, with the same task title and subject as input, the due date displayed in DD-MM-YYYY format, and marked as incomplete.
Typed 'Integration by Substitution' for the task title, selected 'Mathematics' from	Extreme	Presence check, length check, button pressed	Yes	The task was added to the agenda, with the same task title and subject as input, the due date

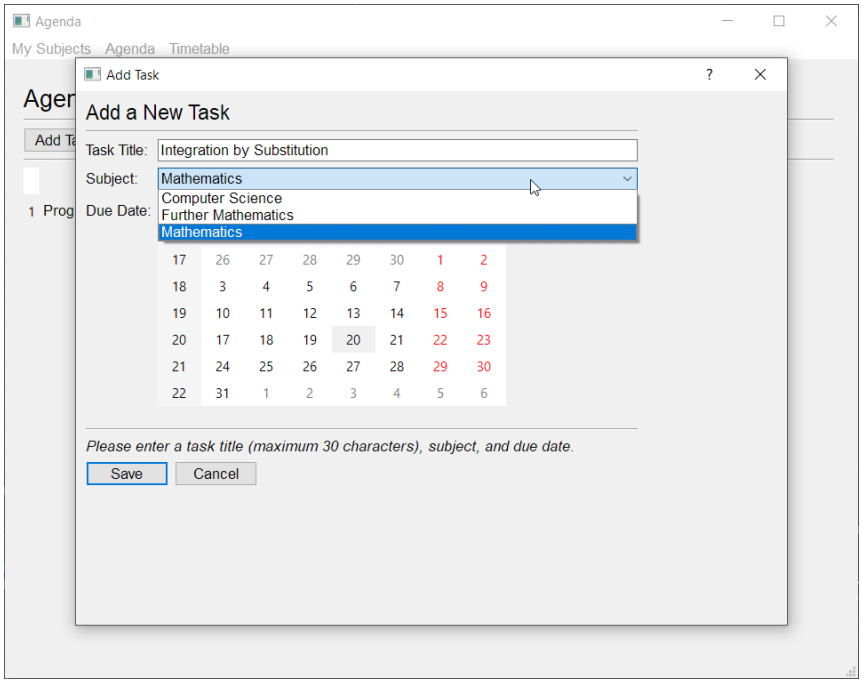
the dropdown menu as the subject, and the 20th May 2021 as the due date, then saved the task by pressing the 'Save' button.				displayed in DD-MM-YYYY format, and marked as incomplete.
Typed ' ' [three spaces] for the task title, 'Further Mathematics' from the dropdown menu as the subject, and 31st May 2019 as the due date, then saved the task by pressing the 'Save' button.	Erroneous	Presence check, length check, button pressed	Yes	The task was not added to the agenda, and the instruction label in the dialog changed to specify that they had not entered a task title, and to try again.
Typed nothing for the task title, and left the subject and due date as default, then saved the task by pressing the 'Save' button.	Null	Presence check, length check, button pressed	Yes	The task was not added to the agenda, and the instruction label in the dialog changed to specify that they had not entered a task title, and to try again.
Typed 'Further Differentiation and Integration' for the task title, 'Mathematics' from the dropdown menu as the subject, and 10th May 2019 as the due date, then saved the task by pressing the 'Save' button.	Erroneous	Presence check, length check, button pressed	Yes	The task was not added to the agenda, and the instruction label in the dialog changed to specify that their task title exceeded 30 characters, and to try again.
Typed 'Practice Paper' for the task title, selected 'Mathematics' from the dropdown menu as the subject, and the 18th April 2019 as the due date, then saved the task by pressing the 'Save' button.	Normal	Presence check, length check, button pressed	Yes	The task was added to the agenda, with the same task title and subject as input, the due date displayed in DD-MM-YYYY format, and marked as incomplete. However, the task list is not displayed in chronological due date order; it was placed in the third

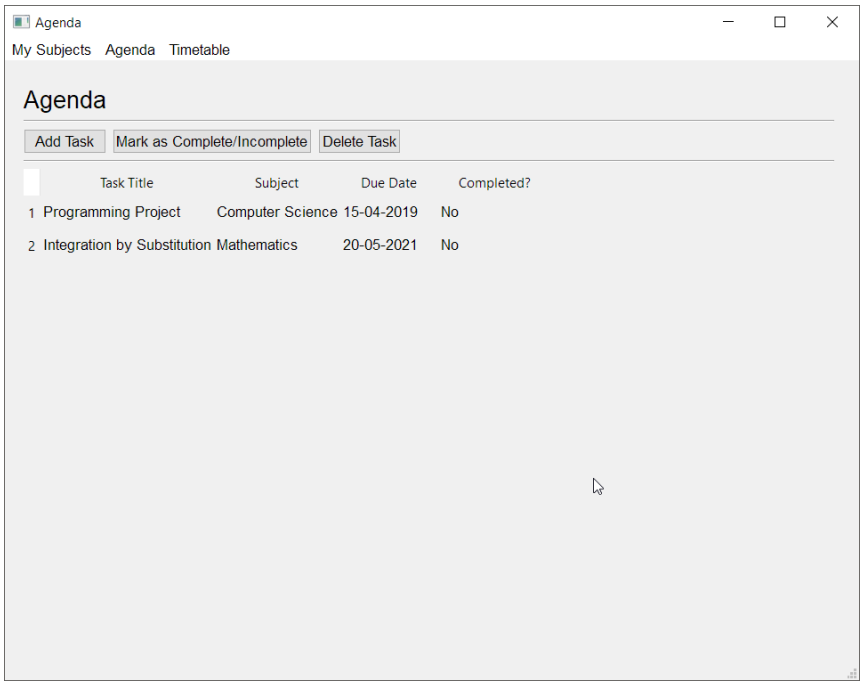
				position rather than the second position.
Typed ‘Practice Paper’ for the task title, selected ‘Further Mathematics’ from the dropdown menu as the subject, and the 19th April 2019 as the due date, then saved the task by pressing the ‘Save’ button.	Normal	Presence check, length check, button pressed	Yes	The previously existing task with task title ‘Practice Paper’ was overwritten with the new subject (‘Further Mathematics’) and due date (’19-04-2019’). The task list is not displayed in chronological due date order; it was placed in the third position rather than the second position.

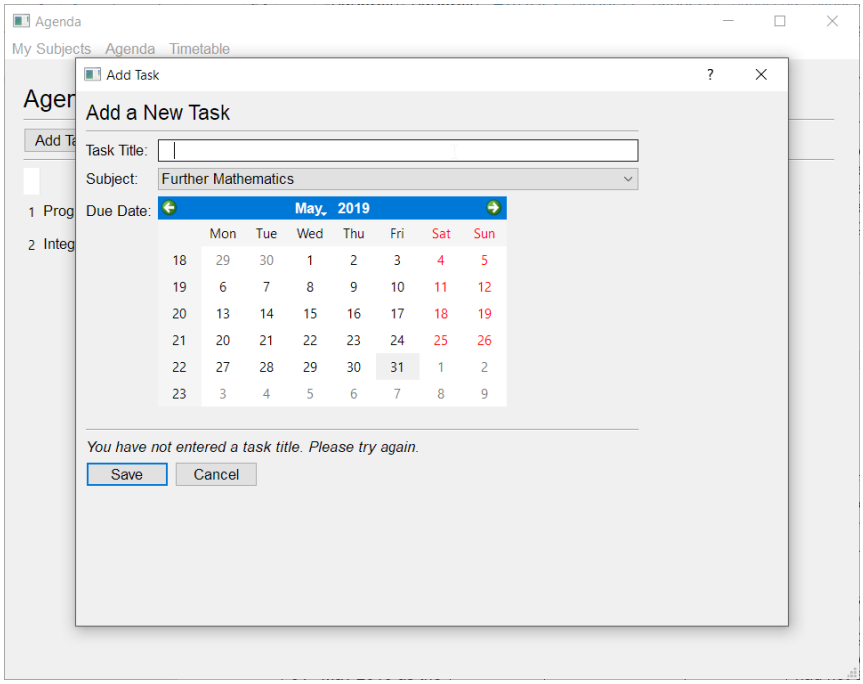


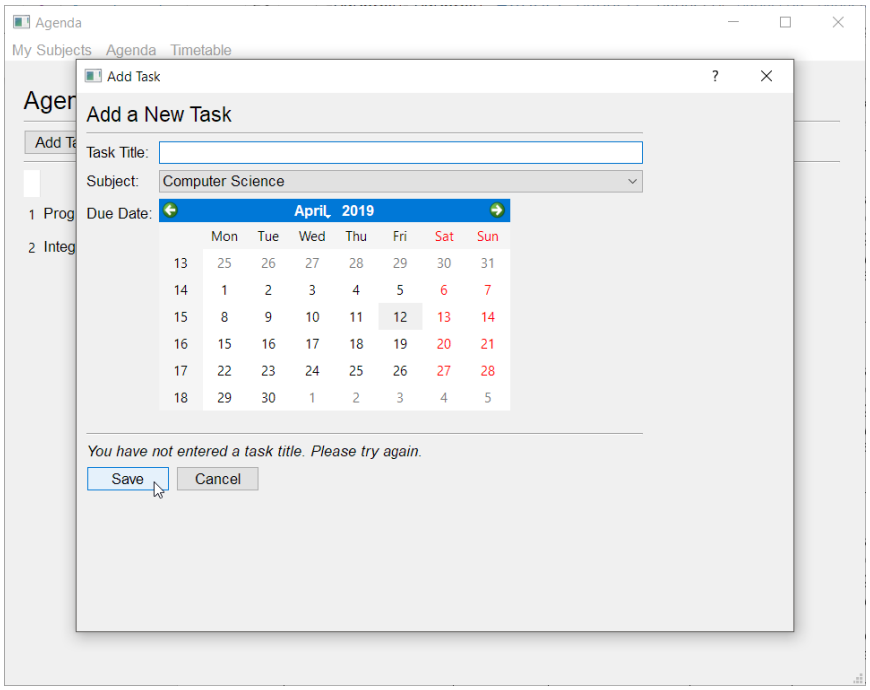

Agenda
My Subjects Agenda Timetable
Add Task
Add a New Task
Task Title: Programming Project
Subject: Computer Science
Computer Science
Further Mathematics
Mathematics
Due Date:
Please enter a task title (maximum 30 characters), subject, and due date.
Save
Cancel

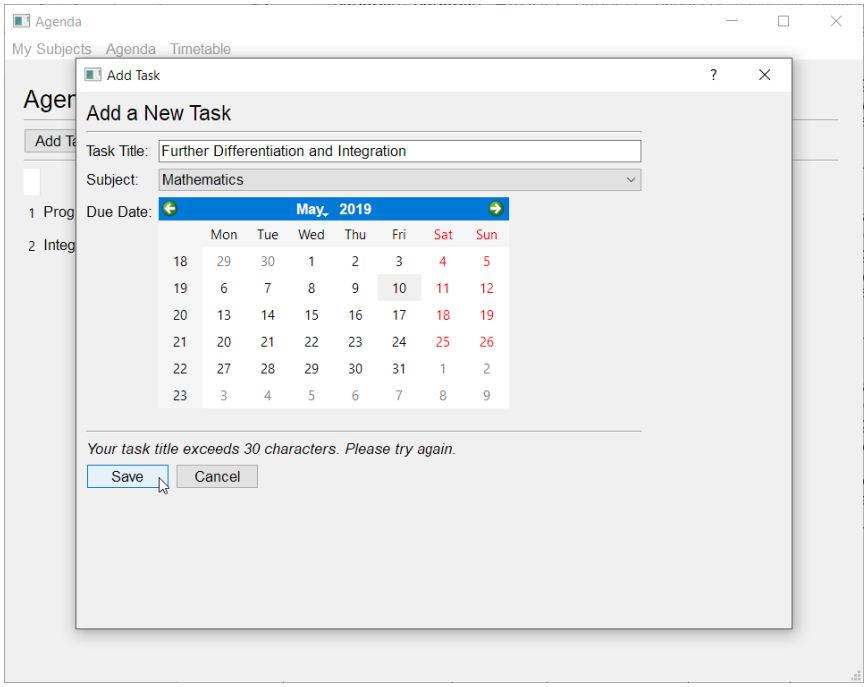

Agenda
My Subjects Agenda Timetable
Agenda
Add Task
Mark as Complete/Incomplete
Delete Task
Task Title Subject Due Date Completed?
1 Programming Project Computer Science 15-04-2019 No

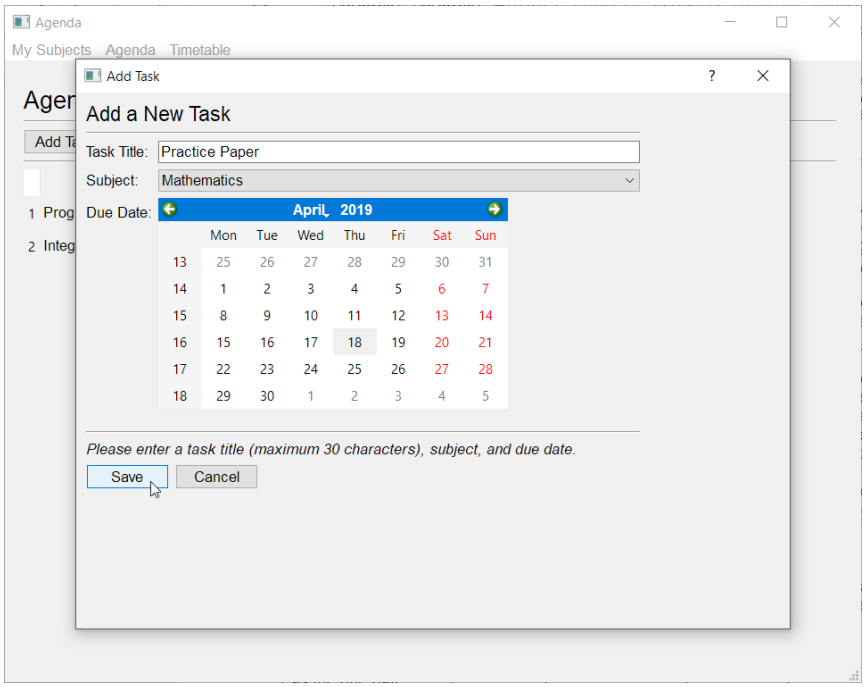

Agenda
My Subjects Agenda Timetable
Add Task
Add a New Task
Task Title: Integration by Substitution
Subject: Mathematics
Computer Science
Further Mathematics
Mathematics
Due Date:
Please enter a task title (maximum 30 characters), subject, and due date.
Save
Cancel

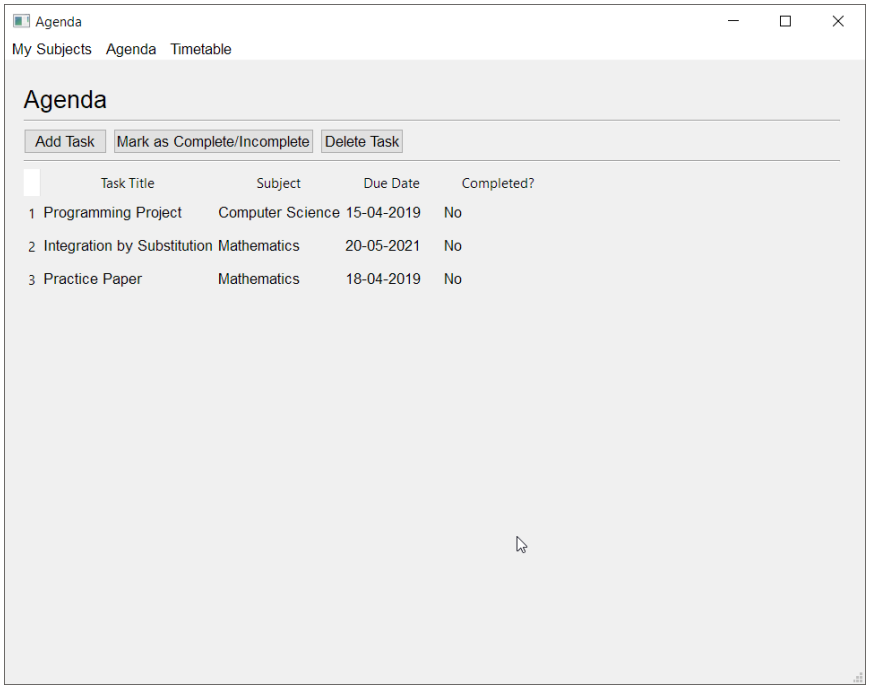

Agenda
My Subjects Agenda Timetable
Agenda
Add Task
Mark as Complete/Incomplete
Delete Task
Task Title Subject Due Date Completed?
1 Programming Project Computer Science 15-04-2019 No
2 Integration by Substitution Mathematics 20-05-2021 No

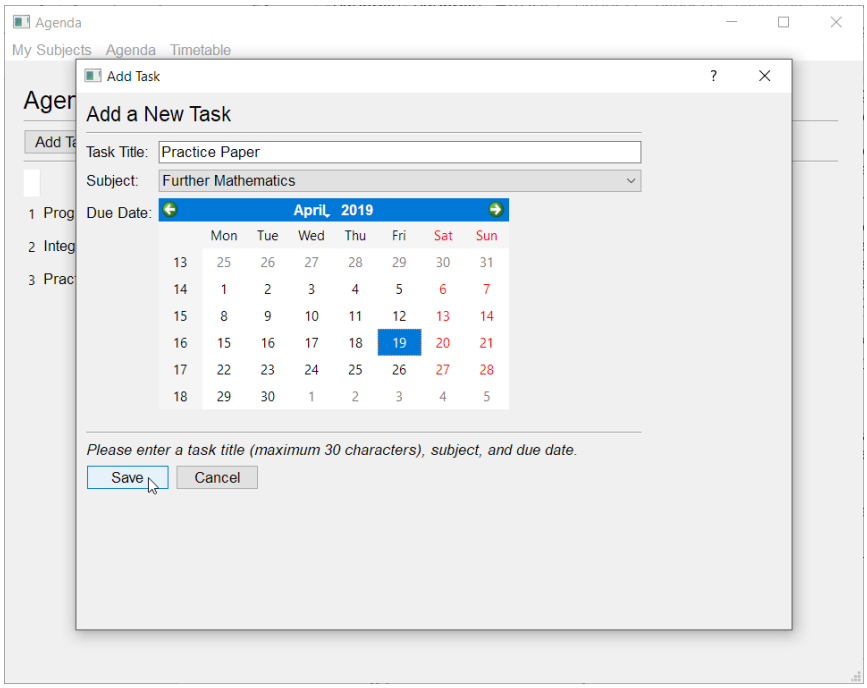

Agenda
My Subjects Agenda Timetable
Add Task
Add a New Task
Task Title:
Subject: Further Mathematics
Due Date: May 2019
Mon Tue Wed Thu Fri Sat Sun
18 29 30 1 2 3 4 5
19 6 7 8 9 10 11 12
20 13 14 15 16 17 18 19
21 20 21 22 23 24 25 26
22 27 28 29 30 31 1 2
23 3 4 5 6 7 8 9
You have not entered a task title. Please try again.
Save
Cancel

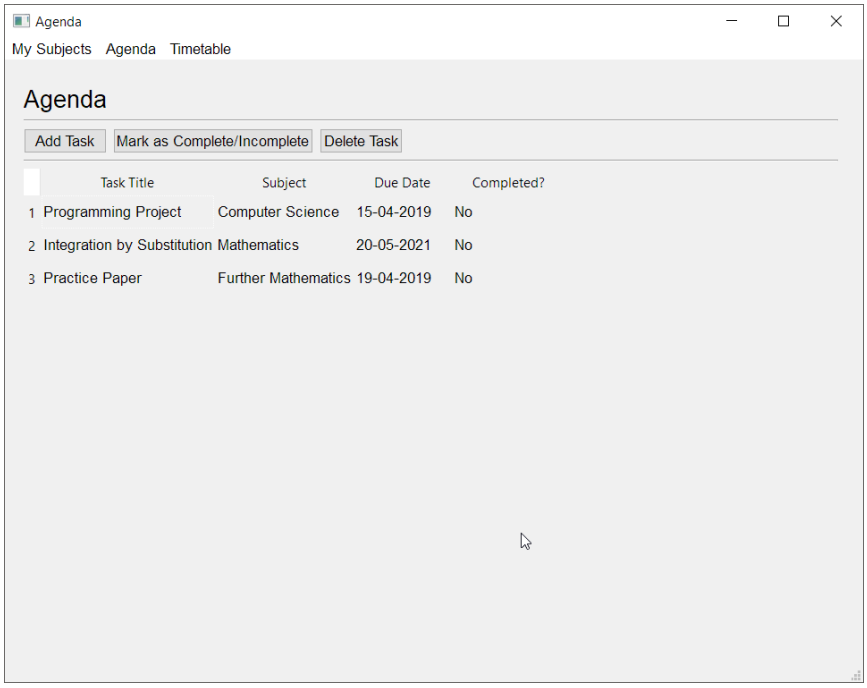

Agenda
My Subjects Agenda Timetable
Add Task
Add a New Task
Task Title:
Subject: Computer Science
Due Date: April 2019
Mon Tue Wed Thu Fri Sat Sun
13 25 26 27 28 29 30 31
14 1 2 3 4 5 6 7
15 8 9 10 11 12 13 14
16 15 16 17 18 19 20 21
17 22 23 24 25 26 27 28
18 29 30 1 2 3 4 5
You have not entered a task title. Please try again.
Save
Cancel


Agenda
My Subjects Agenda Timetable
Add Task
Add a New Task
Task Title: Further Differentiation and Integration
Subject: Mathematics
Due Date: May 2019
Mon Tue Wed Thu Fri Sat Sun
18 29 30 1 2 3 4 5
19 6 7 8 9 10 11 12
20 13 14 15 16 17 18 19
21 20 21 22 23 24 25 26
22 27 28 29 30 31 1 2
23 3 4 5 6 7 8 9
Your task title exceeds 30 characters. Please try again.
Save
Cancel


Agenda
My Subjects Agenda Timetable
Add Task
Add a New Task
Task Title: Practice Paper
Subject: Mathematics
Due Date: April 2019
Mon Tue Wed Thu Fri Sat Sun
13 25 26 27 28 29 30 31
14 1 2 3 4 5 6 7
15 8 9 10 11 12 13 14
16 15 16 17 18 19 20 21
17 22 23 24 25 26 27 28
18 29 30 1 2 3 4 5
Please enter a task title (maximum 30 characters), subject, and due date.
Save
Cancel


Agenda
My Subjects Agenda Timetable
Agenda
Add Task
Mark as Complete/Incomplete
Delete Task
Task Title Subject Due Date Completed?
1 Programming Project Computer Science 15-04-2019 No
2 Integration by Substitution Mathematics 20-05-2021 No
3 Practice Paper Mathematics 18-04-2019 No


Agenda
My Subjects Agenda Timetable
Add Task
Add a New Task
Task Title: Practice Paper
Subject: Further Mathematics
Due Date:
April, 2019
Mon Tue Wed Thu Fri Sat Sun
Please enter a task title (maximum 30 characters), subject, and due date.
Save
Cancel



During the testing process for the inputs, I discovered a bug when entering a new task with the same task title as an existing task. In this case, rather than adding the new task to the agenda and keeping the existing tasks, the program overwrote the existing task with the same task title, and updated the subject and due date to the new task details.

When I discovered this bug, I looked over the code, and found that it was most likely caused by using the task title as the key. This was bad coding practice in the first place, as the key is meant to be unique, which the task title will not be in many cases, such as in my example where two practice papers were set for different subjects and with different due dates.

The fix for this bug will be explored further in the next development iteration alongside other new features.

Apart from this bug and the lack of sorting, the beta tests for inputs were successful, so I moved onto the general testing for this development iteration.

Test	**Expected Result**	**Outcome**	**Further Actions**
Is the user able to add a new task to the agenda through the appropriate widgets in a dialog window?	The user should be able to add a new task to the agenda using a dialog window which has a line edit input for the task title, dropdown menu for the subject, and calendar GUI for the due date.	The user is able to add a new task to the agenda using a dialog window which has a line edit input for the task title, dropdown menu for the subject, and calendar GUI for the due date.	N/A.
Are user inputs when adding a task validated using presence and length checks?	The user should have their line edit input for the task title validated using a presence check and a length check for 30 characters or less.	The user has their line edit input for the task title validated using a presence check and a length check for 30 characters or less.	Use a different key or a different system of storing the user's tasks to prevent issues when there are

	(Subject and due date are already selected by default, so there are no cases when a presence check would be failed.) If validation is passed, the task should be added to the agenda.	However, the task is not always added to the agenda. When a task title which was already used is entered as a new subject, the existing task with that task title is overwritten.	duplicate task titles.
Are tasks on the agenda displayed in sorted chronological order by due date?	The tasks should be displayed in sorted chronological order by due date, meaning that tasks with earlier due dates should be positioned above tasks with later due dates.	The tasks were not displayed in sorted chronological order by due date. Instead, they were displayed in the order of being added.	Implement or create a sorting algorithm to display the tasks in sorted chronological order by due date.

The general beta testing has demonstrated that although there is now basic functionality in Agenda, there are still fixes and improvements to make. These will be addressed in the next development iteration.

Development: Iteration #2 – Sorting, Marking, and Deleting

The primary focuses for this development iteration will be implementing fixes for the duplicate task title issue I had in the previous development iteration (as shown in the testing phase), and developing features for sorting, marking, and deleting tasks in Agenda.

To start with, I changed *.toString("dd-MM-yyyy")* to *.toString("dd/MM/yyyy")*, as slashes are more commonly used than dashes to separate the day, month, and year within my target demographic of the United Kingdom.

```
93          new_due_date = self.Dialog.calendar_due_date.selectedDate().toString("dd/MM/yyyy")
```

Next, I moved onto solving the issue of adding tasks with pre-existing task titles. This stemmed from the use of the task title as a key, which was not good programming practice, as it is likely for a task title to be used in more than one instance. To solve this, I changed my system to use a list of dictionaries, as it had previously only used a single dictionary.

In the *save_task()* method, I changed the part where the program saves the new task as a dictionary, which had been using the task title as the key, as the subject, due date, and completion as the values. Instead, I defined the variable *task* to a dictionary which contains all the task details; the task title, subject, due date, and completion. This task is then appended to the task list using *.append(dict(task))*.

```
96          if len(new_task_title) <= 30 and len((new_task_title).strip(" ")) > 0:
97              # Appends new task to the task list as a dictionary.
98              task = {
99                  "task_title": new_task_title,
100                 "subject": new_subject,
101                 "due_date": new_due_date,
102                 "completed": new_completed}
103             task_list.append(dict(task))
```

The changes in that method meant that I had to apply a small change at the start of the program to define *task_list* as a list rather than a dictionary.

```
10  # Identifies the task list as a list variable.
11  task_list = []
```

Using a list of dictionaries rather than a single dictionary also changed the way that I needed to update the list, so I had to update the method *update_list()*. I used two *for* loops (one nested into the other), but the nested *for* loop is responsible for adding all items to the *QTableWidget* this time, as the task title is now also a value alongside the subject, due date, and completion. This means that *current_column* starts at 0, because the nested *for* loop did not previously have to add the far left column values.

The first *for* loop iterates between each dictionary in the list, which represent the rows in the *QTableWidget*. This allows the *current_column* variable to be reset to 0 after the values are iterated, so that the next row can be added by starting on the far left column. It also adds 1 to the *current_row* variable to achieve this.

In the nested *for* loop, the values in each dictionary are iterated through and added to the *current_row* and *current_column* in the table. At the end of this nested loop, *current_column* is incremented by 1, so that the next iteration will add the item to the next column until all values in the dictionary have been added to the *QTableWidget*.

```
57          # Adds tasks to the table widget.
58          current_row = 0
59          for task in task_list:
60              current_column = 0
61
62              # Adds task details to the agenda.
63              for task_details in task.values():
64                  self.table_widget_task_list.setItem(
65                      current_row, current_column,
66                      QtWidgets.QTableWidgetItem(task_details))
67                  current_column += 1
68
69              current_row += 1
```

Changes in the format the due date is stored and how the data is stored as a list rather than a dictionary meant that the existing JSON file was incompatible. As the data saved in that file was only used for test purposes, I wiped the file to ensure future compatibility with the code in my program.

Once I updated my program to store the data in a list of dictionaries, I looked into methods of sorting the agenda by due date, as the default sorting only sorts the data alphanumerically.

At first, I was considering overriding the built-in sort methods for the *QTableWidget*, as this would be the most obvious idea. However, I considered other alternatives, and found that it would be easier to sort the list of dictionaries by the due date values, then populate the *QTableWidget* using this sorted list of dictionaries.

I found online (https://stackoverflow.com/questions/23158766/sort-a-python-date-string-list) that I could use the *sort()* function to sort an existing list. In this function, it uses a key to perform the sort logic. For the key, *lambda* acts as a nameless function, and the date string

is converted into *datetime* so that the function can compare dates in a logical manner for the sort to be performed.

Using this knowledge, I created a method, *sort_task_list()*, to sort the task list and then update the JSON file with this sorted list. It takes the value from 'due_date' from each dictionary, and this is compared between the dictionaries to decide the order of the list. Then, it calls a new function, *save_task_list()*, to save the sorted list to the JSON file, *task_list.json*.

```
132     # Sorts the task list by due date.
133     def sort_task_list(self):
134         task_list.sort(key=lambda task: datetime.strptime(task["due_date"], '%d/%m/%Y'))
135         self.save_task_list()
```

I created *save_task_list()* to store the same code which I used in the previous stage to overwrite *task_list.json* with the list stored in the variable *task_list*. This is because it helps with code reusability, as this code is already being used in *sort_task_list()* and *save_task()* (which I have also changed to call this method rather than execute the code directly).

```
126     # Updates the JSON file with the current task list.
127     def save_task_list(self):
128         with open('task_list.json', 'w') as outfile:
129             json.dump(task_list, outfile, ensure_ascii=False, indent=4)
```

```
92      # Saves new tasks to the agenda and JSON file.
93      def save_task(self):
94          # Gets user inputs for subject and due date, and assigns task as incomplete.
95          new_subject = self.Dialog.comb_box_subject.currentText()
96          new_due_date = self.Dialog.calendar_due_date.selectedDate().toString("dd/MM/yyyy")
97          new_completed = ("No")
98
99          # Validates task title input and adds task if validation passed.
100         new_task_title = self.Dialog.line_edit_task_title.text()
101         if len(new_task_title) <= 30 and len((new_task_title).strip(" ")) > 0:
102             # Appends new task to the task list as a dictionary.
103             task = {
104                 "task_title": new_task_title,
105                 "subject": new_subject,
106                 "due_date": new_due_date,
107                 "completed": new_completed}
108             task_list.append(dict(task))
109
110             self.save_task_list()
```

These cumulative changes resulted in the following code for the program after updating it to sort the task list by due date.

```python
import json
import sys
from datetime import datetime

from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QDialog, QMainWindow

from add_task_setup import Ui_dialog_new_task
from agenda_setup import Ui_mwindow_agenda

# Identifies the task list as a list variable.
task_list = []

# Reads existing JSON file for list of tasks.
with open('task_list.json', 'r') as outfile:
    try:
        task_list = json.load(outfile)
        json.dump(task_list, sys.stdout, ensure_ascii=False, indent=4)
    except ValueError:
        print("Empty JSON file.")

# Sets up the Add Task dialog.


class AddTaskDialog(QDialog, Ui_dialog_new_task):
    def __init__(self):
        super().__init__()
        self.setupUi(self)

# Sets up the Agenda main window.


class AgendaWindow(QMainWindow, Ui_mwindow_agenda):
    def __init__(self):
        super().__init__()
        self.setupUi(self)

        # Sets the list of tasks to have a transparent background.
        self.setStyleSheet("""QTableWidget {background-color: transparent;}
            QHeaderView::section {background-color: transparent;}
            QHeaderView {background-color: transparent;}""")

        # Connects 'Add Task' button to the Add Task dialog.
        self.btn_add_task.clicked.connect(self.open_dialog_add_task)

        # Populates list on start-up.
        self.update_list()

    # Reads the existing JSON file and populates the table widget.
    def update_list(self):
        # Sets number of rows, column size, and row size.
        self.table_widget_task_list.setRowCount(len(task_list))
        self.table_widget_task_list.horizontalHeader().setSectionResizeMode(
            QtWidgets.QHeaderView.ResizeToContents)
        self.table_widget_task_list.verticalHeader().setSectionResizeMode(
            QtWidgets.QHeaderView.Fixed)

        # Sorts the task list by due date.
        self.sort_task_list()

        # Adds tasks to the table widget.
        current_row = 0
        for task in task_list:
            current_column = 0

            # Adds task details to the agenda.
            for task_details in task.values():
                self.table_widget_task_list.setItem(
                    current_row, current_column,
                    QtWidgets.QTableWidgetItem(task_details))
                current_column += 1

            current_row += 1

    # Opens the dialog for the user to add a task.
    def open_dialog_add_task(self):
```

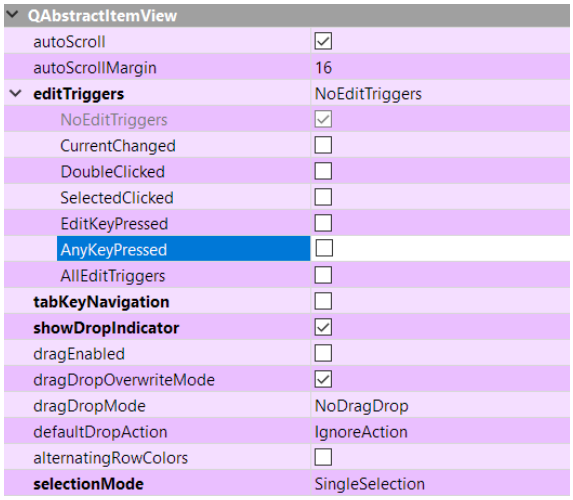
```
        self.Dialog = AddTaskDialog()

        # Connects 'Save' button to save the user's task to Agenda.
        self.Dialog.button_box_new_task.accepted.disconnect()
        self.Dialog.button_box_new_task.accepted.connect(self.save_task)

        # Populates the combo box with subject options.
        with open("subject_list.txt", "r") as data_file:
            subject_list = data_file.readlines()
            for line in subject_list:
                self.Dialog.comb_box_subject.addItem(line.strip("\n"))

        self.Dialog.open()

    # Saves new tasks to the agenda and JSON file.
    def save_task(self):
        # Gets user inputs for subject and due date, and assigns task as incomplete.
        new_subject = self.Dialog.comb_box_subject.currentText()
        new_due_date = self.Dialog.calendar_due_date.selectedDate().toString("dd/MM/yyyy")
        new_completed = ("No")

        # Validates task title input and adds task if validation passed.
        new_task_title = self.Dialog.line_edit_task_title.text()
        if len(new_task_title) <= 30 and len((new_task_title).strip(" ")) > 0:
            # Appends new task to the task list as a dictionary.
            task = {
                "task_title": new_task_title,
                "subject": new_subject,
                "due_date": new_due_date,
                "completed": new_completed}
            task_list.append(dict(task))

            self.save_task_list()

            # Closes the dialog and updates the task list.
            self.Dialog.close()
            self.update_list()
```

```
        # Rejects input if no task title entered.
        elif len((new_task_title).strip(" ")) == 0:
            self.Dialog.lbl_instruction.setText(
                "You have not entered a task title. Please try again.")

        # Rejects input if task title exceeds 30 characters.
        else:
            self.Dialog.lbl_instruction.setText(
                "Your task title exceeds 30 characters. Please try again.")

    # Updates the JSON file with the current task list.
    def save_task_list(self):
        with open('task_list.json', 'w') as outfile:
            json.dump(task_list, outfile, ensure_ascii=False, indent=4)

    # Sorts the task list by due date.
    def sort_task_list(self):
        task_list.sort(key=lambda task: datetime.strptime(
            task["due_date"], '%d/%m/%Y'))
        self.save_task_list()


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    mwindow_agenda = AgendaWindow()
    mwindow_agenda.show()
    sys.exit(app.exec_())
```

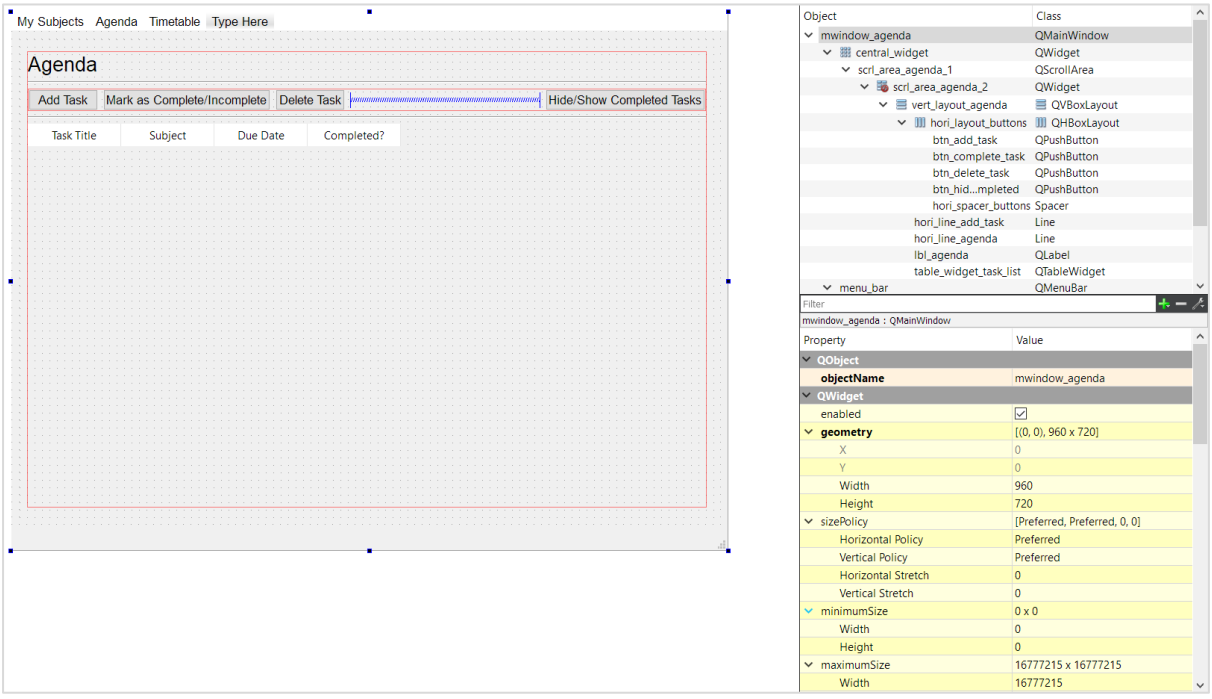
To prevent the user from sorting the agenda by clicking on the columns, I disabled *sortingEnabled*, and I also disabled *cornerButtonEnabled* because the corner button is not necessary for this use case. Both of these changes were made in Qt Creator, and I generated the updated code using pyuic5 to *agenda_setup.py*.

QTableView	
showGrid	☐
gridStyle	SolidLine
sortingEnabled	☐
wordWrap	☑
cornerButtonEnabled	☐

I also found out during some informal testing that it was possible to edit the items on the *QTableWidget* by performing double clicks and other actions. This was not intended, so I disabled the triggers for editing the items.



In addition, I added a button for hiding/showing completed tasks. I placed this on the far right side of the button row, as it is a viewing option as opposed to a task option. This functionality will be added later in this stage when the user is able to mark tasks as complete/incomplete.



I converted the user interface into Python code in *agenda_setup.py* using *pyuic5.*

```python
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'agenda.ui'
#
# Created by: PyQt5 UI code generator 5.12.1
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_mwindow_agenda(object):
    def setupUi(self, mwindow_agenda):
        mwindow_agenda.setObjectName("mwindow_agenda")
        mwindow_agenda.resize(960, 720)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        mwindow_agenda.setFont(font)
        self.central_widget = QtWidgets.QWidget(mwindow_agenda)
        self.central_widget.setObjectName("central_widget")
        self.gridLayout = QtWidgets.QGridLayout(self.central_widget)
        self.gridLayout.setObjectName("gridLayout")
        self.scrl_area_agenda_1 = QtWidgets.QScrollArea(self.central_widget)
        self.scrl_area_agenda_1.setFrameShape(QtWidgets.QFrame.NoFrame)
        self.scrl_area_agenda_1.setWidgetResizable(True)
        self.scrl_area_agenda_1.setObjectName("scrl_area_agenda_1")
        self.scrl_area_agenda_2 = QtWidgets.QWidget()
        self.scrl_area_agenda_2.setGeometry(QtCore.QRect(0, 0, 938, 648))
        self.scrl_area_agenda_2.setObjectName("scrl_area_agenda_2")
        self.layoutWidget = QtWidgets.QWidget(self.scrl_area_agenda_2)
        self.layoutWidget.setGeometry(QtCore.QRect(10, 16, 911, 611))
        self.layoutWidget.setObjectName("layoutWidget")
        self.vert_layout_agenda = QtWidgets.QVBoxLayout(self.layoutWidget)
        self.vert_layout_agenda.setContentsMargins(0, 0, 0, 0)
        self.vert_layout_agenda.setObjectName("vert_layout_agenda")
        self.lbl_agenda = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setPointSize(16)
        self.lbl_agenda.setFont(font)
        self.lbl_agenda.setObjectName("lbl_agenda")
        self.vert_layout_agenda.addWidget(self.lbl_agenda)
        self.hori_line_agenda = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_agenda.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_agenda.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_agenda.setObjectName("hori_line_agenda")
        self.vert_layout_agenda.addWidget(self.hori_line_agenda)
        self.hori_layout_buttons = QtWidgets.QHBoxLayout()
        self.hori_layout_buttons.setObjectName("hori_layout_buttons")
        self.btn_add_task = QtWidgets.QPushButton(self.layoutWidget)
        self.btn_add_task.setObjectName("btn_add_task")
        self.hori_layout_buttons.addWidget(
            self.btn_add_task, 0, QtCore.Qt.AlignLeft)
        self.btn_complete_task = QtWidgets.QPushButton(self.layoutWidget)
        self.btn_complete_task.setObjectName("btn_complete_task")
        self.hori_layout_buttons.addWidget(self.btn_complete_task)
        self.btn_delete_task = QtWidgets.QPushButton(self.layoutWidget)
        self.btn_delete_task.setObjectName("btn_delete_task")
        self.hori_layout_buttons.addWidget(self.btn_delete_task)
        spacerItem = QtWidgets.QSpacerItem(
            40, 20, QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Minimum)
        self.hori_layout_buttons.addItem(spacerItem)
        self.btn_hide_completed = QtWidgets.QPushButton(self.layoutWidget)
        self.btn_hide_completed.setObjectName("btn_hide_completed")
        self.hori_layout_buttons.addWidget(self.btn_hide_completed)
        self.vert_layout_agenda.addLayout(self.hori_layout_buttons)
        self.hori_line_add_task = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_add_task.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_add_task.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_add_task.setObjectName("hori_line_add_task")
        self.vert_layout_agenda.addWidget(self.hori_line_add_task)
        self.table_widget_task_list = QtWidgets.QTableWidget(self.layoutWidget)
        palette = QtGui.QPalette()
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Button, brush)
```

```
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.NoBrush)
        palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Base, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Window, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Button, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.NoBrush)
        palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Base, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Window, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Button, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.NoBrush)
        palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Base, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Window, brush)
        self.table_widget_task_list.setPalette(palette)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.table_widget_task_list.setFont(font)
        self.table_widget_task_list.setAutoFillBackground(False)
        self.table_widget_task_list.setStyleSheet(
            "background-color: transparent")
        self.table_widget_task_list.setFrameShape(QtWidgets.QFrame.NoFrame)
        self.table_widget_task_list.setLineWidth(0)
        self.table_widget_task_list.setSizeAdjustPolicy(
            QtWidgets.QAbstractScrollArea.AdjustToContents)
        self.table_widget_task_list.setEditTriggers(
            QtWidgets.QAbstractItemView.NoEditTriggers)
```

```
        self.table_widget_task_list.setTabKeyNavigation(False)
        self.table_widget_task_list.setProperty("showDropIndicator", True)
        self.table_widget_task_list.setSelectionMode(
            QtWidgets.QAbstractItemView.SingleSelection)
        self.table_widget_task_list.setSelectionBehavior(
            QtWidgets.QAbstractItemView.SelectRows)
        self.table_widget_task_list.setShowGrid(False)
        self.table_widget_task_list.setCornerButtonEnabled(False)
        self.table_widget_task_list.setObjectName("table_widget_task_list")
        self.table_widget_task_list.setColumnCount(4)
        self.table_widget_task_list.setRowCount(0)
        item = QtWidgets.QTableWidgetItem()
        item.setBackground(QtGui.QColor(0, 0, 0, 0))
        self.table_widget_task_list.setHorizontalHeaderItem(0, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_task_list.setHorizontalHeaderItem(1, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_task_list.setHorizontalHeaderItem(2, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_task_list.setHorizontalHeaderItem(3, item)
        self.table_widget_task_list.horizontalHeader().setVisible(True)
        self.table_widget_task_list.horizontalHeader().setHighlightSections(True)
        self.table_widget_task_list.horizontalHeader().setStretchLastSection(False)
        self.table_widget_task_list.verticalHeader().setVisible(True)
        self.vert_layout_agenda.addWidget(self.table_widget_task_list)
        self.scrl_area_agenda_1.setWidget(self.scrl_area_agenda_2)
        self.gridLayout.addWidget(self.scrl_area_agenda_1, 0, 0, 1, 1)
        mwindow_agenda.setCentralWidget(self.central_widget)
        self.menu_bar = QtWidgets.QMenuBar(mwindow_agenda)
        self.menu_bar.setGeometry(QtCore.QRect(0, 0, 960, 25))
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_bar.setFont(font)
        self.menu_bar.setObjectName("menu_bar")
        self.menu_agenda = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
```

```
        font.setPointSize(10)
        self.menu_agenda.setFont(font)
        self.menu_agenda.setObjectName("menu_agenda")
        self.menu_my_subjects = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_my_subjects.setFont(font)
        self.menu_my_subjects.setObjectName("menu_my_subjects")
        self.menu_timetable = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_timetable.setFont(font)
        self.menu_timetable.setObjectName("menu_timetable")
        mwindow_agenda.setMenuBar(self.menu_bar)
        self.status_bar = QtWidgets.QStatusBar(mwindow_agenda)
        self.status_bar.setObjectName("status_bar")
        mwindow_agenda.setStatusBar(self.status_bar)
        self.menu_bar.addAction(self.menu_my_subjects.menuAction())
        self.menu_bar.addAction(self.menu_agenda.menuAction())
        self.menu_bar.addAction(self.menu_timetable.menuAction())

        self.retranslateUi(mwindow_agenda)
        QtCore.QMetaObject.connectSlotsByName(mwindow_agenda)

    def retranslateUi(self, mwindow_agenda):
        _translate = QtCore.QCoreApplication.translate
        mwindow_agenda.setWindowTitle(_translate("mwindow_agenda", "Agenda"))
        self.lbl_agenda.setText(_translate("mwindow_agenda", "Agenda"))
        self.btn_add_task.setText(_translate("mwindow_agenda", "Add Task"))
        self.btn_complete_task.setText(_translate(
            "mwindow_agenda", "Mark as Complete/Incomplete"))
        self.btn_delete_task.setText(
            _translate("mwindow_agenda", "Delete Task"))
        self.btn_hide_completed.setText(_translate(
            "mwindow_agenda", "Hide/Show Completed Tasks"))
        self.table_widget_task_list.setSortingEnabled(False)
```

```
        item = self.table_widget_task_list.horizontalHeaderItem(0)
        item.setText(_translate("mwindow_agenda", "Task Title"))
        item = self.table_widget_task_list.horizontalHeaderItem(1)
        item.setText(_translate("mwindow_agenda", "Subject"))
        item = self.table_widget_task_list.horizontalHeaderItem(2)
        item.setText(_translate("mwindow_agenda", "Due Date"))
        item = self.table_widget_task_list.horizontalHeaderItem(3)
        item.setText(_translate("mwindow_agenda", "Completed?"))
        self.menu_agenda.setTitle(_translate("mwindow_agenda", "Agenda"))
        self.menu_my_subjects.setTitle(
            _translate("mwindow_agenda", "My Subjects"))
        self.menu_timetable.setTitle(_translate("mwindow_agenda", "Timetable"))

```

Next, I worked on functionality for *btn_complete_task*, the toggle button for marking a task as complete or incomplete. I started by connecting clicking on *btn_complete_task* to the method which I will code to mark the selected task as complete/incomplete, *mark_task_complete()*.

```
        # Connects 'Mark as Complete/Incomplete' button to mark selected task.
        self.btn_complete_task.clicked.connect(self.mark_task_complete)
```

In the method *mark_task_complete()*, I store the row which the user selected in the variable *selectedRow*, obtaining this row by using the function *currentRow()*. Then, the item on the third column of the selected row (the ‘Completed?’ column) is changed to ‘Yes’ if the current item (obtained using *.text()*) was ‘No’, and vice versa.

After the completion state has been changed in the table, this is copied over to the list of tasks by changing the value of ‘selected’ in the index, which is navigated to by using the selected row, as each row in the table widget represents a new item in the list. Then, I called the function *save_task_list()* to update the JSON file with the latest list of dictionaries.

```
138     # Marks selected task as complete/incomplete.
139     def mark_task_complete(self):
140         selected_row = self.table_widget_task_list.currentRow()
141         if self.table_widget_task_list.item(selected_row, 3).text() == "No":
142             self.table_widget_task_list.item(selected_row, 3).setText("Yes")
143         else:
144             self.table_widget_task_list.item(selected_row, 3).setText("No")
145 
146         # Updates the task completion state to the list and JSON file.
147         task_list[selected_row]["completed"] = self.table_widget_task_list.item(
148             selected_row, 3).text()
149         self.save_task_list()
```

Marking tasks as complete/incomplete is useless for the user unless they have the option to hide completed tasks, so I decided to develop this feature next. Once again, I started by connecting the button for this to the method which I will be using to hide/show completed tasks, *hide_completed_tasks()*.

```
47          # Connects 'Hide/Show Completed Tasks' button to hide/show tasks.
48          self.btn_hide_completed.clicked.connect(self.hide_completed_tasks)
```

Before creating this method, I defined the variable *hidden_tasks* as *True* by default at the start of the program (and outside all classes), which means that the agenda will hide completed tasks. This is because showing completed tasks by default would result in a cluttered aesthetic for the user once they have used the application over a long time. I defined it as a global variable, as it will be used across multiple functions.

```
14  # Hides completed tasks by default.
15  hidden_tasks = True
```

I also set the text of the task view button to ‘Show Completed Tasks’ when Agenda is initialised, as this will go with the default value of *hidden_tasks* being *True*, meaning clicking the button would show completed tasks.

```
53          # Sets text of task view button to 'Show Completed Tasks'.
54          self.btn_hide_completed.setText("Show Completed Tasks")
```

Then I created the method *hide_completed_tasks()* to use the global variable *hidden_tasks* using *global*.

If this value is *False*, it will change *hidden_tasks* to True, and then change the text of the button to ‘Show Completed Tasks’, as pressing the button again would make *hidden_tasks* be set to *False*.

The vice versa was applied for if *hidden_tasks* is *True*, which I used an *else* statement for. It will change *hidden_tasks* to *False*, then change the text of the button to show ‘Hide Completed Tasks’.

At the end of the method, the other method *update_list()* is called. This method will be updated next to hide completed tasks on the list if *hidden_tasks* is *True*.

```
    # Hides/shows completed tasks.
    def hide_completed_tasks(self):
        # Calls for use of the global variable.
        global hidden_tasks

        # Changes variable value and sets text according to previous value.
        if hidden_tasks is False:
            hidden_tasks = True
            self.btn_hide_completed.setText("Show Completed Tasks")
        else:
            hidden_tasks = False
            self.btn_hide_completed.setText("Hide Completed Tasks")

        self.update_list()
```

To store the list of incomplete tasks, I created a new list variable, *task_list_hidden*, and a new JSON file, *task_list_hidden.json*.

This meant that I had to define the new list variable, *task_list_hidden*, at the start of the program. I was initially going to use *task_list_hidden = [ ]*, but I realised that it was not necessary for me to do this, as there is a block of code which defines *task_list* by reading the list from *task_list.json*. Hence, I removed the definition *task_list = [ ]*, and instead updated the block of code which reads the lists from the files.

Inside the block of code, I added code inside the *try* indentation to open *task_list_hidden.json* and define *task_list_hidden* as the contents of this file. It was placed inside this indentation because the code will not need to be run if *task_list.json* is empty, as *task_list.json* being empty implies that *task_list_hidden.json* is also empty.

If the file is read and found to be empty, the block of code will return a *ValueError*. In this case, the code prints 'Empty JSON file' to the console for debugging purposes, then defines *task_list* and *task_list_hidden* as empty list variables.

```
# Reads existing JSON files for lists of tasks.
with open('task_list.json', 'r') as outfile:
    try:
        task_list = json.load(outfile)
        json.dump(task_list, sys.stdout, ensure_ascii=False, indent=4)

        with open('task_list_hidden.json', 'r') as outfile:
            task_list_hidden = json.load(outfile)
    except ValueError:
        print("Empty JSON file.")
        task_list = []
        task_list_hidden = []
```

In the method *update_list()*, I changed the code so that it reads from the list variable *current_list* instead of *task_list*.

To implement this properly, I added code which makes the contents of *current_list* be *task_list_hidden* if the user has selected the option to hide completed tasks, or *task_list* if the user has chosen to show completed tasks.

```
59      # Populates the agenda with all tasks.
60      def update_list(self):
61          # Displays completed tasks if user selected the option.
62          if hidden_tasks is True:
63              current_list = task_list_hidden
64          else:
65              current_list = task_list
```

This meant that in the first *for* loop in this method, I used *task in current_list* rather than *task in task_list*.

```
77          # Adds tasks to the agenda.
78          current_row = 0
79          for task in current_list:
```

Another change I made to fully implement the hiding/showing of completed tasks into my program was to the method *save_task_list()*.

In this method, I added code to define the contents of *task_list_hidden[:]* as all the tasks in *task_list* which have their 'completed' value as 'No'. The *[:]* part is responsible for updating the variable *task_list_hidden* from its previous contents.

Then, the program saves this list to *task_list_hidden.json*, which ensures that the option to show/hide completed tasks works on initial program start-up, and not just when the user has marked a task as complete/incomplete.

```
146     # Updates the JSON file with the current task list.
147     def save_task_list(self):
148         with open('task_list.json', 'w') as outfile:
149             json.dump(task_list, outfile, ensure_ascii=False, indent=4)
150
151         task_list_hidden[:] = [
152             task for task in task_list if task["completed"] == ("No")]
153         with open('task_list_hidden.json', 'w') as outfile:
154             json.dump(task_list_hidden, outfile, ensure_ascii=False, indent=4)
```

Finally, I needed to edit the *mark_task_complete()* method, as updating the completion state to the list and the JSON file used the selected row as the index for the list, but this will not be accurate if the user has hidden completed tasks. For example, the second task in *task_list_hidden* may be the third task in *task_list* (as the second task in *task_list* may be marked as completed).

I coded a smarter way of checking the list for the selected task. My program gets the details of the selected task (task title, subject, and due date), then it iterates through *task_list* to check for a task which matches to the user's selected task, using *enumerate()* and *num* to count the number of iterations.

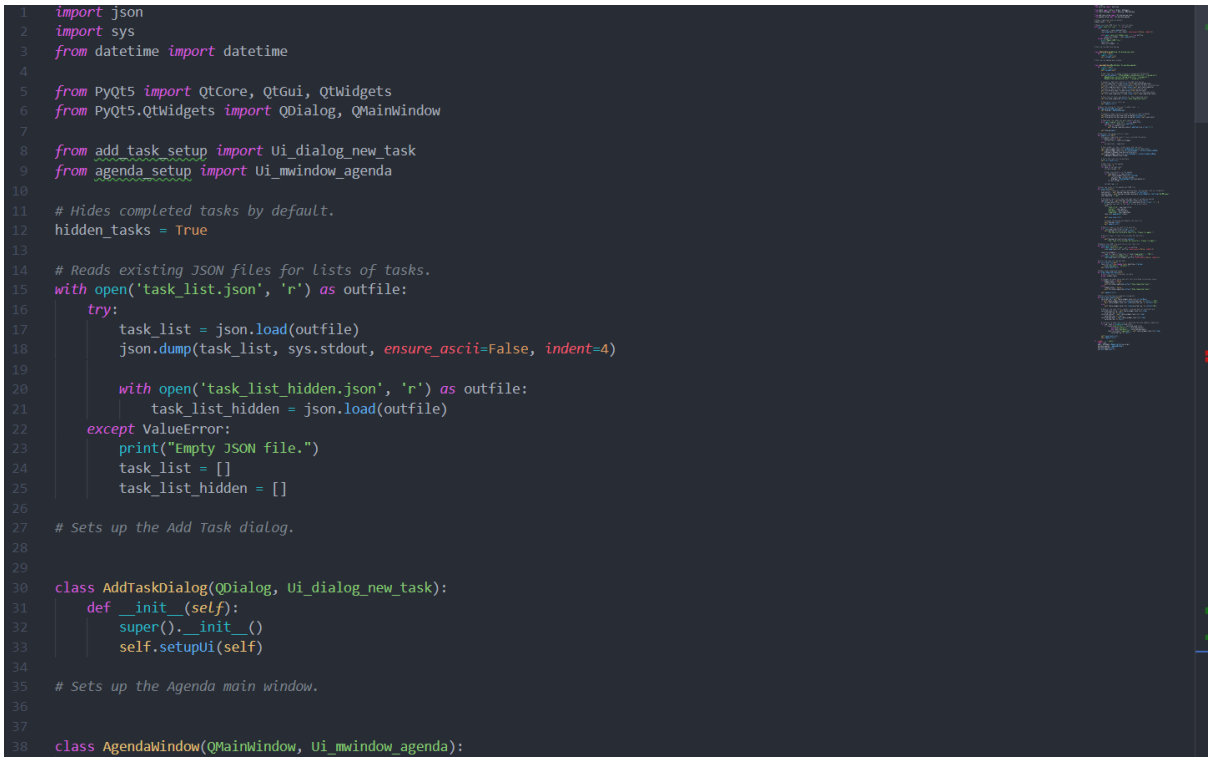
Once a matching task has been found, the 'completed' value for that task is changed to the new completion value ('Yes' or 'No').

After the completion value has been updated, the method saves the task list with *save_task_list()*, and updates the table widget with *update_list()*.

```
        # Obtains the task title, subject, and due date of selected task.
        selected_task_title = self.table_widget_task_list.item(
            selected_row, 0).text()
        selected_subject = self.table_widget_task_list.item(
            selected_row, 1).text()
        selected_due_date = self.table_widget_task_list.item(
            selected_row, 2).text()

        # Iterates through task list for matching task and updates completion.
        for num, task in enumerate(task_list):
            if (task["task_title"] == selected_task_title
                    and task["subject"] == selected_subject
                    and task["due_date"] == selected_due_date):
                task_list[num]["completed"] = self.table_widget_task_list.item(
                    selected_row, 3).text()

        self.save_task_list()
        self.update_list()
```

At the end of the implementation of marking tasks as complete and showing/hiding completed tasks, I rearranged the methods to position them in a more logical order, which will make it easier to read through the code. More commonly used/general methods have been placed closer to the top, and methods which are relevant to one another are positioned closely.

These changes to the program cumulated to produce the following code.



```
import json
import sys
from datetime import datetime

from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QDialog, QMainWindow

from add_task_setup import Ui_dialog_new_task
from agenda_setup import Ui_mwindow_agenda

# Hides completed tasks by default.
hidden_tasks = True

# Reads existing JSON files for lists of tasks.
with open('task_list.json', 'r') as outfile:
    try:
        task_list = json.load(outfile)
        json.dump(task_list, sys.stdout, ensure_ascii=False, indent=4)

        with open('task_list_hidden.json', 'r') as outfile:
            task_list_hidden = json.load(outfile)
    except ValueError:
        print("Empty JSON file.")
        task_list = []
        task_list_hidden = []

# Sets up the Add Task dialog.


class AddTaskDialog(QDialog, Ui_dialog_new_task):
    def __init__(self):
        super().__init__()
        self.setupUi(self)

# Sets up the Agenda main window.


class AgendaWindow(QMainWindow, Ui_mwindow_agenda):
```

```
    def __init__(self):
        super().__init__()
        self.setupUi(self)

        # Sets the list of tasks to have a transparent background.
        self.setStyleSheet("""QTableWidget {background-color: transparent;}
            QHeaderView::section {background-color: transparent;}
            QHeaderView {background-color: transparent;}""")

        # Connects 'Add Task' button to the Add Task dialog.
        self.btn_add_task.clicked.connect(self.open_dialog_add_task)
        # Connects 'Mark as Complete/Incomplete' button to mark selected task.
        self.btn_complete_task.clicked.connect(self.mark_task_complete)
        # Connects 'Delete Task' button to delete selected task.
        self.btn_delete_task.clicked.connect(self.delete_task)
        # Connects 'Hide/Show Completed Tasks' button to hide/show tasks.
        self.btn_hide_completed.clicked.connect(self.hide_completed_tasks)

        # Sets text of task view button to 'Show Completed Tasks'.
        self.btn_hide_completed.setText("Show Completed Tasks")

        # Populates list on start-up.
        self.update_list()

    # Opens the dialog for the user to add a task.  u
    def open_dialog_add_task(self):
        self.Dialog = AddTaskDialog()

        # Connects 'Save' button to save the user's task to Agenda.
        self.Dialog.button_box_new_task.accepted.disconnect()
        self.Dialog.button_box_new_task.accepted.connect(self.save_task)

        # Populates the combo box with subject options.
        with open("subject_list.txt", "r") as data_file:
            subject_list = data_file.readlines()
            for line in subject_list:
                self.Dialog.comb_box_subject.addItem(line.strip("\n"))

        self.Dialog.open()

    # Populates the agenda with all tasks.
    def update_list(self):
        # Displays completed tasks if user selected the option.
        if hidden_tasks is True:
            current_list = task_list_hidden
        else:
            current_list = task_list

        # Sets number of rows, column size, and row size.
        self.table_widget_task_list.setRowCount(len(current_list))
        self.table_widget_task_list.horizontalHeader().setSectionResizeMode(
            QtWidgets.QHeaderView.ResizeToContents)
        self.table_widget_task_list.verticalHeader().setSectionResizeMode(
            QtWidgets.QHeaderView.Fixed)

        # Sorts the task list by due date.
        self.sort_task_list()

        # Adds tasks to the agenda.
        current_row = 0
        for task in current_list:
            current_column = 0

            # Adds task details to the agenda.
            for task_details in task.values():
                self.table_widget_task_list.setItem(
                    current_row, current_column,
                    QtWidgets.QTableWidgetItem(task_details))
                current_column += 1

            current_row += 1

    # Saves new tasks to the agenda and JSON file.
    def save_task(self):
        # Gets user inputs for subject and due date, and assigns task as incomplete.
        new_subject = self.Dialog.comb_box_subject.currentText()
```

```
        new_due_date = self.Dialog.calendar_due_date.selectedDate().toString("dd/MM/yyyy")
        new_completed = ("No")

        # Validates task title input and adds task if validation passed.
        new_task_title = self.Dialog.line_edit_task_title.text()
        if len(new_task_title) <= 30 and len((new_task_title).strip(" ")) > 0:
            # Appends new task to the task list as a dictionary.
            task = {
                "task_title": new_task_title,
                "subject": new_subject,
                "due_date": new_due_date,
                "completed": new_completed}
            task_list.append(dict(task))

            self.save_task_list()

            # Closes the dialog and updates the task list.
            self.Dialog.close()
            self.update_list()

        # Rejects input if no task title entered.
        elif len((new_task_title).strip(" ")) == 0:
            self.Dialog.lbl_instruction.setText(
                "You have not entered a task title. Please try again.")

        # Rejects input if task title exceeds 30 characters.
        else:
            self.Dialog.lbl_instruction.setText(
                "Your task title exceeds 30 characters. Please try again.")

    # Updates the JSON file with the current task list.
    def save_task_list(self):
        with open('task_list.json', 'w') as outfile:
            json.dump(task_list, outfile, ensure_ascii=False, indent=4)

        task_list_hidden[:] = [
            task for task in task_list if task["completed"] == ("No")]
        with open('task_list_hidden.json', 'w') as outfile:
            json.dump(task_list_hidden, outfile, ensure_ascii=False, indent=4)

    # Sorts the task list by due date.
    def sort_task_list(self):
        task_list.sort(key=lambda task: datetime.strptime(
            task["due_date"], '%d/%m/%Y'))
        self.save_task_list()

    # Hides/shows completed tasks.
    def hide_completed_tasks(self):
        # Calls for use of the global variable.
        global hidden_tasks

        # Changes variable value and sets text according to previous value.
        if hidden_tasks is False:
            hidden_tasks = True
            self.btn_hide_completed.setText("Show Completed Tasks")
        else:
            hidden_tasks = False
            self.btn_hide_completed.setText("Hide Completed Tasks")

        self.update_list()

    # Marks selected task as complete/incomplete.
    def mark_task_complete(self):
        selected_row = self.table_widget_task_list.currentRow()
        if self.table_widget_task_list.item(selected_row, 3).text() == "No":
            self.table_widget_task_list.item(selected_row, 3).setText("Yes")
        else:
            self.table_widget_task_list.item(selected_row, 3).setText("No")

        # Obtains the task title, subject, and due date of selected task.
        selected_task_title = self.table_widget_task_list.item(
            selected_row, 0).text()
        selected_subject = self.table_widget_task_list.item(
            selected_row, 1).text()
        selected_due_date = self.table_widget_task_list.item(
            selected_row, 2).text()
```

```
        # Iterates through task list for matching task and updates completion.
        for num, task in enumerate(task_list):
            if (task["task_title"] == selected_task_title
                    and task["subject"] == selected_subject
                    and task["due_date"] == selected_due_date):
                task_list[num]["completed"] = self.table_widget_task_list.item(
                    selected_row, 3).text()

        self.save_task_list()
        self.update_list()

if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    mwindow_agenda = AgendaWindow()
    mwindow_agenda.show()
    sys.exit(app.exec_())

```

The final feature I developed in this development iteration was the deletion of tasks.

However, first I added an additional line in *.setStyleSheet()* to remove the white background from the disabled corner button (*QTableCornerButton::section)* in the *QTableWidget*. I set the background colour to be transparent to remove this white background.

```
        # Sets the list of tasks to have a transparent background.
        self.setStyleSheet("""QTableWidget {background-color: transparent;}
            QHeaderView::section {background-color: transparent;}
            QHeaderView {background-color: transparent;}
            QTableCornerButton::section{background-color: transparent;}""")
```

Then, I moved onto implementing the deletion of tasks feature. I started by connecting the press of *btn_delete_task* to a new method, *delete_task()*.

```
        # Connects 'Delete Task' button to delete selected task.
        self.btn_delete_task.clicked.connect(self.delete_task)
```

Coding *delete_task()* was a relatively simple task, as the processes needed to perform the deletion of the selected task from the list was similar to marking the selected task as complete/incomplete.

To start with, the method obtains the row of the selected task. Then, using this row, it obtains the task title, subject, and due date.

Next, the method iterates through each task in *task_list* using a *for* loop. When the method finds a task that matches the task title, subject, and due date, it removes that task from the list using *.remove()* on the *task_list*.

At the end of the method, the task list is saved, and the list displayed through the table widget is updated.

```
204         # Deletes selected task from agenda.
205         def delete_task(self):
206             selected_row = self.table_widget_task_list.currentRow()
207 
208             # Obtains the task title, subject, and due date of selected task.
209             selected_task_title = self.table_widget_task_list.item(
210                 selected_row, 0).text()
211             selected_subject = self.table_widget_task_list.item(
212                 selected_row, 1).text()
213             selected_due_date = self.table_widget_task_list.item(
214                 selected_row, 2).text()
215 
216             # Iterates through task list and deletes matching task.
217             for task in task_list:
218                 if (task["task_title"] == selected_task_title
219                         and task["subject"] == selected_subject
220                         and task["due_date"] == selected_due_date):
221                     task_list.remove(task)
222 
223             self.save_task_list()
224             self.update_list()
```

By the end of this development iteration, I had developed features for sorting the agenda, marking tasks as completed (and showing/hiding completed tasks), and deleting tasks from the agenda.

The code for my program at the end of this development iteration can be seen below.

```
1   import json
2   import sys
3   from datetime import datetime
4   
5   from PyQt5 import QtCore, QtGui, QtWidgets
6   from PyQt5.QtWidgets import QDialog, QMainWindow
7   
8   from add_task_setup import Ui_dialog_new_task
9   from agenda_setup import Ui_mwindow_agenda
10  
11  # Hides completed tasks by default.
12  hidden_tasks = True
13  
14  # Reads existing JSON files for lists of tasks.
15  with open('task_list.json', 'r') as outfile:
16      try:
17          task_list = json.load(outfile)
18          json.dump(task_list, sys.stdout, ensure_ascii=False, indent=4)
19  
20          with open('task_list_hidden.json', 'r') as outfile:
21              task_list_hidden = json.load(outfile)
22      except ValueError:
23          print("Empty JSON file.")
24          task_list = []
25          task_list_hidden = []
26  
27  # Sets up the Add Task dialog.
28  
29  
30  class AddTaskDialog(QDialog, Ui_dialog_new_task):
31      def __init__(self):
32          super().__init__()
33          self.setupUi(self)
34  
35  # Sets up the Agenda main window.
36  
37  
38  class AgendaWindow(QMainWindow, Ui_mwindow_agenda):
```

```
    def __init__(self):
        super().__init__()
        self.setupUi(self)

        # Sets the list of tasks to have a transparent background.
        self.setStyleSheet("""QTableWidget {background-color: transparent;}
            QHeaderView::section {background-color: transparent;}
            QHeaderView {background-color: transparent;}
            QTableCornerButton::section{background-color: transparent;}""")

        # Connects 'Add Task' button to the Add Task dialog.
        self.btn_add_task.clicked.connect(self.open_dialog_add_task)
        # Connects 'Mark as Complete/Incomplete' button to mark selected task.
        self.btn_complete_task.clicked.connect(self.mark_task_complete)
        # Connects 'Delete Task' button to delete selected task.
        self.btn_delete_task.clicked.connect(self.delete_task)
        # Connects 'Hide/Show Completed Tasks' button to hide/show tasks.
        self.btn_hide_completed.clicked.connect(self.hide_completed_tasks)

        # Sets text of task view button to 'Show Completed Tasks'.
        self.btn_hide_completed.setText("Show Completed Tasks")

        # Populates list on start-up.
        self.update_list()

    # Opens the dialog for the user to add a task.  u
    def open_dialog_add_task(self):
        self.Dialog = AddTaskDialog()

        # Connects 'Save' button to save the user's task to Agenda.
        self.Dialog.button_box_new_task.accepted.disconnect()
        self.Dialog.button_box_new_task.accepted.connect(self.save_task)

        # Populates the combo box with subject options.
        with open("subject_list.txt", "r") as data_file:
            subject_list = data_file.readlines()
            for line in subject_list:
                self.Dialog.comb_box_subject.addItem(line.strip("\n"))
```

```

        self.Dialog.open()

    # Populates the agenda with all tasks.
    def update_list(self):
        # Displays completed tasks if user selected the option.
        if hidden_tasks is True:
            current_list = task_list_hidden
        else:
            current_list = task_list

        # Sets number of rows, column size, and row size.
        self.table_widget_task_list.setRowCount(len(current_list))
        self.table_widget_task_list.horizontalHeader().setSectionResizeMode(
            QtWidgets.QHeaderView.ResizeToContents)
        self.table_widget_task_list.verticalHeader().setSectionResizeMode(
            QtWidgets.QHeaderView.Fixed)

        # Sorts the task list by due date.
        self.sort_task_list()

        # Adds tasks to the agenda.
        current_row = 0
        for task in current_list:
            current_column = 0

            # Adds task details to the agenda.
            for task_details in task.values():
                self.table_widget_task_list.setItem(
                    current_row, current_column,
                    QtWidgets.QTableWidgetItem(task_details))
                current_column += 1

            current_row += 1

    # Saves new tasks to the agenda and JSON file.
    def save_task(self):
        # Gets user inputs for subject and due date, and assigns task as incomplete.
```

```
        new_subject = self.Dialog.comb_box_subject.currentText()
        new_due_date = self.Dialog.calendar_due_date.selectedDate().toString("dd/MM/yyyy")
        new_completed = ("No")

        # Validates task title input and adds task if validation passed.
        new_task_title = self.Dialog.line_edit_task_title.text()
        if len(new_task_title) <= 30 and len((new_task_title).strip(" ")) > 0:
            # Appends new task to the task list as a dictionary.
            task = {
                "task_title": new_task_title,
                "subject": new_subject,
                "due_date": new_due_date,
                "completed": new_completed}
            task_list.append(dict(task))

            self.save_task_list()

            # Closes the dialog and updates the task list.
            self.Dialog.close()
            self.update_list()

        # Rejects input if no task title entered.
        elif len((new_task_title).strip(" ")) == 0:
            self.Dialog.lbl_instruction.setText(
                "You have not entered a task title. Please try again.")

        # Rejects input if task title exceeds 30 characters.
        else:
            self.Dialog.lbl_instruction.setText(
                "Your task title exceeds 30 characters. Please try again.")

    # Updates the JSON file with the current task list.
    def save_task_list(self):
        with open('task_list.json', 'w') as outfile:
            json.dump(task_list, outfile, ensure_ascii=False, indent=4)

        task_list_hidden[:] = [
            task for task in task_list if task["completed"] == ("No")]
```

```
        with open('task_list_hidden.json', 'w') as outfile:
            json.dump(task_list_hidden, outfile, ensure_ascii=False, indent=4)

    # Sorts the task list by due date.
    def sort_task_list(self):
        task_list.sort(key=lambda task: datetime.strptime(
            task["due_date"], '%d/%m/%Y'))
        self.save_task_list()

    # Hides/shows completed tasks.
    def hide_completed_tasks(self):
        # Calls for use of the global variable.
        global hidden_tasks

        # Changes variable value and sets text according to previous value.
        if hidden_tasks is False:
            hidden_tasks = True
            self.btn_hide_completed.setText("Show Completed Tasks")
        else:
            hidden_tasks = False
            self.btn_hide_completed.setText("Hide Completed Tasks")

        self.update_list()

    # Marks selected task as complete/incomplete.
    def mark_task_complete(self):
        selected_row = self.table_widget_task_list.currentRow()
        if self.table_widget_task_list.item(selected_row, 3).text() == "No":
            self.table_widget_task_list.item(selected_row, 3).setText("Yes")
        else:
            self.table_widget_task_list.item(selected_row, 3).setText("No")

        # Obtains the task title, subject, and due date of selected task.
        selected_task_title = self.table_widget_task_list.item(
            selected_row, 0).text()
        selected_subject = self.table_widget_task_list.item(
            selected_row, 1).text()
        selected_due_date = self.table_widget_task_list.item(
```

```
            selected_row, 2).text()

        # Iterates through task list for matching task and updates completion.
        for num, task in enumerate(task_list):
            if (task["task_title"] == selected_task_title
                    and task["subject"] == selected_subject
                    and task["due_date"] == selected_due_date):
                task_list[num]["completed"] = self.table_widget_task_list.item(
                    selected_row, 3).text()

        self.save_task_list()
        self.update_list()

    # Deletes selected task from agenda.
    def delete_task(self):
        selected_row = self.table_widget_task_list.currentRow()

        # Obtains the task title, subject, and due date of selected task.
        selected_task_title = self.table_widget_task_list.item(
            selected_row, 0).text()
        selected_subject = self.table_widget_task_list.item(
            selected_row, 1).text()
        selected_due_date = self.table_widget_task_list.item(
            selected_row, 2).text()

        # Iterates through task list and deletes matching task.
        for task in task_list:
            if (task["task_title"] == selected_task_title
                    and task["subject"] == selected_subject
                    and task["due_date"] == selected_due_date):
                task_list.remove(task)

        self.save_task_list()
        self.update_list()

```

```
if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    mwindow_agenda = AgendaWindow()
    mwindow_agenda.show()
    sys.exit(app.exec_())

```

Testing: Iteration #2 – Sorting, Marking, and Deleting

This development iteration involved a lot of new features being added, so it was important to test that these features were implemented correctly. I started by beta testing the inputs for the program.

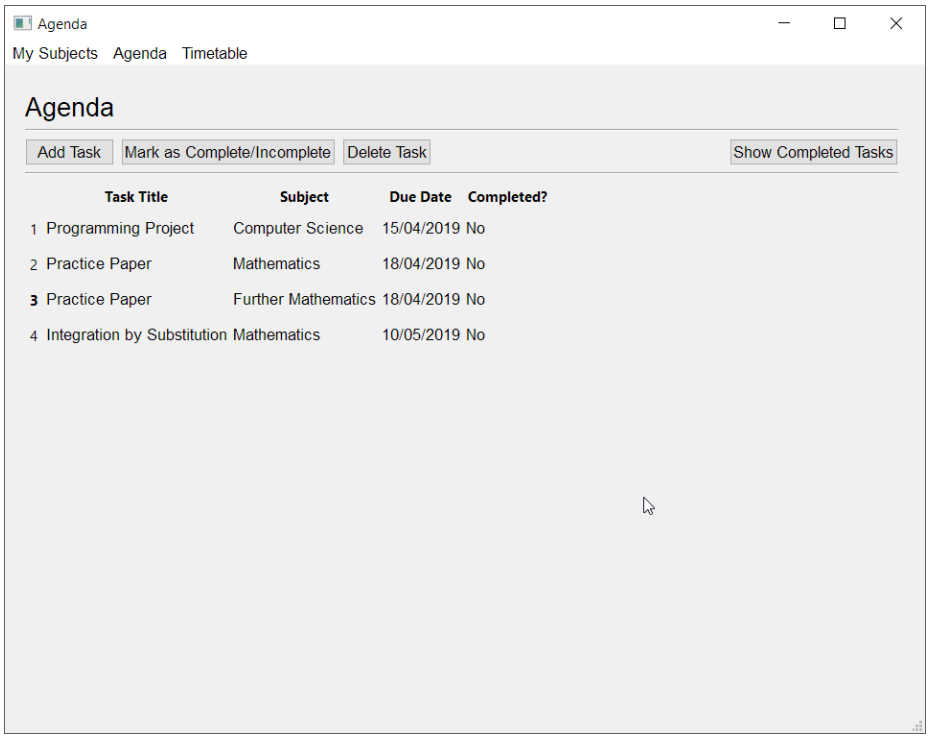
Input	Nature of Data	Data Validation	Is Data Accepted?	Output
Typed ‘Programming Project’ for the task title, selected ‘Computer Science’ from the dropdown menu as the subject, and the 15th April 2019 as the due date, then saved the task by pressing the ‘Save’ button.	Normal	Presence check, length check, button pressed	Yes	The task was added to the agenda, with the same task title and subject as input, the due date displayed in DD-MM-YYYY format, and marked as incomplete.
Typed ‘Practice Paper’ for the task title, selected ‘Mathematics’ from the dropdown menu as the subject,	Normal	Presence check, length check, button pressed	Yes	The task was added to the agenda, with the same task title and subject as input, the due date displayed in DD-MM-YYYY format, and

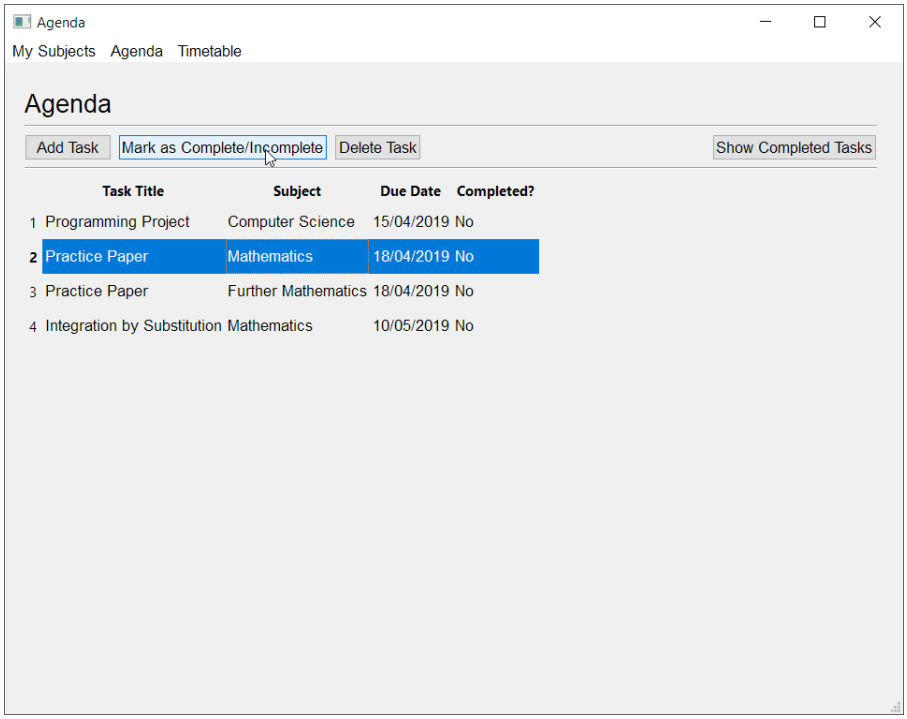
and the 18th April 2019 as the due date, then saved the task by pressing the 'Save' button.				marked as incomplete. It was placed second in the agenda, as it was second when sorted ascending by due date.
Typed 'Practice Paper' for the task title, selected 'Further Mathematics' from the dropdown menu as the subject, and the 18th April 2019 as the due date, then saved the task by pressing the 'Save' button.	Normal	Presence check, length check, button pressed	Yes	The task was added to the agenda, with the same task title and subject as input, the due date displayed in DD-MM-YYYY format, and marked as incomplete. It was placed third in the agenda, as it was joint-second when sorted ascending by due date, and it was the latest of these tasks to be added.
Typed 'Integration by Substitution' for the task title, selected 'Mathematics' from the dropdown menu as the subject, and the 10th May 2019 as the due date, then saved the task by pressing the 'Save' button.	Extreme	Presence check, length check, button pressed	Yes	The task was added to the agenda, with the same task title and subject as input, the due date displayed in DD-MM-YYYY format, and marked as incomplete. It was placed last in the agenda, as it was last when sorted ascending by due date.
Typed ' ' [three spaces] for the task title, 'Further Mathematics' from the dropdown menu as the subject, and 31st May 2019 as the due date, then saved the task by pressing the 'Save' button.	Erroneous	Presence check, length check, button pressed	Yes	The task was not added to the agenda, and the instruction label in the dialog changed to specify that they had not entered a task title, and to try again.

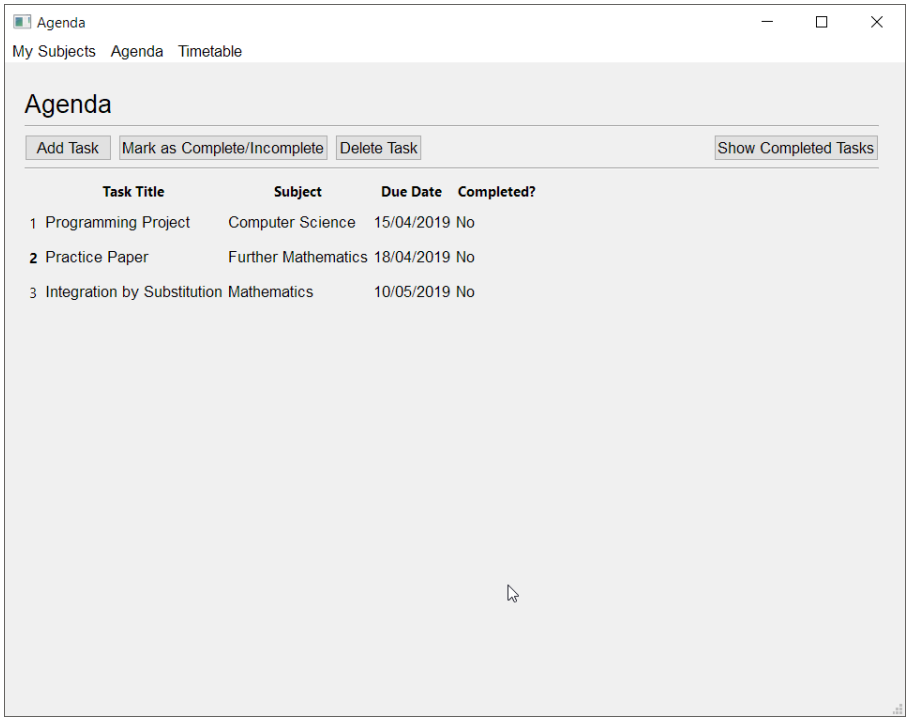
Typed nothing for the task title, and left the subject and due date as default, then saved the task by pressing the 'Save' button.	Null	Presence check, length check, button pressed	Yes	The task was not added to the agenda, and the instruction label in the dialog changed to specify that they had not entered a task title, and to try again.
Selected the second task ('Practice Paper' for 'Mathematics'), and pressed the button to 'Mark as Complete/Incomplete'.	Normal	Button pressed	Yes	The task was marked as completed, with the 'Completed?' value changing from 'No' to 'Yes'. The completed task was no longer displayed in the agenda, as the agenda was hiding completed tasks.
Pressed the 'Show Completed Tasks' button.	Normal	Button pressed	Yes	The completed task was displayed to the agenda alongside the other tasks in ascending due date order.
Pressed the 'Hide Completed Tasks' button, selected the second task ('Practice Paper' for 'Further Mathematics'), and pressed the 'Mark as Complete/Incomplete' button.	Normal	Button pressed	Yes	The task was marked as completed, with the 'Completed?' value changing from 'No' to 'Yes'. The completed task was no longer displayed in the agenda, as the agenda was hiding completed tasks.
Pressed the 'Show Completed Tasks' button, selected the third task ('Practice Paper' for 'Further Mathematics'), and pressed the 'Mark as Complete/Incomplete' button.	Normal	Button pressed	Yes	The task was marked as incomplete, with the 'Completed?' value changing from 'Yes' to 'No'.
Selected the fourth task ('Integration by Substitution' for	Normal	Button pressed	Yes	The task was deleted from the agenda.

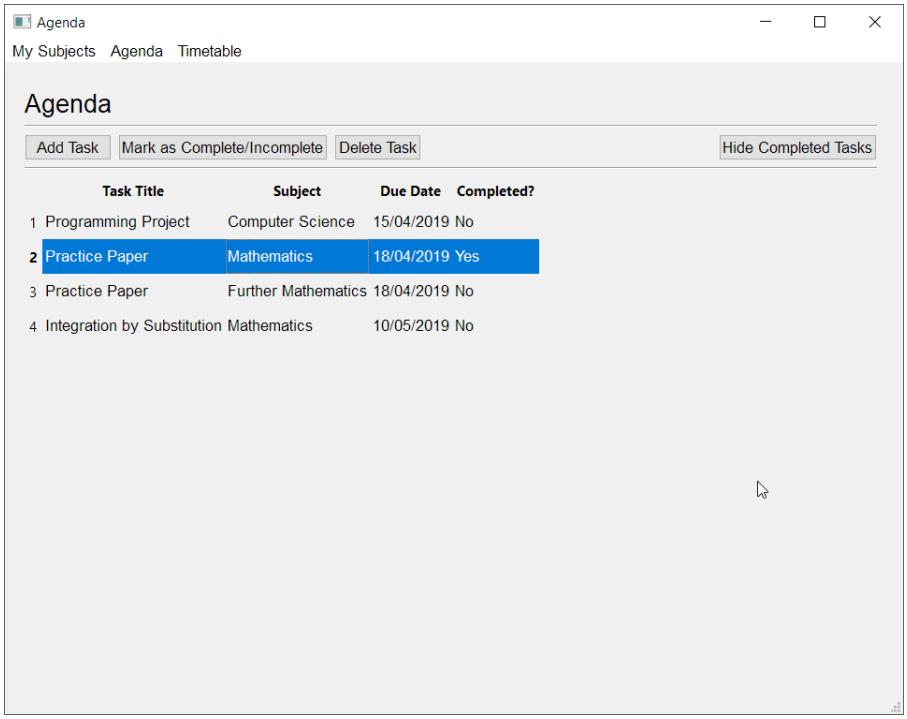
'Mathematics') and pressed the 'Delete Task' button.				

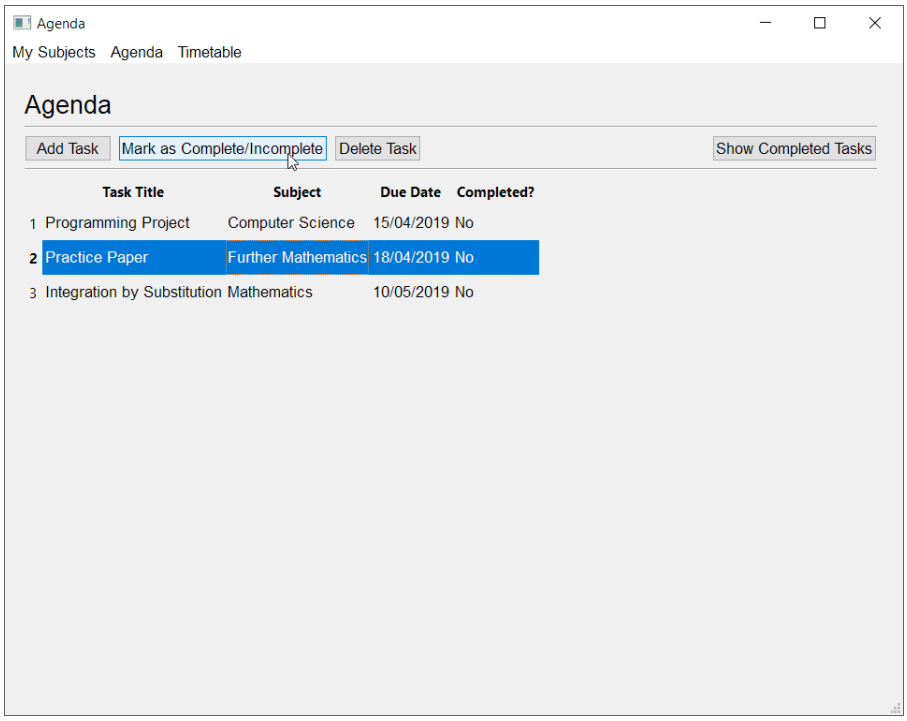
Some screenshots demonstrating the beta testing process for inputs can be seen below.

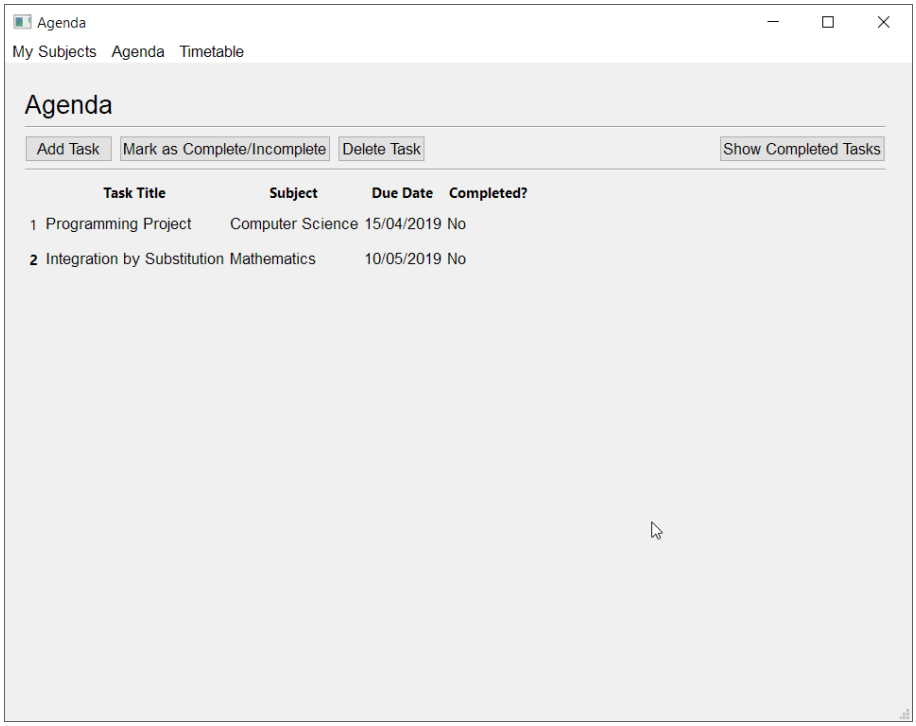


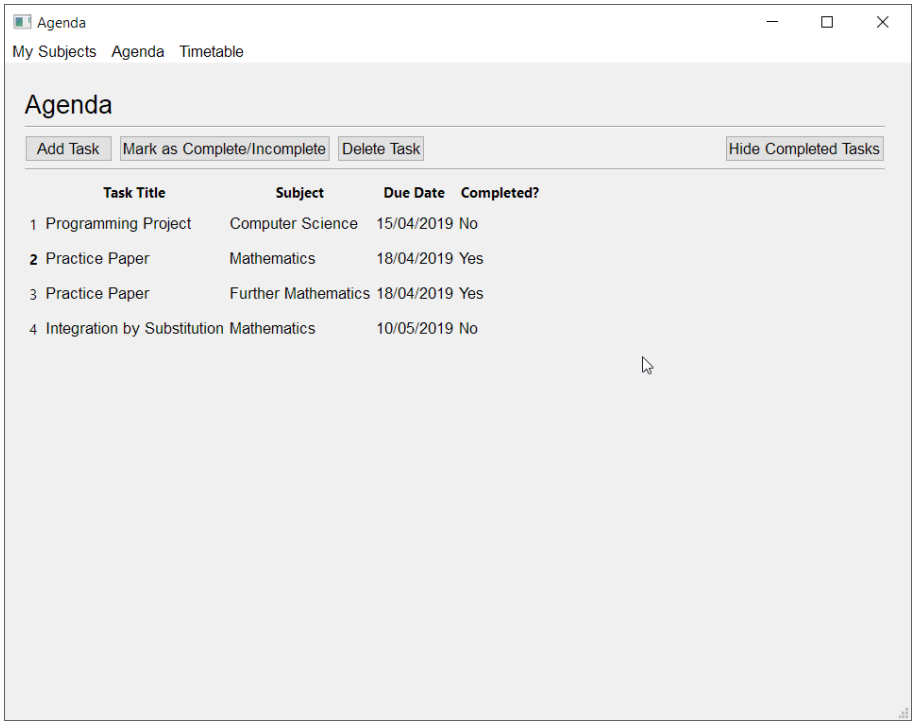

Agenda
My Subjects Agenda Timetable
Agenda
Add Task
Mark as Complete/Incomplete
Delete Task
Show Completed Tasks
Task Title Subject Due Date Completed?
1 Programming Project Computer Science 15/04/2019 No
2 Practice Paper Mathematics 18/04/2019 No
3 Practice Paper Further Mathematics 18/04/2019 No
4 Integration by Substitution Mathematics 10/05/2019 No

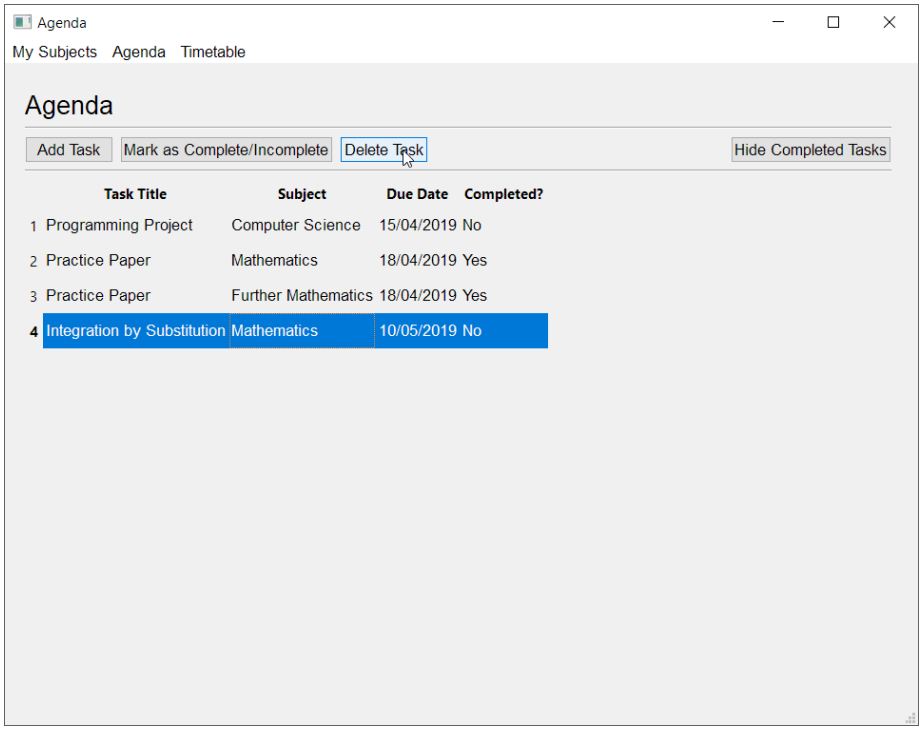

Agenda
My Subjects Agenda Timetable
Agenda
Add Task
Mark as Complete/Incomplete
Delete Task
Show Completed Tasks
Task Title Subject Due Date Completed?
1 Programming Project Computer Science 15/04/2019 No
2 Practice Paper Further Mathematics 18/04/2019 No
3 Integration by Substitution Mathematics 10/05/2019 No

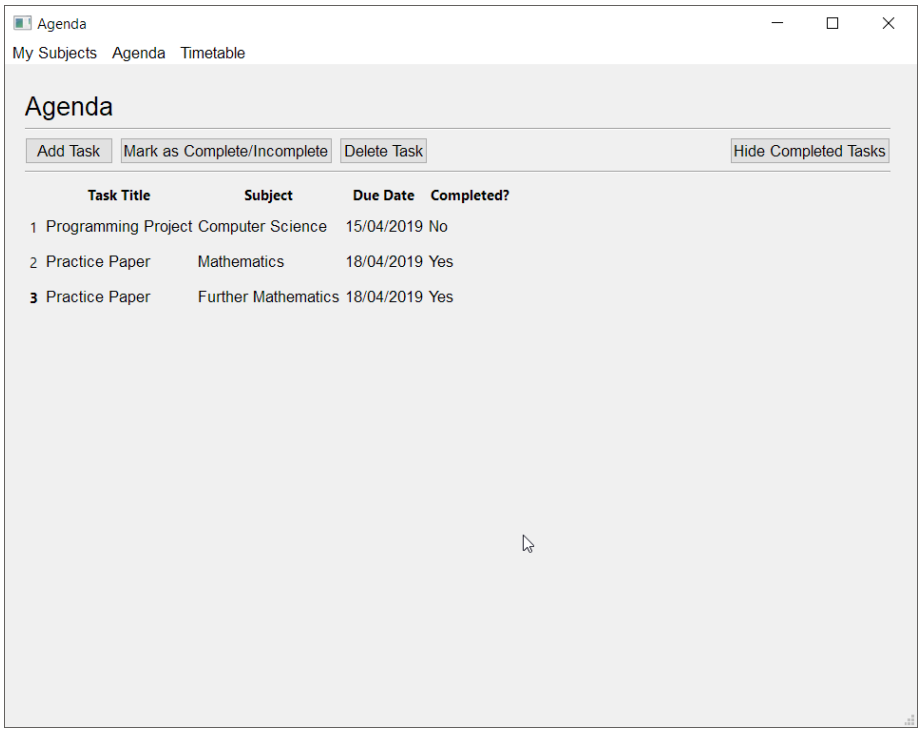

Agenda
My Subjects Agenda Timetable
Agenda
Add Task
Mark as Complete/Incomplete
Delete Task
Hide Completed Tasks
Task Title Subject Due Date Completed?
1 Programming Project Computer Science 15/04/2019 No
2 Practice Paper Mathematics 18/04/2019 Yes
3 Practice Paper Further Mathematics 18/04/2019 No
4 Integration by Substitution Mathematics 10/05/2019 No


Agenda
My Subjects Agenda Timetable
Agenda
Add Task
Mark as Complete/Incomplete
Delete Task
Show Completed Tasks
Task Title Subject Due Date Completed?
1 Programming Project Computer Science 15/04/2019 No
2 Practice Paper Further Mathematics 18/04/2019 No
3 Integration by Substitution Mathematics 10/05/2019 No


Agenda
My Subjects Agenda Timetable
Agenda
Add Task
Mark as Complete/Incomplete
Delete Task
Show Completed Tasks
Task Title Subject Due Date Completed?
1 Programming Project Computer Science 15/04/2019 No
2 Integration by Substitution Mathematics 10/05/2019 No


Agenda
My Subjects Agenda Timetable
Agenda
Add Task
Mark as Complete/Incomplete
Delete Task
Hide Completed Tasks
Task Title Subject Due Date Completed?
1 Programming Project Computer Science 15/04/2019 No
2 Practice Paper Mathematics 18/04/2019 Yes
3 Practice Paper Further Mathematics 18/04/2019 Yes
4 Integration by Substitution Mathematics 10/05/2019 No


Agenda
My Subjects Agenda Timetable
Agenda
Add Task
Mark as Complete/Incomplete
Delete Task
Hide Completed Tasks
Task Title Subject Due Date Completed?
1 Programming Project Computer Science 15/04/2019 No
2 Practice Paper Mathematics 18/04/2019 Yes
3 Practice Paper Further Mathematics 18/04/2019 Yes
4 Integration by Substitution Mathematics 10/05/2019 No


Agenda
My Subjects Agenda Timetable
Agenda
Add Task
Mark as Complete/Incomplete
Delete Task
Hide Completed Tasks
Task Title Subject Due Date Completed?
1 Programming Project Computer Science 15/04/2019 No
2 Practice Paper Mathematics 18/04/2019 Yes
3 Practice Paper Further Mathematics 18/04/2019 Yes

agenda.py | task_list.json | task_list_hidden.json

```
[
    {
        "task_title": "Programming Project",
        "subject": "Computer Science",
        "due_date": "15/04/2019",
        "completed": "No"
    },
    {
        "task_title": "Practice Paper",
        "subject": "Mathematics",
        "due_date": "18/04/2019",
        "completed": "Yes"
    },
    {
        "task_title": "Practice Paper",
        "subject": "Further Mathematics",
        "due_date": "18/04/2019",
        "completed": "Yes"
    }
]
```

agenda.py | task_list.json | task_list_hidden.json

```
[
    {
        "task_title": "Programming Project",
        "subject": "Computer Science",
        "due_date": "15/04/2019",
        "completed": "No"
    }
]
```

Due to the number of features added in this development iteration, I needed to perform many more tests for inputs than I did in previous development iterations. I ensured that every scenario was tested, which meant that I could be assured that all the features were functioning as intended.

All the tests resulted in the expected outputs, meaning that the features were successfully implemented.

To test my program against the objectives set at the start of the stage, I performed more general tests, including the tests in the previous development iteration which did not produce the expected result.

Test	**Expected Result**	**Outcome**	**Further Actions**
Are user inputs when adding a task validated using presence and length checks?	The user should have their line edit input for the task title validated using a presence check and a length check for 30 characters or less. (Subject and due date are already selected by default, so there are no cases when a presence check would be failed.) If validation is passed, the task should be added to the agenda.	The user has their line edit input for the task title validated using a presence check and a length check for 30 characters or less. Tasks are added even if there is an existing task with the same task title.	N/A.

Are tasks on the agenda displayed in sorted chronological order by due date?	The tasks should be displayed in sorted chronological order by due date, meaning that tasks with earlier due dates should be positioned above tasks with later due dates.	The tasks were displayed in sorted chronological order by due date, meaning that tasks with earlier due dates were positioned above tasks with later due dates. In cases where two or more tasks have the same due date, tasks which were added to the agenda earlier were positioned higher.	N/A.
Is the user able to mark tasks as complete/incomplete?	The user should be able to change the completion status of tasks by selecting a task, then pressing on the 'Mark as Complete/Incomplete' button.	The user is able to change the completion status of tasks by selecting a task, then pressing on the 'Mark as Complete/Incomplete' button. This changes the status from 'No' to 'Yes', and vice versa.	N/A.
Is the user able to delete tasks from the agenda?	The user should be able to delete tasks by selecting a task, then pressing on the 'Delete Task' button.	The user is able to delete tasks by selecting a task, then pressing on the 'Delete Task' button.	N/A.
Is the user able to hide/show completed tasks?	The user should be able to hide/show completed tasks by clicking on the 'Hide Completed Tasks' or 'Show Completed Tasks' button.	The user is able to hide/show completed tasks by clicking on the 'Hide Completed Tasks' or 'Show Completed Tasks' button.	N/A.

General testing has demonstrated that my program meets all the objectives, and that previous problems in the program have been addressed by the fixes in this development iteration.

Review

My objectives for this stage were to develop full functionality for Agenda, with the ability to add, mark, and delete tasks, as well as hide/show completed tasks.

I have successfully developed a program with implementations of these features in a robust, efficient manner with validation where necessary. The user is able to use an agenda which

allows them to add tasks to an agenda which is sorted by due date, mark tasks as complete/incomplete, delete tasks, and hide/show completed tasks.

Stage 8: Editing Timetable

Objectives

My objectives for this stage are to update Timetable to add full functionality.

The user should be able to view their timetable in a 5 x 5 table format (representing five days and five periods), and edit timetable slots with the subject, teacher, and room.

Prototype Program

For my prototype program, I created a program similar to in stage 7, but applied to the context of editing a timetable.

This program will be more about adding a timetable slot, but it will provide the foundations for a method I could use in the first development iteration for editing existing timetable slots.

I was limited by the data I had to work with, so my program takes the inputs from the user to get the details of the lesson, and then adds the lesson as a dictionary onto the timetable list. This is similar to how I stored the tasks in stage 7.

At the start of the program, I imported *json*, as this is needed to work with JSON files in the program. Then, the program reads from *timetable.json*. If there are any lessons stored in that file, then the list is loaded into the variable *timetable*. Otherwise, the *timetable* list variable is defined as empty.

Then, the program gets the inputs from the user for the subject, teacher, and room. For *timetable_subject*, *timetable_teacher*, and *timetable_room*, it performs a length check to ensure that these inputs do not exceed 30 characters, 30 characters, and 20 characters respectively. This is to prevent display bugs caused by excessively long strings.

Out of the three inputs, only *timetable_subject* is subjected to a presence check. This is because users may not want to input a teacher and room for every lesson, as their teacher/room may be constantly changing.

Next, the program creates a dictionary to store this lesson in the *lesson* variable. Once this has been defined, the dictionary data in the *lesson* variable is appended to the *timetable* list variable using *.append()*. This effectively creates a list of dictionaries, as seen in the previous stage.

At the end, the *timetable* list variable is saved to *timetable.json* for permanent storage.

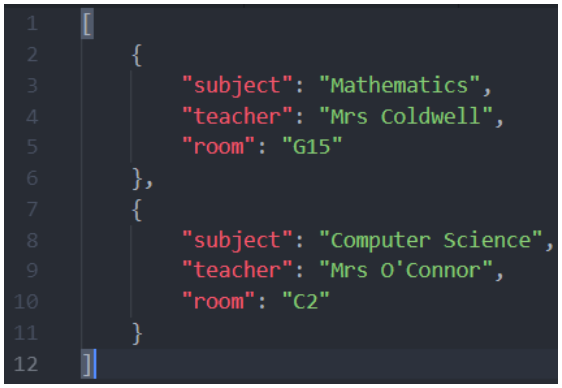
```python
import json

# Reads the existing JSON file for the timetable list.
with open('timetable.json', 'r') as outfile:
    try:
        timetable = json.load(outfile)
    except ValueError:
        timetable = []
        print("Empty JSON file.")

# Validates the entries of lesson details.
timetable_subject = str(input("\nSubject: "))
while len(timetable_subject) > 30:
    print("Your subject exceeds 30 characters. Please try again.\n")
    timetable_subject = str(input("Subject: "))
while len((timetable_subject).strip(" ")) == 0:
    print("You have not entered a subject. Please try again.\n")
    timetable_subject = str(input("Subject: "))

timetable_teacher = str(input("\nTeacher: "))
while len(timetable_teacher) > 30:
    print("The teacher name exceeds 30 characters. Please try again.\n")
    timetable_teacher = str(input("Teacher: "))

timetable_room = str(input("\nRoom: "))
while len(timetable_room) > 20:
    print("The room exceeds 20 characters. Please try again.\n")
    timetable_room = str(input("Room: "))

# Adds lesson to timetable.
lesson = {"subject": timetable_subject,
          "teacher": timetable_teacher,
          "room": timetable_room}
timetable.append(dict(lesson))

# Saves the timetable list to a JSON file.
with open('timetable.json', 'w') as outfile:
    json.dump(timetable, outfile, ensure_ascii=False, indent=4)

```

Once I created this prototype program, I performed some brief testing to ensure that it was working as intended.

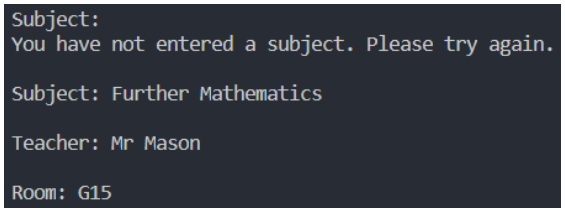
```
Empty JSON file.

Subject: Mathematics

Teacher: Mrs Coldwell

Room: G15
```

```json
[
    {
        "subject": "Mathematics",
        "teacher": "Mrs Coldwell",
        "room": "G15"
    }
]
```

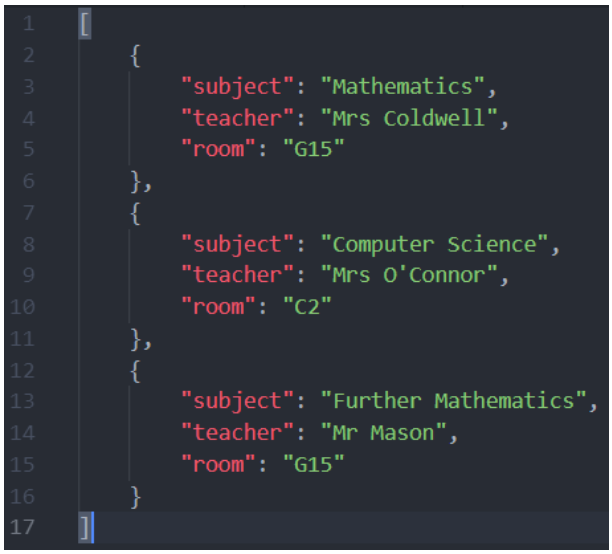
```
Subject: asdopkasopdkasopdkasopkdopaskdkopasopdkasopdasopkdopaskopdkaopsdkaskdopkasdopkaspdop
Your subject exceeds 30 characters. Please try again.

Subject: Computer Science

Teacher: dkopasdkasopdkopaskdopkasdopksopdkasokdopaskopdaskopdkasopkdopaskdkopaskopdkasop
The teacher name exceeds 30 characters. Please try again.

Teacher: Mrs O'Connor

Room: C2
```



```
[
    {
        "subject": "Mathematics",
        "teacher": "Mrs Coldwell",
        "room": "G15"
    },
    {
        "subject": "Computer Science",
        "teacher": "Mrs O'Connor",
        "room": "C2"
    }
]
```



```
Subject:
You have not entered a subject. Please try again.

Subject: Further Mathematics

Teacher: Mr Mason

Room: G15
```



```
[
    {
        "subject": "Mathematics",
        "teacher": "Mrs Coldwell",
        "room": "G15"
    },
    {
        "subject": "Computer Science",
        "teacher": "Mrs O'Connor",
        "room": "C2"
    },
    {
        "subject": "Further Mathematics",
        "teacher": "Mr Mason",
        "room": "G15"
    }
]
```

The screenshots show that the prototype program is working properly. The program reads the list of lessons from *timetable.json* if there are any lessons stored in there, obtains the user's validated inputs, then appends it to the list of lessons and save this list back to *timetable.json*.

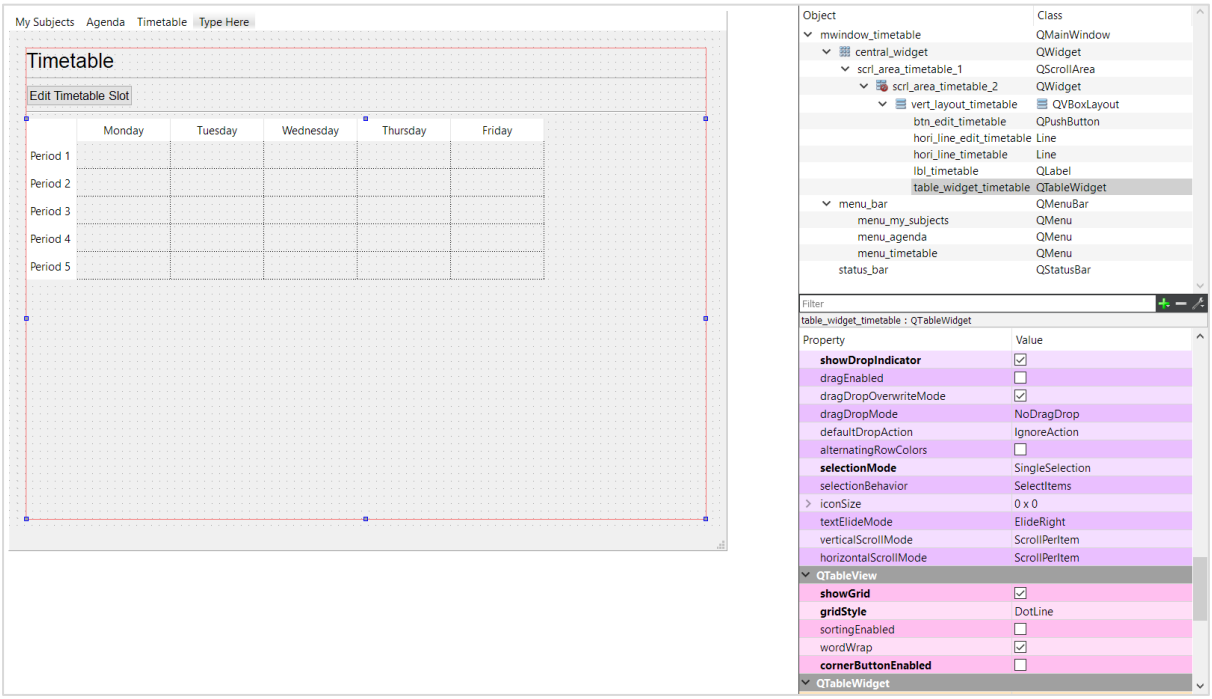
Development: Iteration #1 – User Interface

I started the development iteration by updating the user interface for Timetable in Qt Creator to use a *QTableWidget* like in Agenda, as the grid layout and the labels were not an efficient way of storing and displaying the timetable.

A key difference between the *QTableWidget* in Timetable and the one in Agenda is that this timetable has vertical headers to represent the periods which break down each day. In addition, the grids were enabled and shown as dotted lines in Timetable, as each item in the timetable is separate to each other, whereas items in the same row were considered to be linked in Agenda.

I called the new *QTableWidget* in Timetable *table_widget_timetable*, and changed the button from being called *btn_add_task* (which I realised was a mistake made in a previous stage) to

the name *btn_edit_timetable*. The text of this button was changed from ‘Edit Timetable’ to ‘Edit Timetable Slot’, as I have decided that it would be more intuitive for the user to select a timetable slot, then press the button to edit it.



I made these changes in Qt Creator, then I used *pyuic5* to generate the code for the user interface and save it as *timetable_setup.py*.

The most important part of the generated code is the class *Ui_mwindow_timetable*, as this initialises the user interface widgets, and this is the class which will be imported in the operational code, *timetable.py*.

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'timetable.ui'
#
# Created by: PyQt5 UI code generator 5.12.1
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_mwindow_timetable(object):
    def setupUi(self, mwindow_timetable):
        mwindow_timetable.setObjectName("mwindow_timetable")
        mwindow_timetable.resize(960, 720)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        font.setKerning(True)
        mwindow_timetable.setFont(font)
        self.central_widget = QtWidgets.QWidget(mwindow_timetable)
        self.central_widget.setObjectName("central_widget")
        self.gridLayout = QtWidgets.QGridLayout(self.central_widget)
        self.gridLayout.setObjectName("gridLayout")
        self.scrl_area_timetable_1 = QtWidgets.QScrollArea(self.central_widget)
        self.scrl_area_timetable_1.setFrameShape(QtWidgets.QFrame.NoFrame)
        self.scrl_area_timetable_1.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.scrl_area_timetable_1.setLineWidth(0)
        self.scrl_area_timetable_1.setWidgetResizable(True)
        self.scrl_area_timetable_1.setObjectName("scrl_area_timetable_1")
        self.scrl_area_timetable_2 = QtWidgets.QWidget()
        self.scrl_area_timetable_2.setGeometry(QtCore.QRect(0, 0, 938, 647))
        font = QtGui.QFont()
        font.setKerning(True)
        self.scrl_area_timetable_2.setFont(font)
        self.scrl_area_timetable_2.setAutoFillBackground(True)
        self.scrl_area_timetable_2.setObjectName("scrl_area_timetable_2")
        self.layoutWidget = QtWidgets.QWidget(self.scrl_area_timetable_2)
```

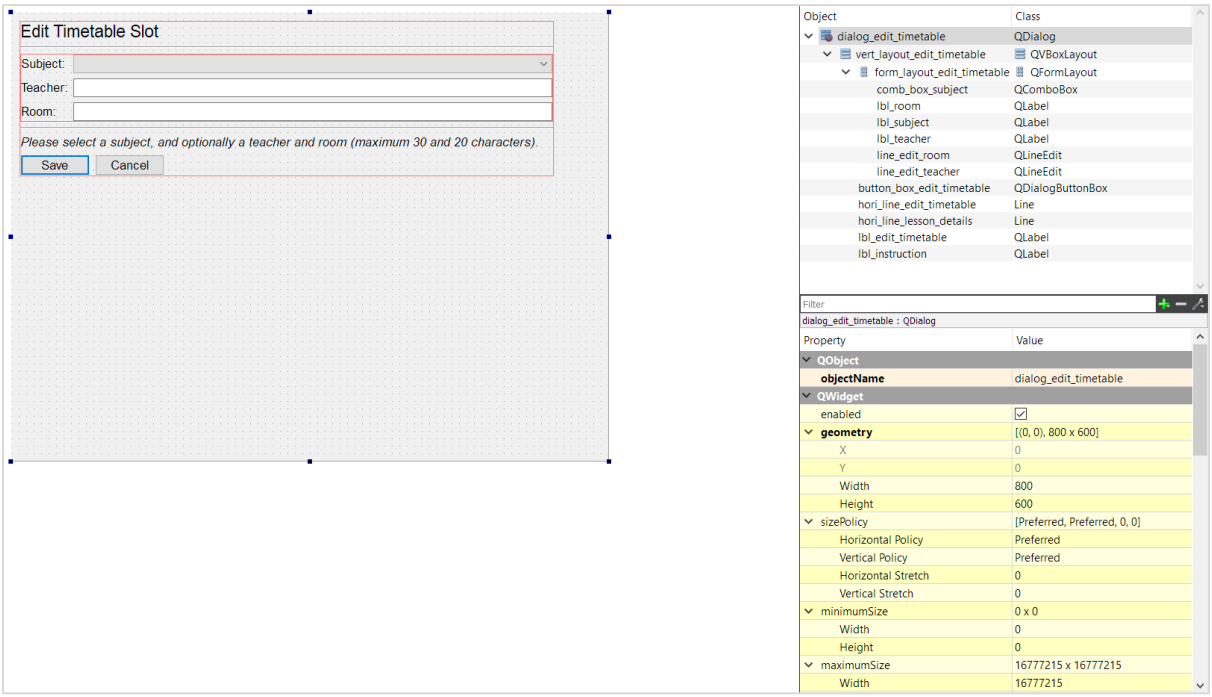
```
        self.layoutWidget.setGeometry(QtCore.QRect(11, 11, 911, 631))
        self.layoutWidget.setObjectName("layoutWidget")
        self.vert_layout_timetable = QtWidgets.QVBoxLayout(self.layoutWidget)
        self.vert_layout_timetable.setContentsMargins(0, 0, 0, 0)
        self.vert_layout_timetable.setObjectName("vert_layout_timetable")
        self.lbl_timetable = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setPointSize(16)
        self.lbl_timetable.setFont(font)
        self.lbl_timetable.setObjectName("lbl_timetable")
        self.vert_layout_timetable.addWidget(self.lbl_timetable)
        self.hori_line_timetable = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_timetable.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_timetable.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_timetable.setObjectName("hori_line_timetable")
        self.vert_layout_timetable.addWidget(self.hori_line_timetable)
        self.btn_edit_timetable = QtWidgets.QPushButton(self.layoutWidget)
        self.btn_edit_timetable.setObjectName("btn_edit_timetable")
        self.vert_layout_timetable.addWidget(
            self.btn_edit_timetable, 0, QtCore.Qt.AlignLeft)
        self.hori_line_edit_timetable = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_edit_timetable.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_edit_timetable.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_edit_timetable.setObjectName("hori_line_edit_timetable")
        self.vert_layout_timetable.addWidget(self.hori_line_edit_timetable)
        self.table_widget_timetable = QtWidgets.QTableWidget(self.layoutWidget)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.table_widget_timetable.setFont(font)
        self.table_widget_timetable.setStyleSheet(
            "background-color: transparent")
        self.table_widget_timetable.setFrameShape(QtWidgets.QFrame.NoFrame)
        self.table_widget_timetable.setEditTriggers(
            QtWidgets.QAbstractItemView.NoEditTriggers)
        self.table_widget_timetable.setTabKeyNavigation(False)
        self.table_widget_timetable.setProperty("showDropIndicator", True)
        self.table_widget_timetable.setSelectionMode(
            QtWidgets.QAbstractItemView.SingleSelection)
        self.table_widget_timetable.setShowGrid(True)
        self.table_widget_timetable.setGridStyle(QtCore.Qt.DotLine)
        self.table_widget_timetable.setCornerButtonEnabled(False)
        self.table_widget_timetable.setObjectName("table_widget_timetable")
        self.table_widget_timetable.setColumnCount(5)
        self.table_widget_timetable.setRowCount(5)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_timetable.setVerticalHeaderItem(0, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_timetable.setVerticalHeaderItem(1, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_timetable.setVerticalHeaderItem(2, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_timetable.setVerticalHeaderItem(3, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_timetable.setVerticalHeaderItem(4, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_timetable.setHorizontalHeaderItem(0, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_timetable.setHorizontalHeaderItem(1, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_timetable.setHorizontalHeaderItem(2, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_timetable.setHorizontalHeaderItem(3, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_timetable.setHorizontalHeaderItem(4, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_timetable.setItem(0, 0, item)
        self.table_widget_timetable.horizontalHeader().setHighlightSections(True)
        self.table_widget_timetable.horizontalHeader().setStretchLastSection(False)
        self.vert_layout_timetable.addWidget(self.table_widget_timetable)
        self.scrl_area_timetable_1.setWidget(self.scrl_area_timetable_2)
        self.gridLayout.addWidget(self.scrl_area_timetable_1, 0, 0, 1, 1)
        mwindow_timetable.setCentralWidget(self.central_widget)
        self.menu_bar = QtWidgets.QMenuBar(mwindow_timetable)
        self.menu_bar.setGeometry(QtCore.QRect(0, 0, 960, 26))
        self.menu_bar.setObjectName("menu_bar")
```

```
        self.menu_my_subjects = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_my_subjects.setFont(font)
        self.menu_my_subjects.setObjectName("menu_my_subjects")
        self.menu_agenda = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_agenda.setFont(font)
        self.menu_agenda.setObjectName("menu_agenda")
        self.menu_timetable = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_timetable.setFont(font)
        self.menu_timetable.setObjectName("menu_timetable")
        mwindow_timetable.setMenuBar(self.menu_bar)
        self.status_bar = QtWidgets.QStatusBar(mwindow_timetable)
        self.status_bar.setObjectName("status_bar")
        mwindow_timetable.setStatusBar(self.status_bar)
        self.actionAgenda = QtWidgets.QAction(mwindow_timetable)
        self.actionAgenda.setObjectName("actionAgenda")
        self.menu_bar.addAction(self.menu_my_subjects.menuAction())
        self.menu_bar.addAction(self.menu_agenda.menuAction())
        self.menu_bar.addAction(self.menu_timetable.menuAction())

        self.retranslateUi(mwindow_timetable)
        QtCore.QMetaObject.connectSlotsByName(mwindow_timetable)

    def retranslateUi(self, mwindow_timetable):
        _translate = QtCore.QCoreApplication.translate
        mwindow_timetable.setWindowTitle(
            _translate("mwindow_timetable", "Timetable"))
        self.lbl_timetable.setText(_translate(
            "mwindow_timetable", "Timetable"))
        self.btn_edit_timetable.setText(_translate(
```

```
            "mwindow_timetable", "Edit Timetable Slot"))
        item = self.table_widget_timetable.verticalHeaderItem(0)
        item.setText(_translate("mwindow_timetable", "Period 1"))
        item = self.table_widget_timetable.verticalHeaderItem(1)
        item.setText(_translate("mwindow_timetable", "Period 2"))
        item = self.table_widget_timetable.verticalHeaderItem(2)
        item.setText(_translate("mwindow_timetable", "Period 3"))
        item = self.table_widget_timetable.verticalHeaderItem(3)
        item.setText(_translate("mwindow_timetable", "Period 4"))
        item = self.table_widget_timetable.verticalHeaderItem(4)
        item.setText(_translate("mwindow_timetable", "Period 5"))
        item = self.table_widget_timetable.horizontalHeaderItem(0)
        item.setText(_translate("mwindow_timetable", "Monday"))
        item = self.table_widget_timetable.horizontalHeaderItem(1)
        item.setText(_translate("mwindow_timetable", "Tuesday"))
        item = self.table_widget_timetable.horizontalHeaderItem(2)
        item.setText(_translate("mwindow_timetable", "Wednesday"))
        item = self.table_widget_timetable.horizontalHeaderItem(3)
        item.setText(_translate("mwindow_timetable", "Thursday"))
        item = self.table_widget_timetable.horizontalHeaderItem(4)
        item.setText(_translate("mwindow_timetable", "Friday"))
        __sortingEnabled = self.table_widget_timetable.isSortingEnabled()
        self.table_widget_timetable.setSortingEnabled(False)
        self.table_widget_timetable.setSortingEnabled(__sortingEnabled)
        self.menu_my_subjects.setTitle(
            _translate("mwindow_timetable", "My Subjects"))
        self.menu_agenda.setTitle(_translate("mwindow_timetable", "Agenda"))
        self.menu_timetable.setTitle(
            _translate("mwindow_timetable", "Timetable"))
        self.actionAgenda.setText(_translate("mwindow_timetable", "Agenda"))

```

Next, I needed to create the dialog for editing a timetable slot. This was a relatively simple process, as I was able to take the user interface from *add_task.ui* and adapt it to editing a timetable, due to the two dialogs achieving a similar purpose.

The user is able to select the subject through a combo box, like in Agenda. Meanwhile, the user can optionally enter a teacher and a room using the line edit widgets. These will have character limits of 30 characters and 20 characters respectively.



I was happy with the design of the new dialog window, so I used *pyuic5* to generate the Python code for this user interface, and saved it as *edit_timetable_setup.py*.

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'edit_timetable.ui'
#
# Created by: PyQt5 UI code generator 5.12.1
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_dialog_edit_timetable(object):
    def setupUi(self, dialog_edit_timetable):
        dialog_edit_timetable.setObjectName("dialog_edit_timetable")
        dialog_edit_timetable.resize(800, 600)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        dialog_edit_timetable.setFont(font)
        self.layoutWidget = QtWidgets.QWidget(dialog_edit_timetable)
        self.layoutWidget.setGeometry(QtCore.QRect(11, 11, 716, 208))
        self.layoutWidget.setObjectName("layoutWidget")
        self.vert_layout_edit_timetable = QtWidgets.QVBoxLayout(
            self.layoutWidget)
        self.vert_layout_edit_timetable.setContentsMargins(0, 0, 0, 0)
        self.vert_layout_edit_timetable.setObjectName(
            "vert_layout_edit_timetable")
        self.lbl_edit_timetable = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setPointSize(14)
        self.lbl_edit_timetable.setFont(font)
        self.lbl_edit_timetable.setObjectName("lbl_edit_timetable")
        self.vert_layout_edit_timetable.addWidget(self.lbl_edit_timetable)
        self.hori_line_edit_timetable = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_edit_timetable.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_edit_timetable.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_edit_timetable.setObjectName("hori_line_edit_timetable")
        self.vert_layout_edit_timetable.addWidget(
```

```
            self.hori_line_edit_timetable)
        self.form_layout_edit_timetable = QtWidgets.QFormLayout()
        self.form_layout_edit_timetable.setObjectName(
            "form_layout_edit_timetable")
        self.lbl_subject = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_subject.setObjectName("lbl_subject")
        self.form_layout_edit_timetable.setWidget(
            0, QtWidgets.QFormLayout.LabelRole, self.lbl_subject)
        self.comb_box_subject = QtWidgets.QComboBox(self.layoutWidget)
        self.comb_box_subject.setFrame(True)
        self.comb_box_subject.setObjectName("comb_box_subject")
        self.form_layout_edit_timetable.setWidget(
            0, QtWidgets.QFormLayout.FieldRole, self.comb_box_subject)
        self.lbl_teacher = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_teacher.setObjectName("lbl_teacher")
        self.form_layout_edit_timetable.setWidget(
            1, QtWidgets.QFormLayout.LabelRole, self.lbl_teacher)
        self.line_edit_teacher = QtWidgets.QLineEdit(self.layoutWidget)
        self.line_edit_teacher.setObjectName("line_edit_teacher")
        self.form_layout_edit_timetable.setWidget(
            1, QtWidgets.QFormLayout.FieldRole, self.line_edit_teacher)
        self.lbl_room = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_room.setObjectName("lbl_room")
        self.form_layout_edit_timetable.setWidget(
            2, QtWidgets.QFormLayout.LabelRole, self.lbl_room)
        self.line_edit_room = QtWidgets.QLineEdit(self.layoutWidget)
        self.line_edit_room.setObjectName("line_edit_room")
        self.form_layout_edit_timetable.setWidget(
            2, QtWidgets.QFormLayout.FieldRole, self.line_edit_room)
        self.vert_layout_edit_timetable.addLayout(
            self.form_layout_edit_timetable)
        self.hori_line_lesson_details = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_lesson_details.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_lesson_details.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_lesson_details.setObjectName("hori_line_lesson_details")
        self.vert_layout_edit_timetable.addWidget(
            self.hori_line_lesson_details)
        self.lbl_instruction = QtWidgets.QLabel(self.layoutWidget)
```

```
        font = QtGui.QFont()
        font.setItalic(True)
        self.lbl_instruction.setFont(font)
        self.lbl_instruction.setObjectName("lbl_instruction")
        self.vert_layout_edit_timetable.addWidget(self.lbl_instruction)
        self.button_box_edit_timetable = QtWidgets.QDialogButtonBox(
            self.layoutWidget)
        self.button_box_edit_timetable.setOrientation(QtCore.Qt.Horizontal)
        self.button_box_edit_timetable.setStandardButtons(
            QtWidgets.QDialogButtonBox.Cancel | QtWidgets.QDialogButtonBox.Save)
        self.button_box_edit_timetable.setObjectName(
            "button_box_edit_timetable")
        self.vert_layout_edit_timetable.addWidget(
            self.button_box_edit_timetable, 0, QtCore.Qt.AlignLeft)

        self.retranslateUi(dialog_edit_timetable)
        self.button_box_edit_timetable.accepted.connect(
            dialog_edit_timetable.accept)
        self.button_box_edit_timetable.rejected.connect(
            dialog_edit_timetable.reject)
        QtCore.QMetaObject.connectSlotsByName(dialog_edit_timetable)

    def retranslateUi(self, dialog_edit_timetable):
        _translate = QtCore.QCoreApplication.translate
        dialog_edit_timetable.setWindowTitle(
            _translate("dialog_edit_timetable", "Add Task"))
        self.lbl_edit_timetable.setText(_translate(
            "dialog_edit_timetable", "Edit Timetable Slot"))
        self.lbl_subject.setText(_translate(
            "dialog_edit_timetable", "Subject:"))
        self.lbl_teacher.setText(_translate(
            "dialog_edit_timetable", "Teacher:"))
        self.lbl_room.setText(_translate("dialog_edit_timetable", "Room:"))
        self.lbl_instruction.setText(_translate(
            "dialog_edit_timetable", "Please select a subject, and optionally a teacher and room (maximum 30 and 20 characters).")
```

To run the program in *timetable.py*, I imported both of the generated code into the program, and created classes for their windows which inherit from the generated code.

```
# Sets up the Edit Timetable dialog.


class EditTimetableDialog(QDialog, Ui_dialog_edit_timetable):
    def __init__(self):
        super().__init__()
        self.setupUi(self)
```

```
# Sets up the Timetable main window.


class TimetableWindow(QMainWindow, Ui_mwindow_timetable):
    def __init__(self):
        super().__init__()
        self.setupUi(self)
```

When Timetable is initialised, a style sheet is used to make the *QTableWidget* have a transparent background, which helps it to fit into the clean design aesthetic of the entire application.

```
        # Sets the timetable to have a transparent background.
        self.setStyleSheet("""QTableWidget {background-color: transparent;}
            QHeaderView::section {background-color: transparent;}
            QHeaderView {background-color: transparent;}""")
```

I also connected the click of *btn_edit_timetable* to perform the method *open_dialog_edit_timetable()* by using the function *.connect().*

```
        # Connects 'Edit Timetable Slot' button to the Edit Timetable dialog.
        self.btn_edit_timetable.clicked.connect(
            self.open_dialog_edit_timetable)
```

The method *open_dialog_edit_timetable* initialises the dialog window and opens it.

```
    # Opens the dialog for Edit Timetable.
    def open_dialog_edit_timetable(self):
        self.Dialog = EditTimetableDialog()
        self.Dialog.open()
```

These changes cumulated to produce the following code for *timetable.py*.

```python
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QDialog, QMainWindow

from edit_timetable_setup import Ui_dialog_edit_timetable
from timetable_setup import Ui_mwindow_timetable

# Sets up the Edit Timetable dialog.


class EditTimetableDialog(QDialog, Ui_dialog_edit_timetable):
    def __init__(self):
        super().__init__()
        self.setupUi(self)

# Sets up the Timetable main window.


class TimetableWindow(QMainWindow, Ui_mwindow_timetable):
    def __init__(self):
        super().__init__()
        self.setupUi(self)

        # Sets the timetable to have a transparent background.
        self.setStyleSheet("""QTableWidget {background-color: transparent;}
            QHeaderView::section {background-color: transparent;}
            QHeaderView {background-color: transparent;}""")

        # Connects 'Edit Timetable Slot' button to the Edit Timetable dialog.
        self.btn_edit_timetable.clicked.connect(
            self.open_dialog_edit_timetable)

    # Opens the dialog for Edit Timetable.
    def open_dialog_edit_timetable(self):
        self.Dialog = EditTimetableDialog()
        self.Dialog.open()

```

```python
# Opens the main window when the program is executed.
if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    mwindow_timetable = TimetableWindow()
    mwindow_timetable.show()
    sys.exit(app.exec_())

```

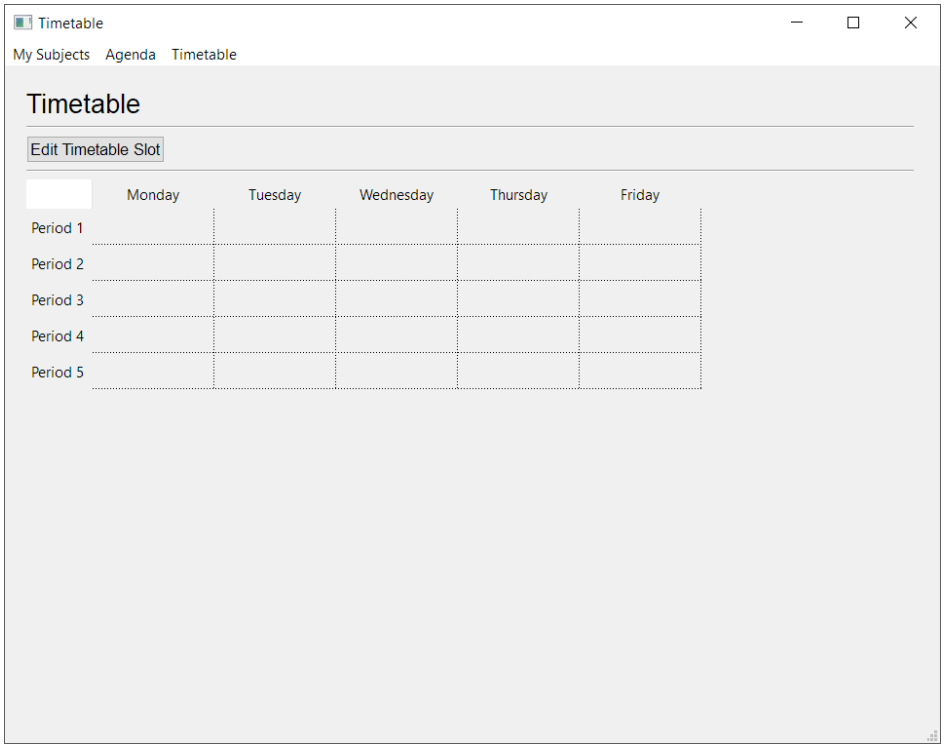
Testing: Iteration #1 – User Interface

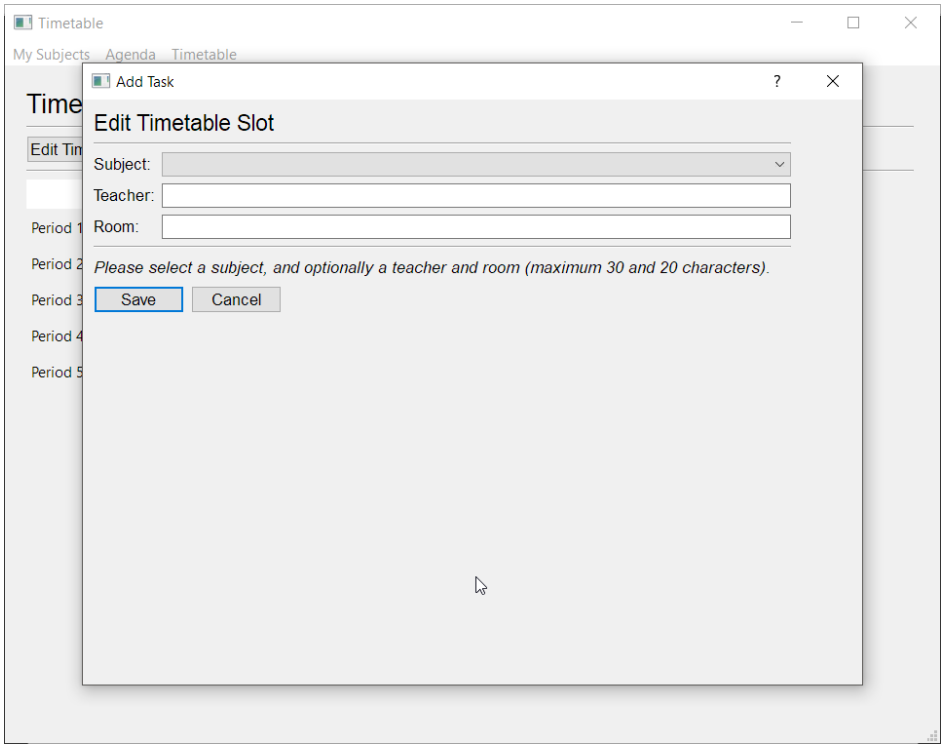
Although there was only basic functionality in the program at this point, it was important to test the changes I made, as the user interface is the basis for which the program is run upon.

I started by testing the small number of inputs possible for the program.

Input	Nature of Data	Data Validation	Is Data Accepted?	Output
Pressed the 'Edit Timetable Slot' button on the main window.	Normal	Button pressed	Yes	The dialog for editing a timetable slot was opened.
Pressed the dropdown menu for 'Subject' in the dialog window.	Normal	Button pressed	Yes	No subjects were shown as options in the dropdown menu; it was empty.
Pressed the 'Save' button in the dialog window.	Normal	Button pressed	Yes	The dialog for editing a timetable slot was closed.
Pressed the 'Cancel' button in the dialog window.	Normal	Button pressed	Yes	The dialog for editing a timetable slot was closed.

Screenshots for the beta testing process of inputs can be seen below.





During the testing process, the program was working as intended. Although I do not want the dialog window to close straight away when the 'Save' button is pressed, this was expected to occur, as I did not disconnect the button from performing the method which closes it.

This will be performed in the next development iteration, where I will be developing the feature which allows the user to edit timetable slots and save it to a JSON file.

No options were displayed when the dropdown menu was pressed because I had not coded the program to get the subjects from *subject_list.txt* yet. This will also be added in the next development iteration.

I moved on and performed a general test for this development iteration, which compares the program's functionality to the objectives.

Test	Expected Result	Outcome	Further Actions
Is the user able to view their timetable in a 5 x 5 table format?	The user should be able to view their timetable in a 5 x 5 table format, with the vertical headers showing the periods, and the horizontal headers showing the days of the week.	The user is able to view their timetable in a 5 x 5 table format, with the vertical headers showing the periods, and the horizontal headers showing the days of the week.	N/A.
Is the user able to edit timetable slots with a subject, teacher, and room?	The user should be able to edit the timetable slots with a subject, teacher, and room. Once the lesson details have been validated, the timetable slot should be updated to display these details.	The user is able to edit the timetable slots with a teacher and room, but the options from the dropdown menu for subject are not loaded in. However, no validation for the entered data is performed, and the timetable slot is not updated to display these details.	Add the same method used in Agenda to populate the combo box options for the user to select. Use methods to get the row and column of the user's selected subject, validate the lesson details, and update the selected timetable slot with these details.

The general tests showed that my program has achieved the first objective to provide a way for the user to view their timetable in a table format, but more development needs to be done to fulfil the other objectives which enable the user to interact with the timetable.

Outside of the more formal general tests, I also noticed that the window title for Edit Timetable Slot was still set to 'Add Task'. This will be addressed in the next development iteration.

<u>Development: Iteration #2 – Editing Slots</u>

To start this development iteration, I addressed the window title for the Edit Timetable Slot dialog being set as 'Add Task'. I changed it to 'Edit Timetable Slot' in Qt Creator.

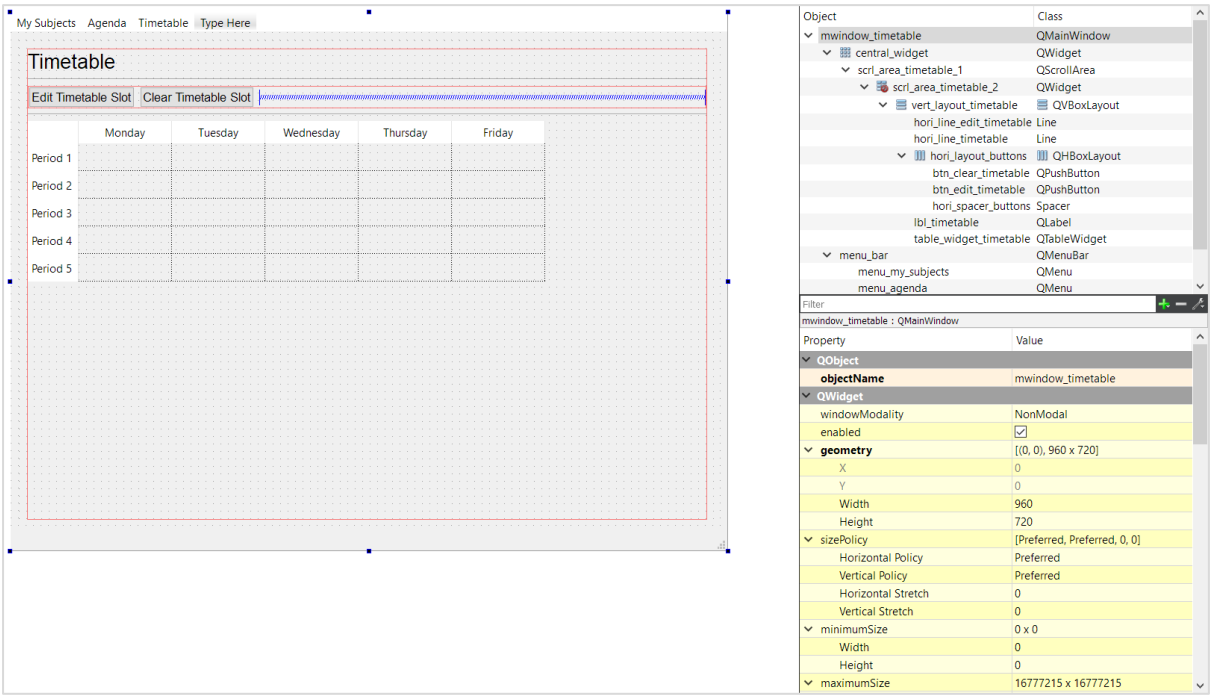
After this change, I updated *edit_timetable_slot_setup.py* by generating the code using *pyuic5*.

```python
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'edit_timetable.ui'
#
# Created by: PyQt5 UI code generator 5.12.1
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_dialog_edit_timetable(object):
    def setupUi(self, dialog_edit_timetable):
        dialog_edit_timetable.setObjectName("dialog_edit_timetable")
        dialog_edit_timetable.resize(800, 600)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        dialog_edit_timetable.setFont(font)
        self.layoutWidget = QtWidgets.QWidget(dialog_edit_timetable)
        self.layoutWidget.setGeometry(QtCore.QRect(11, 11, 716, 208))
        self.layoutWidget.setObjectName("layoutWidget")
        self.vert_layout_edit_timetable = QtWidgets.QVBoxLayout(
            self.layoutWidget)
        self.vert_layout_edit_timetable.setContentsMargins(0, 0, 0, 0)
        self.vert_layout_edit_timetable.setObjectName(
            "vert_layout_edit_timetable")
        self.lbl_edit_timetable = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setPointSize(14)
        self.lbl_edit_timetable.setFont(font)
        self.lbl_edit_timetable.setObjectName("lbl_edit_timetable")
        self.vert_layout_edit_timetable.addWidget(self.lbl_edit_timetable)
        self.hori_line_edit_timetable = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_edit_timetable.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_edit_timetable.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_edit_timetable.setObjectName("hori_line_edit_timetable")
        self.vert_layout_edit_timetable.addWidget(
            self.hori_line_edit_timetable)
        self.form_layout_edit_timetable = QtWidgets.QFormLayout()
        self.form_layout_edit_timetable.setObjectName(
            "form_layout_edit_timetable")
        self.lbl_subject = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_subject.setObjectName("lbl_subject")
        self.form_layout_edit_timetable.setWidget(
            0, QtWidgets.QFormLayout.LabelRole, self.lbl_subject)
        self.comb_box_subject = QtWidgets.QComboBox(self.layoutWidget)
        self.comb_box_subject.setFrame(True)
        self.comb_box_subject.setObjectName("comb_box_subject")
        self.form_layout_edit_timetable.setWidget(
            0, QtWidgets.QFormLayout.FieldRole, self.comb_box_subject)
        self.lbl_teacher = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_teacher.setObjectName("lbl_teacher")
        self.form_layout_edit_timetable.setWidget(
            1, QtWidgets.QFormLayout.LabelRole, self.lbl_teacher)
        self.line_edit_teacher = QtWidgets.QLineEdit(self.layoutWidget)
        self.line_edit_teacher.setObjectName("line_edit_teacher")
        self.form_layout_edit_timetable.setWidget(
            1, QtWidgets.QFormLayout.FieldRole, self.line_edit_teacher)
        self.lbl_room = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_room.setObjectName("lbl_room")
        self.form_layout_edit_timetable.setWidget(
            2, QtWidgets.QFormLayout.LabelRole, self.lbl_room)
        self.line_edit_room = QtWidgets.QLineEdit(self.layoutWidget)
        self.line_edit_room.setObjectName("line_edit_room")
        self.form_layout_edit_timetable.setWidget(
            2, QtWidgets.QFormLayout.FieldRole, self.line_edit_room)
        self.vert_layout_edit_timetable.addLayout(
            self.form_layout_edit_timetable)
        self.hori_line_lesson_details = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_lesson_details.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_lesson_details.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_lesson_details.setObjectName("hori_line_lesson_details")
        self.vert_layout_edit_timetable.addWidget(
            self.hori_line_lesson_details)
        self.lbl_instruction = QtWidgets.QLabel(self.layoutWidget)
```

```
        font = QtGui.QFont()
        font.setItalic(True)
        self.lbl_instruction.setFont(font)
        self.lbl_instruction.setObjectName("lbl_instruction")
        self.vert_layout_edit_timetable.addWidget(self.lbl_instruction)
        self.button_box_edit_timetable = QtWidgets.QDialogButtonBox(
            self.layoutWidget)
        self.button_box_edit_timetable.setOrientation(QtCore.Qt.Horizontal)
        self.button_box_edit_timetable.setStandardButtons(
            QtWidgets.QDialogButtonBox.Cancel | QtWidgets.QDialogButtonBox.Save)
        self.button_box_edit_timetable.setObjectName(
            "button_box_edit_timetable")
        self.vert_layout_edit_timetable.addWidget(
            self.button_box_edit_timetable, 0, QtCore.Qt.AlignLeft)

        self.retranslateUi(dialog_edit_timetable)
        self.button_box_edit_timetable.accepted.connect(
            dialog_edit_timetable.accept)
        self.button_box_edit_timetable.rejected.connect(
            dialog_edit_timetable.reject)
        QtCore.QMetaObject.connectSlotsByName(dialog_edit_timetable)

    def retranslateUi(self, dialog_edit_timetable):
        _translate = QtCore.QCoreApplication.translate
        dialog_edit_timetable.setWindowTitle(_translate(
            "dialog_edit_timetable", "Edit Timetable Slot"))
        self.lbl_edit_timetable.setText(_translate(
            "dialog_edit_timetable", "Edit Timetable Slot"))
        self.lbl_subject.setText(_translate(
            "dialog_edit_timetable", "Subject:"))
        self.lbl_teacher.setText(_translate(
            "dialog_edit_timetable", "Teacher:"))
        self.lbl_room.setText(_translate("dialog_edit_timetable", "Room:"))
        self.lbl_instruction.setText(_translate(
            "dialog_edit_timetable", "Please select a subject, and optionally a teacher and room (maximum 30 and 20 characters).")
```

Next, I added a new button to clear a timetable slot, which will enable users to empty a timetable slot from any lessons if they have made a mistake, or if they no longer have lessons at that time.

To do this in Qt Creator, I added a horizontal layout (*hori_layout_buttons*), and placed the existing *btn_edit_timetable* button inside it. Then, I added a new button and named this new button *btn_clear_timetable*, and put it on the right of *btn_edit_timetable.* To align these buttons to the left, I placed a horizontal spacer, *hori_spacer_buttons*, on the far right side of these two buttons.



I generated the updated code for this user interface through *pyuic5*, which produced the following code for *timetable_setup.py*.

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'timetable.ui'
#
# Created by: PyQt5 UI code generator 5.12.1
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_mwindow_timetable(object):
    def setupUi(self, mwindow_timetable):
        mwindow_timetable.setObjectName("mwindow_timetable")
        mwindow_timetable.resize(960, 720)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        font.setKerning(True)
        mwindow_timetable.setFont(font)
        self.central_widget = QtWidgets.QWidget(mwindow_timetable)
        self.central_widget.setObjectName("central_widget")
        self.gridLayout = QtWidgets.QGridLayout(self.central_widget)
        self.gridLayout.setObjectName("gridLayout")
        self.scrl_area_timetable_1 = QtWidgets.QScrollArea(self.central_widget)
        self.scrl_area_timetable_1.setFrameShape(QtWidgets.QFrame.NoFrame)
        self.scrl_area_timetable_1.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.scrl_area_timetable_1.setLineWidth(0)
        self.scrl_area_timetable_1.setWidgetResizable(True)
        self.scrl_area_timetable_1.setObjectName("scrl_area_timetable_1")
        self.scrl_area_timetable_2 = QtWidgets.QWidget()
        self.scrl_area_timetable_2.setGeometry(QtCore.QRect(0, 0, 938, 647))
        font = QtGui.QFont()
        font.setKerning(True)
        self.scrl_area_timetable_2.setFont(font)
        self.scrl_area_timetable_2.setAutoFillBackground(True)
        self.scrl_area_timetable_2.setObjectName("scrl_area_timetable_2")
        self.layoutWidget = QtWidgets.QWidget(self.scrl_area_timetable_2)
        self.layoutWidget.setGeometry(QtCore.QRect(11, 11, 911, 631))
        self.layoutWidget.setObjectName("layoutWidget")
        self.vert_layout_timetable = QtWidgets.QVBoxLayout(self.layoutWidget)
        self.vert_layout_timetable.setContentsMargins(0, 0, 0, 0)
        self.vert_layout_timetable.setObjectName("vert_layout_timetable")
        self.lbl_timetable = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setPointSize(16)
        self.lbl_timetable.setFont(font)
        self.lbl_timetable.setObjectName("lbl_timetable")
        self.vert_layout_timetable.addWidget(self.lbl_timetable)
        self.hori_line_timetable = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_timetable.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_timetable.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_timetable.setObjectName("hori_line_timetable")
        self.vert_layout_timetable.addWidget(self.hori_line_timetable)
        self.hori_layout_buttons = QtWidgets.QHBoxLayout()
        self.hori_layout_buttons.setObjectName("hori_layout_buttons")
        self.btn_edit_timetable = QtWidgets.QPushButton(self.layoutWidget)
        self.btn_edit_timetable.setObjectName("btn_edit_timetable")
        self.hori_layout_buttons.addWidget(
            self.btn_edit_timetable, 0, QtCore.Qt.AlignLeft)
        self.btn_clear_timetable = QtWidgets.QPushButton(self.layoutWidget)
        self.btn_clear_timetable.setObjectName("btn_clear_timetable")
        self.hori_layout_buttons.addWidget(
            self.btn_clear_timetable, 0, QtCore.Qt.AlignLeft)
        spacerItem = QtWidgets.QSpacerItem(
            40, 20, QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Minimum)
        self.hori_layout_buttons.addItem(spacerItem)
        self.vert_layout_timetable.addLayout(self.hori_layout_buttons)
        self.hori_line_edit_timetable = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_edit_timetable.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_edit_timetable.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_edit_timetable.setObjectName("hori_line_edit_timetable")
        self.vert_layout_timetable.addWidget(self.hori_line_edit_timetable)
        self.table_widget_timetable = QtWidgets.QTableWidget(self.layoutWidget)
        font = QtGui.QFont()
        font.setFamily("Arial")
```

```
        font.setPointSize(10)
        self.table_widget_timetable.setFont(font)
        self.table_widget_timetable.setStyleSheet(
            "background-color: transparent")
        self.table_widget_timetable.setFrameShape(QtWidgets.QFrame.NoFrame)
        self.table_widget_timetable.setEditTriggers(
            QtWidgets.QAbstractItemView.NoEditTriggers)
        self.table_widget_timetable.setTabKeyNavigation(False)
        self.table_widget_timetable.setProperty("showDropIndicator", True)
        self.table_widget_timetable.setSelectionMode(
            QtWidgets.QAbstractItemView.SingleSelection)
        self.table_widget_timetable.setShowGrid(True)
        self.table_widget_timetable.setGridStyle(QtCore.Qt.DotLine)
        self.table_widget_timetable.setCornerButtonEnabled(False)
        self.table_widget_timetable.setObjectName("table_widget_timetable")
        self.table_widget_timetable.setColumnCount(5)
        self.table_widget_timetable.setRowCount(5)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_timetable.setVerticalHeaderItem(0, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_timetable.setVerticalHeaderItem(1, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_timetable.setVerticalHeaderItem(2, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_timetable.setVerticalHeaderItem(3, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_timetable.setVerticalHeaderItem(4, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_timetable.setHorizontalHeaderItem(0, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_timetable.setHorizontalHeaderItem(1, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_timetable.setHorizontalHeaderItem(2, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_timetable.setHorizontalHeaderItem(3, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_timetable.setHorizontalHeaderItem(4, item)
        item = QtWidgets.QTableWidgetItem()
```

```
        self.table_widget_timetable.setItem(0, 0, item)
        self.table_widget_timetable.horizontalHeader().setHighlightSections(True)
        self.table_widget_timetable.horizontalHeader().setStretchLastSection(False)
        self.vert_layout_timetable.addWidget(self.table_widget_timetable)
        self.scrl_area_timetable_1.setWidget(self.scrl_area_timetable_2)
        self.gridLayout.addWidget(self.scrl_area_timetable_1, 0, 0, 1, 1)
        mwindow_timetable.setCentralWidget(self.central_widget)
        self.menu_bar = QtWidgets.QMenuBar(mwindow_timetable)
        self.menu_bar.setGeometry(QtCore.QRect(0, 0, 960, 26))
        self.menu_bar.setObjectName("menu_bar")
        self.menu_my_subjects = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_my_subjects.setFont(font)
        self.menu_my_subjects.setObjectName("menu_my_subjects")
        self.menu_agenda = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_agenda.setFont(font)
        self.menu_agenda.setObjectName("menu_agenda")
        self.menu_timetable = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_timetable.setFont(font)
        self.menu_timetable.setObjectName("menu_timetable")
        mwindow_timetable.setMenuBar(self.menu_bar)
        self.status_bar = QtWidgets.QStatusBar(mwindow_timetable)
        self.status_bar.setObjectName("status_bar")
        mwindow_timetable.setStatusBar(self.status_bar)
        self.actionAgenda = QtWidgets.QAction(mwindow_timetable)
        self.actionAgenda.setObjectName("actionAgenda")
        self.menu_bar.addAction(self.menu_my_subjects.menuAction())
        self.menu_bar.addAction(self.menu_agenda.menuAction())
        self.menu_bar.addAction(self.menu_timetable.menuAction())

```

```
        self.retranslateUi(mwindow_timetable)
        QtCore.QMetaObject.connectSlotsByName(mwindow_timetable)

    def retranslateUi(self, mwindow_timetable):
        _translate = QtCore.QCoreApplication.translate
        mwindow_timetable.setWindowTitle(
            _translate("mwindow_timetable", "Timetable"))
        self.lbl_timetable.setText(_translate(
            "mwindow_timetable", "Timetable"))
        self.btn_edit_timetable.setText(_translate(
            "mwindow_timetable", "Edit Timetable Slot"))
        self.btn_clear_timetable.setText(_translate(
            "mwindow_timetable", "Clear Timetable Slot"))
        item = self.table_widget_timetable.verticalHeaderItem(0)
        item.setText(_translate("mwindow_timetable", "Period 1"))
        item = self.table_widget_timetable.verticalHeaderItem(1)
        item.setText(_translate("mwindow_timetable", "Period 2"))
        item = self.table_widget_timetable.verticalHeaderItem(2)
        item.setText(_translate("mwindow_timetable", "Period 3"))
        item = self.table_widget_timetable.verticalHeaderItem(3)
        item.setText(_translate("mwindow_timetable", "Period 4"))
        item = self.table_widget_timetable.verticalHeaderItem(4)
        item.setText(_translate("mwindow_timetable", "Period 5"))
        item = self.table_widget_timetable.horizontalHeaderItem(0)
        item.setText(_translate("mwindow_timetable", "Monday"))
        item = self.table_widget_timetable.horizontalHeaderItem(1)
        item.setText(_translate("mwindow_timetable", "Tuesday"))
        item = self.table_widget_timetable.horizontalHeaderItem(2)
        item.setText(_translate("mwindow_timetable", "Wednesday"))
        item = self.table_widget_timetable.horizontalHeaderItem(3)
        item.setText(_translate("mwindow_timetable", "Thursday"))
        item = self.table_widget_timetable.horizontalHeaderItem(4)
        item.setText(_translate("mwindow_timetable", "Friday"))
        __sortingEnabled = self.table_widget_timetable.isSortingEnabled()
        self.table_widget_timetable.setSortingEnabled(False)
        self.table_widget_timetable.setSortingEnabled(__sortingEnabled)
        self.menu_my_subjects.setTitle(
            _translate("mwindow_timetable", "My Subjects"))
```

```
        self.menu_agenda.setTitle(_translate("mwindow_timetable", "Agenda"))
        self.menu_timetable.setTitle(
            _translate("mwindow_timetable", "Timetable"))
        self.actionAgenda.setText(_translate("mwindow_timetable", "Agenda"))

```

At the start of the program, I created a block of code to read existing JSON files for the list of tasks, so that the timetable can be loaded into the *QTableWidget* if it was used before.

It reads the file *timetable.json*, and if there is something inside that file, it will load it to the list variable, *timetable*. Otherwise, it will print ‘Empty JSON file’ to the console for debugging purposes, and then define *timetable* as a list variable.

After reading the file, the program checks for any missing lessons in the list variable *timetable*. It should have 25 lessons, as there are 25 (5 x 5) lessons in the fixed timetable template, so it uses *len()* to check if there are less than 25 lessons in the list.

If there are any missing lessons, the program will fill them in with an empty lesson by appending empty lessons at the end of the list with *.append()*. This will prevent missing index errors later on the start-up of the program when the timetable list is read and added onto the *QTableWidget*.

```
    # Adds empty dictionary values to missing indexes in the list.
    missing_lessons = 25 - len(timetable)
    for index in range(missing_lessons):
        lesson = {"subject": " ",
                  "teacher": " ",
                  "room": " "}
        timetable.append(lesson)
```

Next, I connected the new button, *btn_clear_timetable*, to a new method which will carry out its function, *clear_timetable_slot()*. The program will connect this button whenever the main window of Timetable is initialised.

```
        # Connects 'Clear Timetable Slot' button to clear the slot.
        self.btn_clear_timetable.clicked.connect(self.clear_timetable_slot)
```

I also updated the style sheet to remove the white background of the disabled corner button in the *QTableWidget* by making the background colour transparent.

```
        # Sets the timetable to have a transparent background.
        self.setStyleSheet("""QTableWidget {background-color: transparent;}
            QHeaderView::section {background-color: transparent;}
            QHeaderView {background-color: transparent;}
            QTableCornerButton::section{background-color: transparent;}""")
```

As mentioned in the testing iteration #2, I also needed to disconnect the 'Save' button from the built-in functions so length validation on the line edit inputs for the teacher and room can be performed. Then, the button is connected to the method *save_lesson()*, which will save the lesson to the selected timetable slot.

I placed this in the method *open_dialog_edit_timetable()*, as the button is part of the dialog window for editing a timetable slot.

```
        # Connects 'Save' button to save the lesson to the timetable slot.
        self.Dialog.button_box_edit_timetable.accepted.disconnect()
        self.Dialog.button_box_edit_timetable.accepted.connect(
            self.save_lesson)
```

Whenever the main window for Timetable is initialised, I coded it to call the method *update_timetable()*, as this will populate the *QTableWidget* with the lessons in the *timetable* list.

The following code opens the main window when the program is executed by using the *.show()* function. It is responsible for the start-up of *timetable.py*, as the user interface for the program would not start without it.

```
# Opens the main window when the program is executed.
if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    mwindow_timetable = TimetableWindow()
    mwindow_timetable.show()
    sys.exit(app.exec_())

```

In the method *clear_timetable_slot()*, the program gets the row and column which the user selected and assigns them to the variables *selected_row* and *selected_column*. Then, it

creates the *lesson* dictionary variable to have empty lesson details, and overwrites the selected timetable slot with those empty lesson details.

The program is able to work out which index to use by multiplying the *selected_row* by 5 (as a new row is started at every 5th index), and adding this to the *selected_column*.

At the end, the method calls for another method, *save_timetable_list()*, to save the updated timetable list to *timetable.json*.

```
126     # Clears the lesson from the selected timetable slot.
127     def clear_timetable_slot(self):
128         # Gets the row and column of the selected timetable slot.
129         selected_row = self.table_widget_timetable.currentRow()
130         selected_column = self.table_widget_timetable.currentColumn()
131 
132         # Removes the lesson details from the timetable slot.
133         lesson = {"subject": "",
134                   "teacher": "",
135                   "room": ""}
136         index = (selected_row * 5) + selected_column
137         timetable[index] = lesson
138 
139         self.save_timetable_list()
```

The method *save_lesson()* is called when the user presses the ‘Save’ button on the dialog to edit a timetable slot.

It starts by getting the row and column of the timetable slot which the user selected to edit.

Then, it gets the selected subject by using *.currentText()* on the combo box, and the selected teacher and room by using *.text()* on their respective line edit widgets.

The method proceeds to perform length check validations on the user’s inputs for the teacher and room for that lesson.

If the input for the teacher name exceeds 30 characters, their lesson input is rejected. The instruction label text is changed to specify that their teacher input exceeds 30 characters, and to try again.

Likewise, if the input for the room exceeds 20 characters, their lesson input is rejected. The instruction label text is changed to specify that their room input exceeds 20 characters, and to try again.

No presence checks are performed on either of these inputs, as the user may not want to input a teacher or room. For example, they may have different teachers on different weeks, or the lesson may be held in a different room depending on the week.

If neither of these conditions are triggered, it means the inputs passed the length validation checks. The *lesson* dictionary is created to include the subject, teacher, and room which the user input. Then, these lesson details are written to the appropriate index of the *timetable* list variable.

At the end, the method *save_timetable_list()* is called, so that the program updates the JSON file with the latest timetable list, and the dialog window is closed.

```
    # Saves the lesson to the selected timetable slot.
    def save_lesson(self):
        # Gets the row and column of the selected timetable slot.
        selected_row = self.table_widget_timetable.currentRow()
        selected_column = self.table_widget_timetable.currentColumn()

        # Gets the user inputs for subject, teacher, and room.
        lesson_subject = self.Dialog.comb_box_subject.currentText()
        lesson_teacher = self.Dialog.line_edit_teacher.text()
        lesson_room = self.Dialog.line_edit_room.text()

        # Saves lesson to selected timetable slot if length validations passed.
        if len(lesson_teacher) > 30:
            self.Dialog.lbl_instruction.setText(
                "Your teacher input exceeds 30 characters. Please try again.")
        elif len(lesson_room) > 20:
            self.Dialog.lbl_instruction.setText(
                "Your room input exceeds 30 characters. Please try again.")
        else:
            lesson = {"subject": lesson_subject,
                      "teacher": lesson_teacher,
                      "room": lesson_room}
            index = (selected_row * 5) + selected_column
            timetable[index] = lesson

            self.save_timetable_list()
            self.Dialog.close()
```

The method *save_timetable_list()* is a simpler one. It saves the timetable list stored in the variable *timetable* to *timetable.json* using *json.dump()*.

Then, it updates the timetable by calling the method *update_timetable()*. This is called because whenever *save_timetable_list()* is called, it is because a change in the timetable has been made, meaning it would be useful to update the timetable to display the latest version of the timetable to the user.

```
    # Updates the JSON file with the current timetable list.
    def save_timetable_list(self):
        with open('timetable.json', 'w') as outfile:
            json.dump(timetable, outfile, ensure_ascii=False, indent=4)

        self.update_timetable()
```

The method *update_timetable()* starts by resizing the columns and rows to fit the contents of the table by using *ResizeToContents* with *.setSectionResizeMode*. This is important in this program especially, as each cell will contain three lines of text rather than just one.

Then, it iterates through every fifth index in the *timetable* list using *timetable[::5]*, as these will iterate through each row in the timetable using a *for* loop. *enumerate()* was used to count the row being iterated.

A nested *for* loop is used to iterate through each of the five columns of each row, with *enumerate()* used again to count the column being iterated.

In this nested loop, the program creates the dictionary variable *lesson_details*, which assigns the values for the subject, teacher, and room according to the *timetable* list, with new lines separating each value. These *lesson_details* are assigned as the contents of each item/cell in the timetable which is iterated through.

```
72      # Populates the timetable with all lessons.
73      def update_timetable(self):
74          # Resizes columns and rows to fit the contents.
75          self.table_widget_timetable.horizontalHeader().setSectionResizeMode(
76              QtWidgets.QHeaderView.ResizeToContents)
77          self.table_widget_timetable.verticalHeader().setSectionResizeMode(
78              QtWidgets.QHeaderView.ResizeToContents)
79
80          # Adds lesson details to each cell in the timetable.
81          for row_index, row in enumerate(timetable[::5]):
82              for column_index, column in enumerate(range(5)):
83                  index = (row_index * 5) + column_index
84                  lesson_details = ((timetable[index]["subject"]) + "\n" +
85                                    (timetable[index]["teacher"]) + "\n" +
86                                    (timetable[index]["room"]))
87                  self.table_widget_timetable.setItem(
88                      row_index, column_index,
89                      QtWidgets.QTableWidgetItem(lesson_details))
```

Finally, the method *open_dialog_edit_timetable()* is called whenever the user presses *btn_edit_timetable*. It defines *self.Dialog* as *EditTimetableDialog()* so that it can inherit the user interface widgets from *timetable_setup.py* when it is opened.

It connects the ‘Save’ button in the dialog window to the *save_lesson()* method, as aforementioned.

Then, it populates the combo box by reading each line of *subject_list.txt* and adding the text on that line as an item of the combo box for the user to choose from when they use the dropdown menu.

At the end, it opens the dialog so that the user can edit their selected timetable slot.

```
55      # Opens the dialog for Edit Timetable.
56      def open_dialog_edit_timetable(self):
57          self.Dialog = EditTimetableDialog()
58
59          # Connects 'Save' button to save the lesson to the timetable slot.
60          self.Dialog.button_box_edit_timetable.accepted.disconnect()
61          self.Dialog.button_box_edit_timetable.accepted.connect(
62              self.save_lesson)
63
64          # Populates the combo box with subject options.
65          with open("subject_list.txt", "r") as data_file:
66              subject_list = data_file.readlines()
67              for line in subject_list:
68                  self.Dialog.comb_box_subject.addItem(line.strip("\n"))
69
70          self.Dialog.open()
```

All of these changes cumulated to produce the following code for *timetable.py*.

```python
import json

from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QDialog, QMainWindow

from edit_timetable_setup import Ui_dialog_edit_timetable
from timetable_setup import Ui_mwindow_timetable

# Reads existing JSON files for list of lessons.
with open('timetable.json', 'r') as outfile:
    try:
        timetable = json.load(outfile)
    except ValueError:
        print("Empty JSON file.")
        timetable = []

    # Adds empty dictionary values to missing indexes in the list.
    missing_lessons = 25 - len(timetable)
    for index in range(missing_lessons):
        lesson = {"subject": " ",
                  "teacher": " ",
                  "room": " "}
        timetable.append(lesson)

# Sets up the Edit Timetable dialog.


class EditTimetableDialog(QDialog, Ui_dialog_edit_timetable):
    def __init__(self):
        super().__init__()
        self.setupUi(self)

# Sets up the Timetable main window.


class TimetableWindow(QMainWindow, Ui_mwindow_timetable):
    def __init__(self):
        super().__init__()
        self.setupUi(self)

        # Sets the timetable to have a transparent background.
        self.setStyleSheet("""QTableWidget {background-color: transparent;}
            QHeaderView::section {background-color: transparent;}
            QHeaderView {background-color: transparent;}
            QTableCornerButton::section{background-color: transparent;}""")

        # Connects 'Edit Timetable Slot' button to the Edit Timetable dialog.
        self.btn_edit_timetable.clicked.connect(
            self.open_dialog_edit_timetable)
        # Connects 'Clear Timetable Slot' button to clear the slot.
        self.btn_clear_timetable.clicked.connect(self.clear_timetable_slot)

        self.update_timetable()

    # Opens the dialog for Edit Timetable.
    def open_dialog_edit_timetable(self):
        self.Dialog = EditTimetableDialog()

        # Connects 'Save' button to save the lesson to the timetable slot.
        self.Dialog.button_box_edit_timetable.accepted.disconnect()
        self.Dialog.button_box_edit_timetable.accepted.connect(
            self.save_lesson)

        # Populates the combo box with subject options.
        with open("subject_list.txt", "r") as data_file:
            subject_list = data_file.readlines()
            for line in subject_list:
                self.Dialog.comb_box_subject.addItem(line.strip("\n"))

        self.Dialog.open()

    # Populates the timetable with all lessons.
    def update_timetable(self):
        # Resizes columns and rows to fit the contents.
        self.table_widget_timetable.horizontalHeader().setSectionResizeMode(
            QtWidgets.QHeaderView.ResizeToContents)
```

```
        self.table_widget_timetable.verticalHeader().setSectionResizeMode(
            QtWidgets.QHeaderView.ResizeToContents)

        # Adds lesson details to each cell in the timetable.
        for row_index, row in enumerate(timetable[::5]):
            for column_index, column in enumerate(range(5)):
                index = (row_index * 5) + column_index
                lesson_details = ((timetable[index]["subject"]) + "\n" +
                                  (timetable[index]["teacher"]) + "\n" +
                                  (timetable[index]["room"]))
                self.table_widget_timetable.setItem(
                    row_index, column_index,
                    QtWidgets.QTableWidgetItem(lesson_details))

    # Updates the JSON file with the current timetable list.
    def save_timetable_list(self):
        with open('timetable.json', 'w') as outfile:
            json.dump(timetable, outfile, ensure_ascii=False, indent=4)

        self.update_timetable()

    # Saves the lesson to the selected timetable slot.
    def save_lesson(self):
        # Gets the row and column of the selected timetable slot.
        selected_row = self.table_widget_timetable.currentRow()
        selected_column = self.table_widget_timetable.currentColumn()

        # Gets the user inputs for subject, teacher, and room.
        lesson_subject = self.Dialog.comb_box_subject.currentText()
        lesson_teacher = self.Dialog.line_edit_teacher.text()
        lesson_room = self.Dialog.line_edit_room.text()

        # Saves lesson to selected timetable slot if length validations passed.
        if len(lesson_teacher) > 30:
            self.Dialog.lbl_instruction.setText(
                "Your teacher input exceeds 30 characters. Please try again.")
        elif len(lesson_room) > 20:
            self.Dialog.lbl_instruction.setText(
                "Your room input exceeds 30 characters. Please try again.")
        else:
            lesson = {"subject": lesson_subject,
                      "teacher": lesson_teacher,
                      "room": lesson_room}
            index = (selected_row * 5) + selected_column
            timetable[index] = lesson

            self.save_timetable_list()
            self.Dialog.close()

    # Clears the lesson from the selected timetable slot.
    def clear_timetable_slot(self):
        # Gets the row and column of the selected timetable slot.
        selected_row = self.table_widget_timetable.currentRow()
        selected_column = self.table_widget_timetable.currentColumn()

        # Removes the lesson details from the timetable slot.
        lesson = {"subject": "",
                  "teacher": "",
                  "room": ""}
        index = (selected_row * 5) + selected_column
        timetable[index] = lesson

        self.save_timetable_list()


# Opens the main window when the program is executed.
if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    mwindow_timetable = TimetableWindow()
    mwindow_timetable.show()
    sys.exit(app.exec_())
```

Testing: Iteration #2 – Editing Slots

Once I completed development for Timetable, I started testing to ensure that the features were implemented properly.

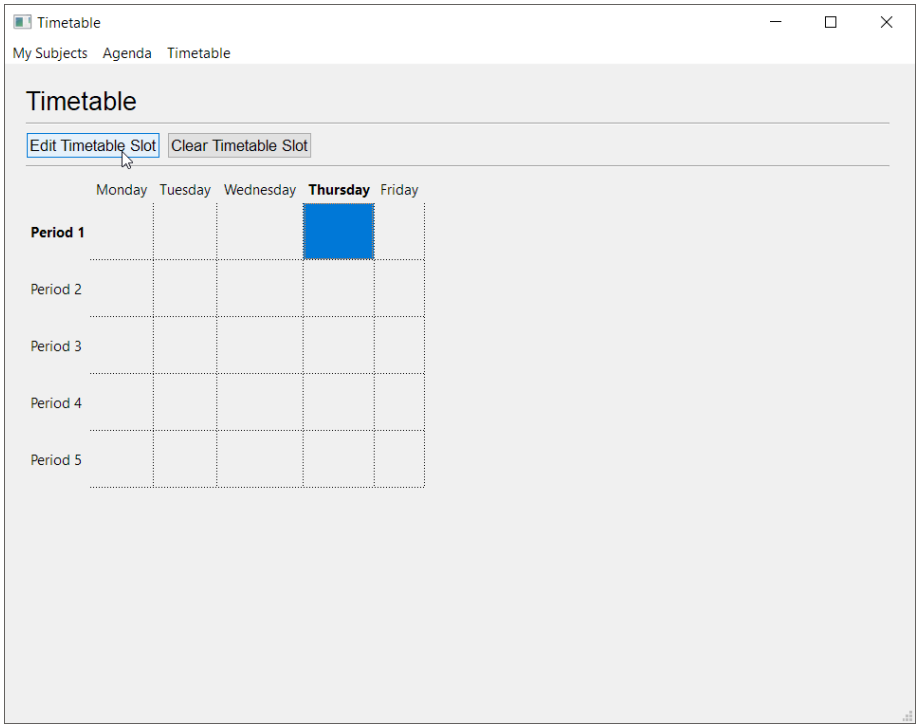
I started by beta testing the inputs for this program.

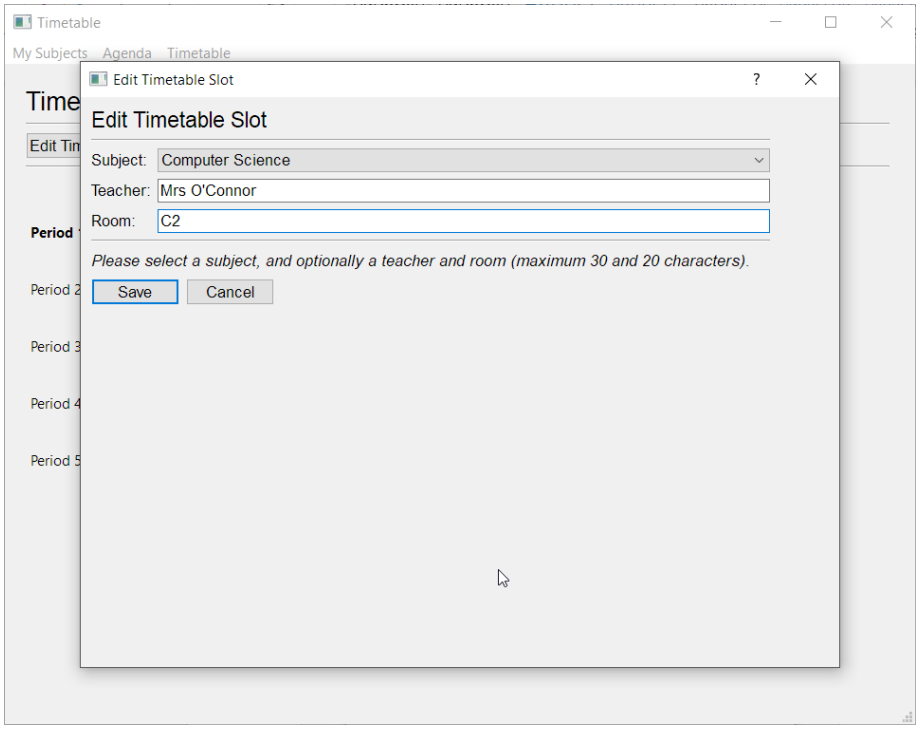
Input	Nature of Data	Data Validation	Is Data Accepted?	Output

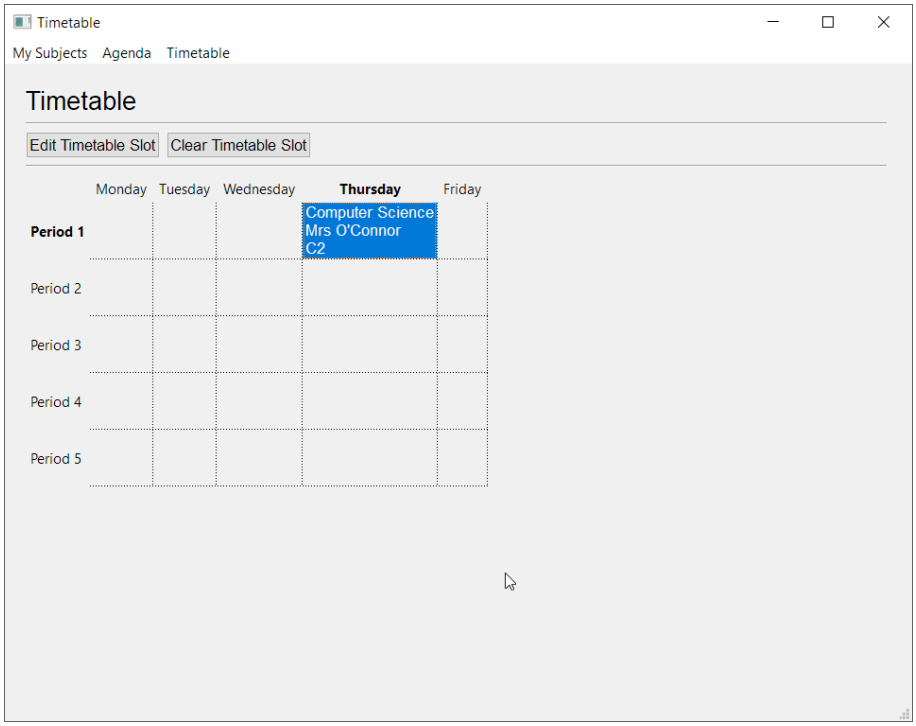
Pressed the dropdown menu for 'Subject' in the dialog window.	Normal	Button pressed	Yes	The subjects from *subject_list.txt* were displayed as options, which were 'Computer Science', 'Further Mathematics', and 'Mathematics'.
Selected the cell for Thursday Period 1 and pressed the 'Edit Timetable Slot' button. Selected 'Computer Science' as the subject, typed 'Mrs O'Connor' as the teacher, and typed 'C2' as the room, then pressed the 'Save' button to save the timetable slot.	Normal	Button pressed, length check	Yes	The dialog window for editing a timetable slot was opened. The timetable slot was successfully edited with the selected subject, teacher, and room.
Selected the cell for Thursday Period 2 and pressed the 'Edit Timetable Slot' button. Selected 'Computer Science' as the subject, and typed 'The School Gymnasium' as the room.	Extreme	Button pressed, length check	Yes	The dialog window for editing a timetable slot was opened. The timetable slot was successfully edited with the selected subject, teacher, and room.
Selected the cell for Thursday Period 2 (the lesson slot with 'The School Gymnasium' as the room) and pressed the 'Clear Timetable Slot' button.	Normal	Button pressed	Yes	The timetable slot was successfully cleared of the existing lesson details.
Selected the cell for Tuesday Period 1 and pressed the 'Edit Timetable Slot'. Selected 'Mathematics' as the subject, entered 'Mr Mason' as the teacher, and entered 'G15' as the room, then pressed the 'Save' button to save the timetable slot.	Normal	Button pressed, length check	Yes	The dialog window for editing a timetable slot was opened. The timetable slot was successfully edited with the selected subject, teacher, and room.
Selected the cell for Friday Period 4 and pressed the 'Edit Timetable Slot'. Selected 'Further Mathematics' as the subject, entered 'Mrs Coldwell' as the teacher, and entered 'G15' as the room, then pressed the	Normal	Button pressed, length check	Yes	The dialog window for editing a timetable slot was opened. The timetable slot was successfully edited with the

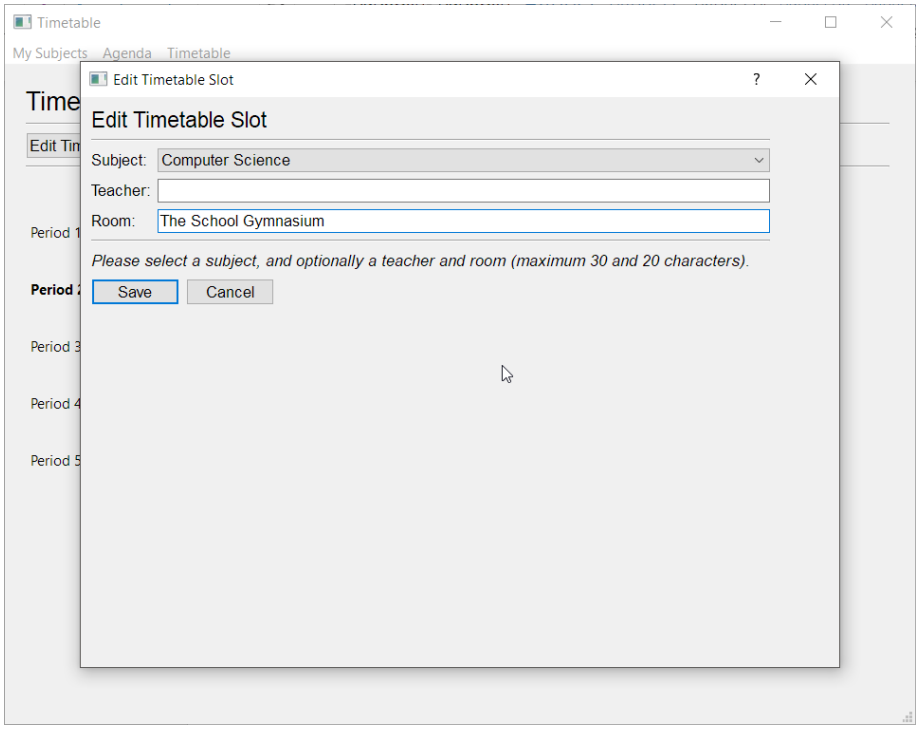
'Save' button to save the timetable slot.				selected subject, teacher, and room.
Selected the cell for Wednesday Period 3 and pressed the 'Edit Timetable Slot'. Selected 'Computer Science' as the subject, entered nothing for the teacher, and entered 'Stanborough School Gymnasium' for the room, then pressed the 'Save' button to save the timetable slot.	Erroneous	Button pressed, length check	Yes	The dialog window for editing a timetable slot was opened. The lesson was rejected, with the instruction label text changed to specify that their room input was too long, and to try again.
Selected the cell for Wednesday Period 3 and pressed the 'Edit Timetable Slot'. Selected 'Computer Science' as the subject, entered 'Mr Mason and Mrs Coldwell and Mr Modi' as the teacher, and entered nothing for the room, then pressed the 'Save' button to save the timetable slot.	Erroneous	Button pressed, length check	Yes	The dialog window for editing a timetable slot was opened. The lesson was rejected, with the instruction label text changed to specify that their teacher input was too long, and to try again.
Stopped the program and then started it again.	Normal	N/A	Yes	The timetable was loaded from *timetable.json*, meaning that progress was saved from the previous program session.

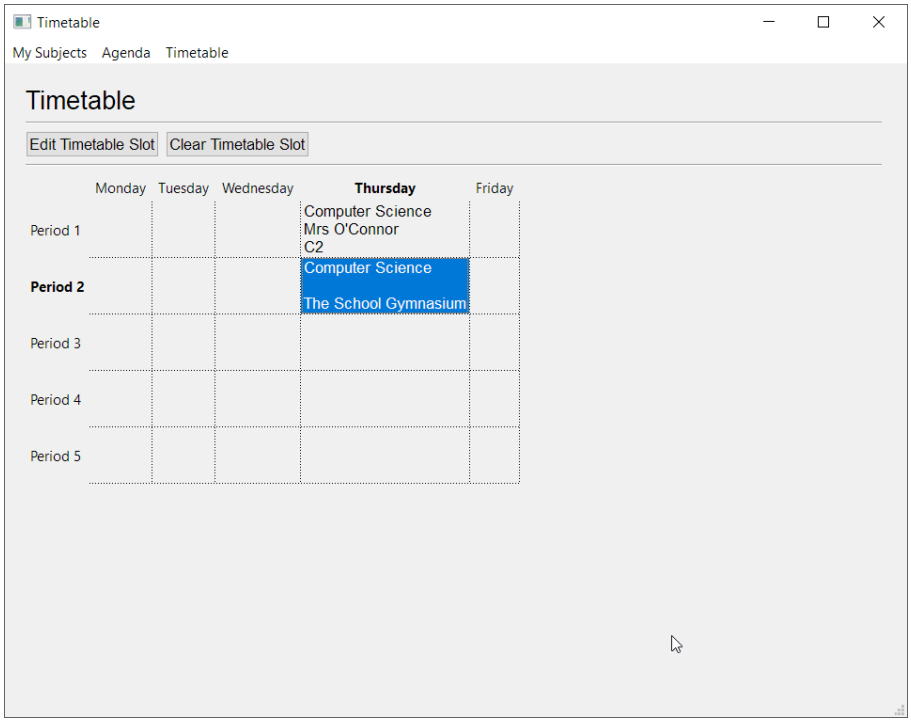
Screenshots which demonstrate the beta testing process of inputs can be seen below.

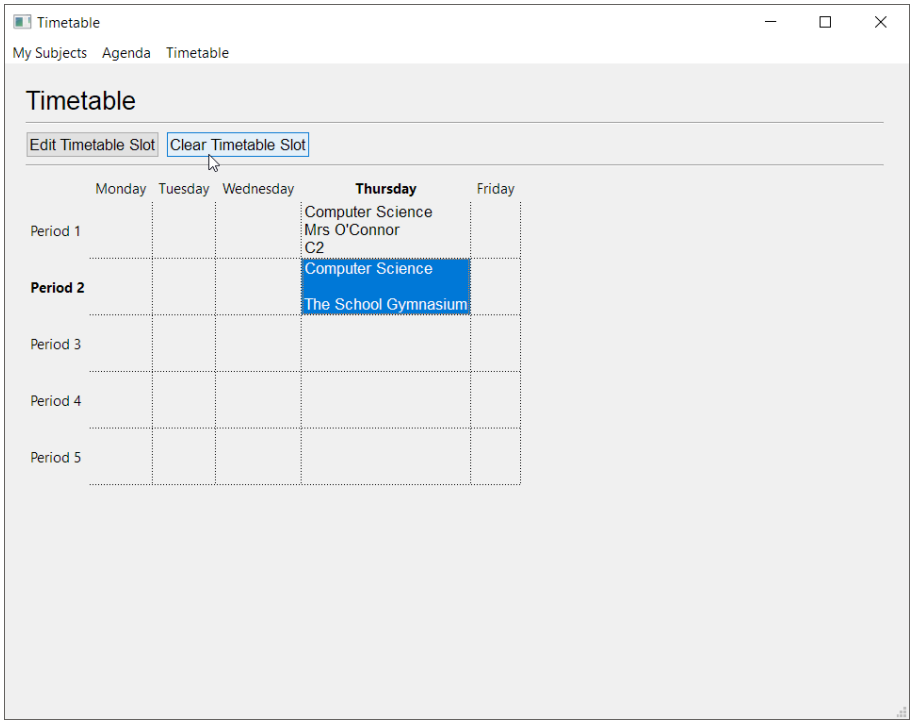

Timetable
My Subjects
Agenda
Timetable
Timetable
Edit Timetable Slot
Clear Timetable Slot
Monday
Tuesday
Wednesday
Thursday
Friday
Period 1
Period 2
Period 3
Period 4
Period 5

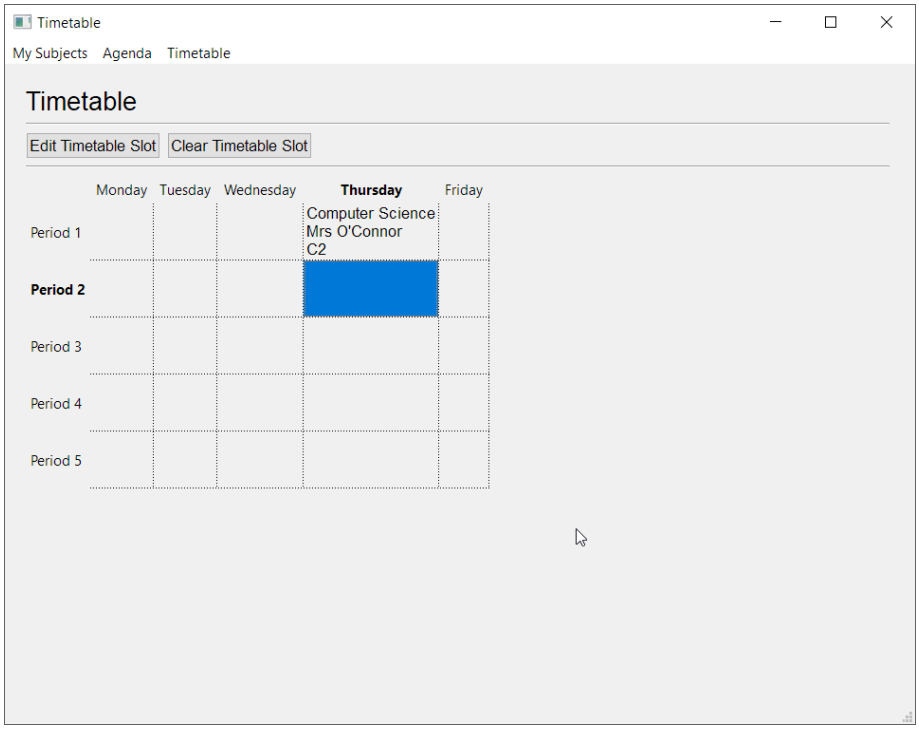

Timetable
My Subjects
Agenda
Timetable
Edit Timetable Slot
Edit Timetable Slot
Subject:
Computer Science
Teacher:
Mrs O'Connor
Room:
C2
Please select a subject, and optionally a teacher and room (maximum 30 and 20 characters).
Save
Cancel

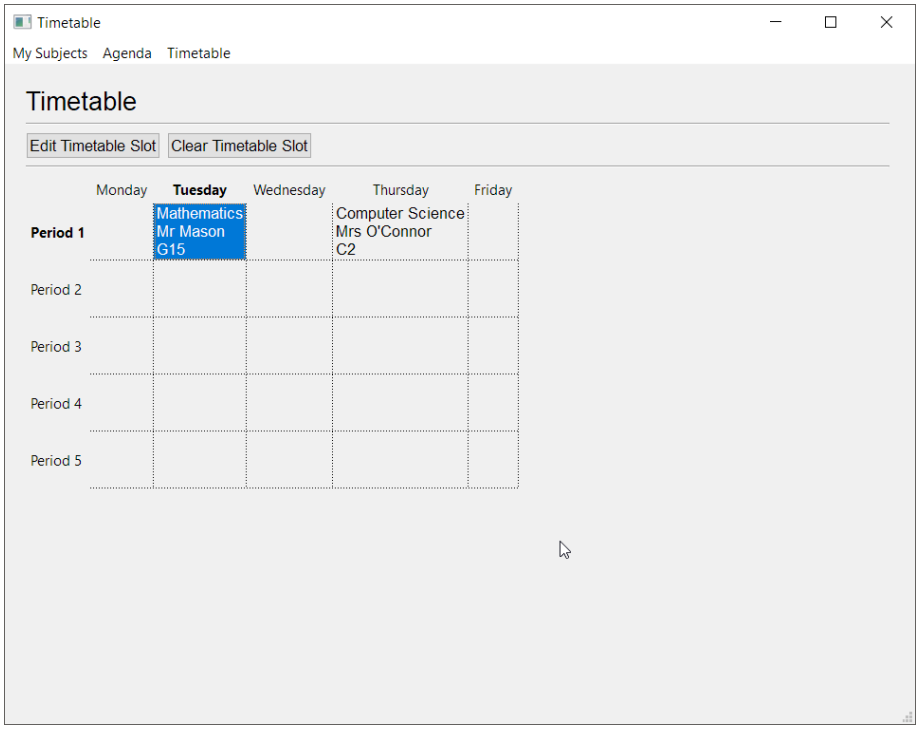

Timetable
My Subjects Agenda Timetable
Timetable
Edit Timetable Slot
Clear Timetable Slot
Monday Tuesday Wednesday Thursday Friday
Period 1
Computer Science
Mrs O'Connor
C2
Period 2
Period 3
Period 4
Period 5

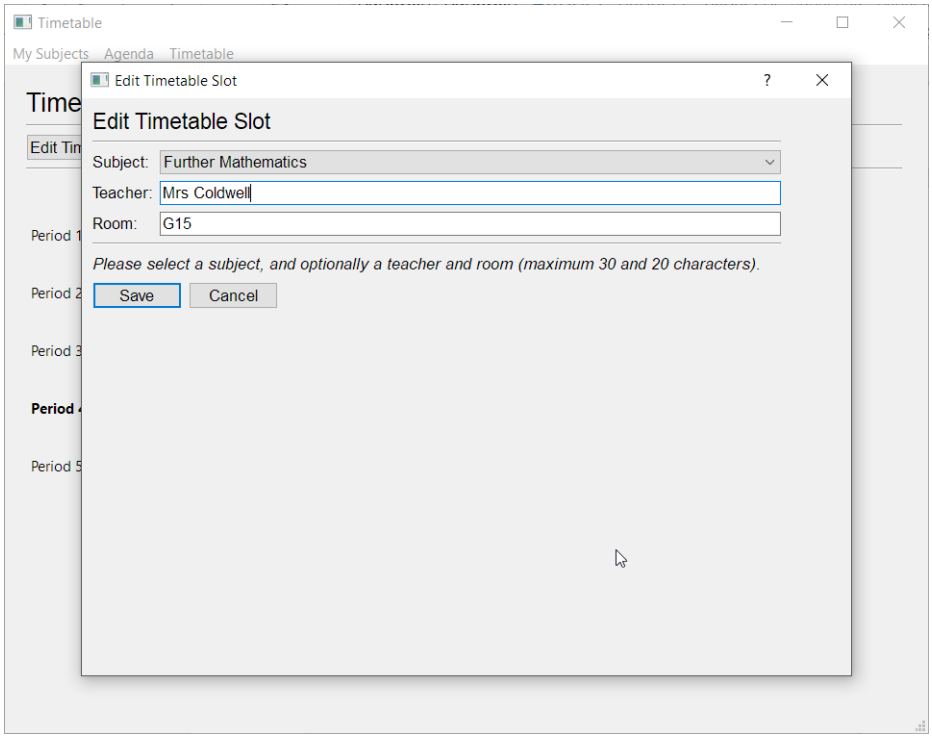

Timetable
My Subjects Agenda Timetable
Edit Timetable Slot
Edit Timetable Slot
Subject: Computer Science
Teacher:
Room: The School Gymnasium
Please select a subject, and optionally a teacher and room (maximum 30 and 20 characters).
Save
Cancel

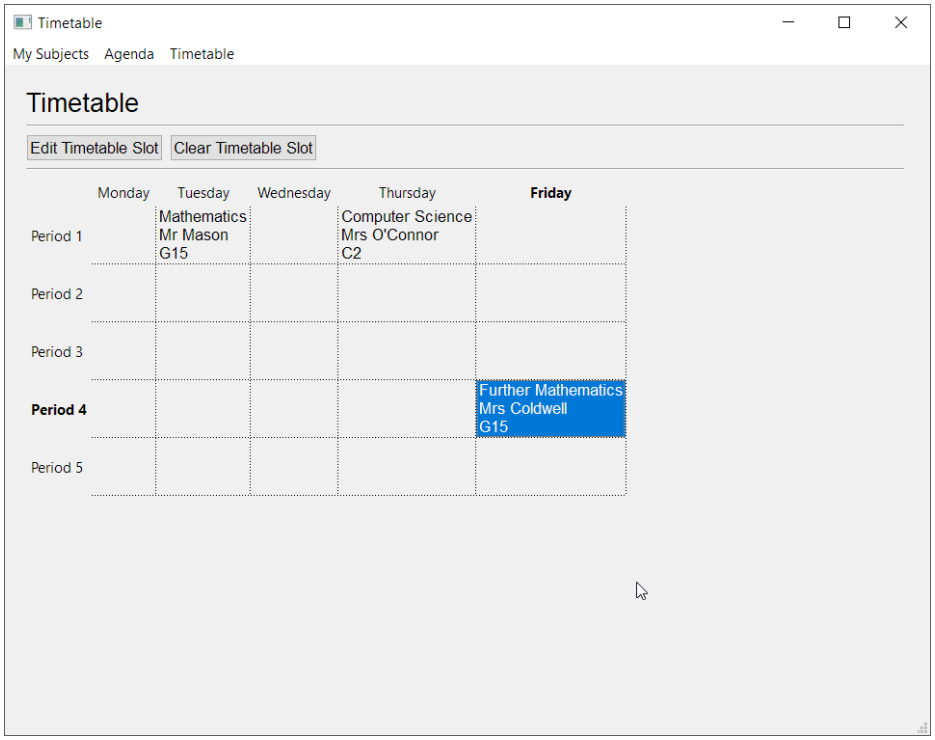

Timetable
My Subjects Agenda Timetable
Timetable
Edit Timetable Slot
Clear Timetable Slot
Monday Tuesday Wednesday Thursday Friday
Period 1
Computer Science
Mrs O'Connor
C2
Period 2
Computer Science
The School Gymnasium
Period 3
Period 4
Period 5

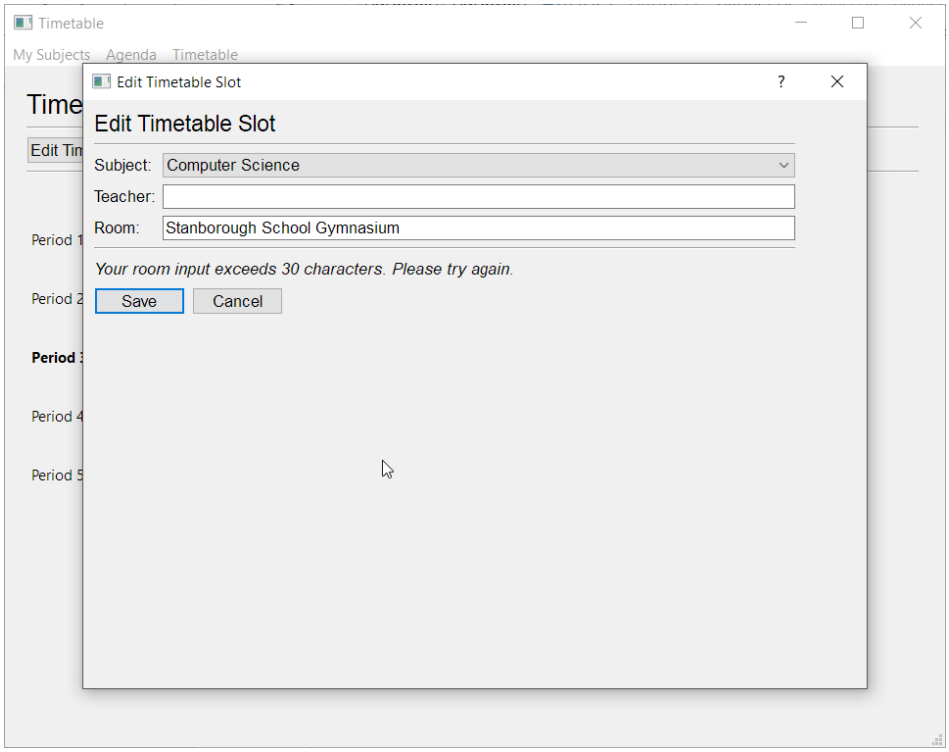

Timetable
My Subjects Agenda Timetable
Timetable
Edit Timetable Slot
Clear Timetable Slot
Monday Tuesday Wednesday Thursday Friday
Period 1
Computer Science
Mrs O'Connor
C2
Period 2
Computer Science
The School Gymnasium
Period 3
Period 4
Period 5

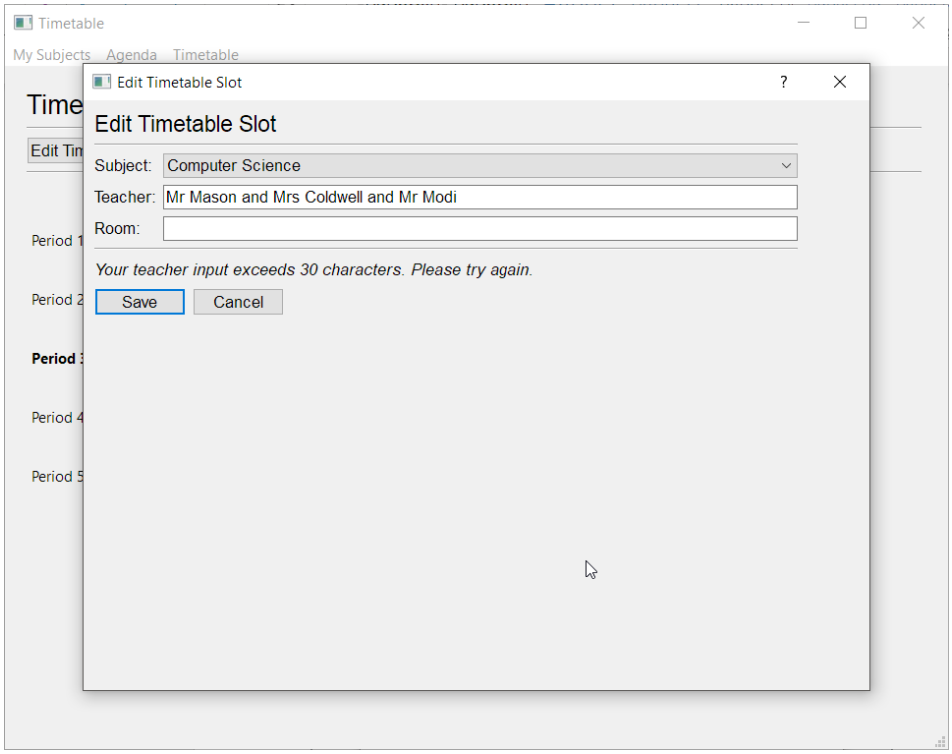

Timetable
My Subjects
Agenda
Timetable
Timetable
Edit Timetable Slot
Clear Timetable Slot
Monday
Tuesday
Wednesday
Thursday
Friday
Period 1
Computer Science
Mrs O'Connor
C2
Period 2
Period 3
Period 4
Period 5

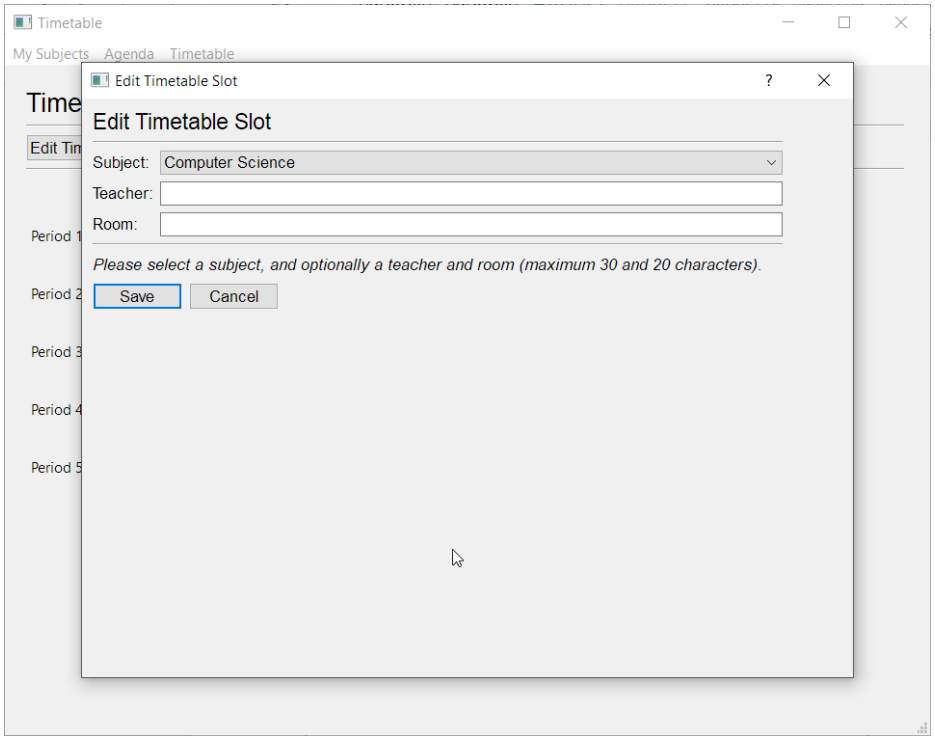

Timetable
My Subjects
Agenda
Timetable
Timetable
Edit Timetable Slot
Clear Timetable Slot
Monday
Tuesday
Wednesday
Thursday
Friday
Period 1
Mathematics
Mr Mason
G15
Computer Science
Mrs O'Connor
C2
Period 2
Period 3
Period 4
Period 5

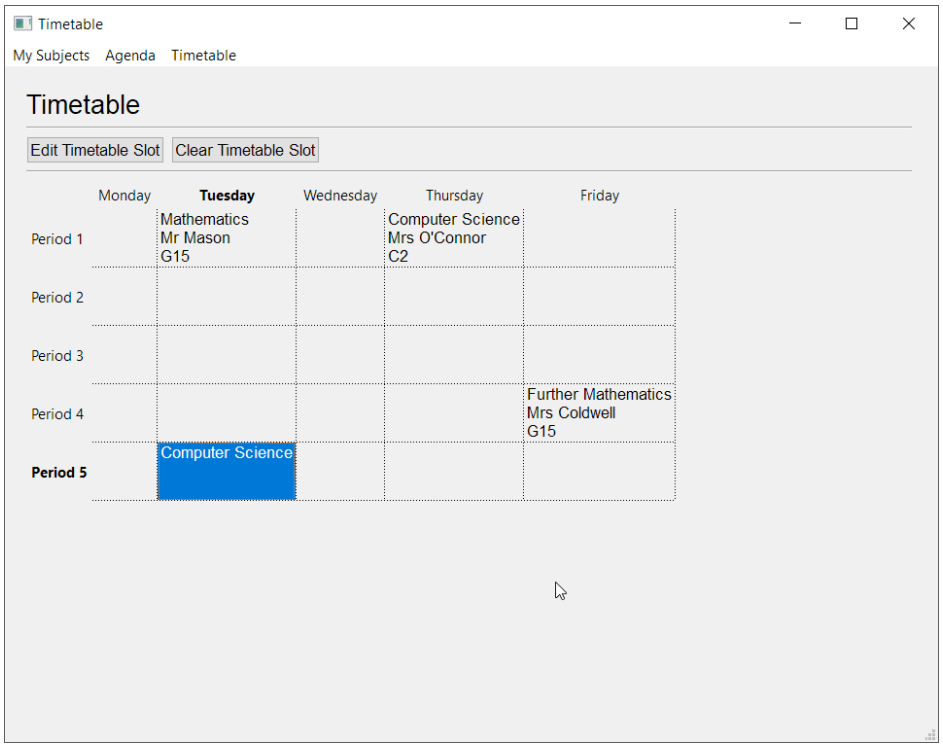

Timetable
My Subjects Agenda Timetable
Edit Timetable Slot
Edit Timetable Slot
Subject: Further Mathematics
Teacher: Mrs Coldwell
Room: G15
Please select a subject, and optionally a teacher and room (maximum 30 and 20 characters).
Save
Cancel


Timetable
My Subjects Agenda Timetable
Timetable
Edit Timetable Slot
Clear Timetable Slot
Monday
Tuesday
Wednesday
Thursday
Friday
Period 1
Mathematics
Mr Mason
G15
Computer Science
Mrs O'Connor
C2
Period 2
Period 3
Period 4
Further Mathematics
Mrs Coldwell
G15
Period 5


Timetable
My Subjects
Agenda
Timetable
Edit Timetable Slot
Edit Timetable Slot
Subject:
Computer Science
Teacher:
Room:
Stanborough School Gymnasium
Your room input exceeds 30 characters. Please try again.
Save
Cancel


Timetable
My Subjects
Agenda
Timetable
Edit Timetable Slot
Edit Timetable Slot
Subject:
Computer Science
Teacher:
Mr Mason and Mrs Coldwell and Mr Modi
Room:
Your teacher input exceeds 30 characters. Please try again.
Save
Cancel


Timetable
My Subjects Agenda Timetable
Edit Timetable Slot
Edit Timetable Slot
Subject: Computer Science
Teacher:
Room:
Please select a subject, and optionally a teacher and room (maximum 30 and 20 characters).
Save
Cancel


Timetable
My Subjects Agenda Timetable
Timetable
Edit Timetable Slot
Clear Timetable Slot
Monday
Tuesday
Wednesday
Thursday
Friday
Period 1
Mathematics
Mr Mason
G15
Computer Science
Mrs O'Connor
C2
Period 2
Period 3
Period 4
Further Mathematics
Mrs Coldwell
G15
Period 5
Computer Science

timetable.json

```
[
    {
        "subject": "",
        "teacher": "",
        "room": ""
    },
    {
        "subject": "Mathematics",
        "teacher": "Mr Mason",
        "room": "G15"
    },
    {
        "subject": "",
        "teacher": "",
        "room": ""
    },
    {
        "subject": "Computer Science",
        "teacher": "Mrs O'Connor",
        "room": "C2"
    },
    {
        "subject": " ",
        "teacher": " ",
        "room": " "
    },
    {
        "subject": " ",
        "teacher": " ",
        "room": " "
    },
    {
        "subject": "",
        "teacher": "",
        "room": ""
    },
    {
        "subject": " ",
```


timetable.json
"teacher": " ",
"room": " "
},
{
"subject": "",
"teacher": "",
"room": ""
},
{
"subject": " ",
"teacher": " ",
"room": " "
},
{
"subject": " ",
"teacher": " ",
"room": " "
},
{
"subject": " ",
"teacher": " ",
"room": " "
},
{
"subject": " ",
"teacher": " ",
"room": " "
},
{
"subject": " ",
"teacher": " ",
"room": " "
},
{
"subject": " ",
"teacher": " ",
"room": " "
},
{
"subject": " ",
"teacher": " ",
"room": " "
},

timetable.json

```
    {
        "subject": " ",
        "teacher": " ",
        "room": " "
    },
    {
        "subject": " ",
        "teacher": " ",
        "room": " "
    },
    {
        "subject": " ",
        "teacher": " ",
        "room": " "
    },
    {
        "subject": " ",
        "teacher": " ",
        "room": " "
    },
    {
        "subject": "Further Mathematics",
        "teacher": "Mrs Coldwell",
        "room": "G15"
    },
    {
        "subject": " ",
        "teacher": " ",
        "room": " "
    },
    {
        "subject": "Computer Science",
        "teacher": "",
        "room": ""
    },
    {
        "subject": "",
        "teacher": "",
```

timetable.json

```
        "room": ""
    },
    {
        "subject": " ",
        "teacher": " ",
        "room": " "
    },
    {
        "subject": " ",
        "teacher": " ",
        "room": " "
    }
]
```

All the inputs were correctly processed by the program. As you can see in the testing table, all normal and extreme inputs were accepted by the program. Erroneous inputs were rejected, with the instruction label changed to display an error message to the user.

Although null inputs are usually rejected in the case of other programs, they were correctly accepted in this program, as a subject is selected by default (the first one, so 'Computer Science' in this case).

As you can see in the final few screenshots, the timetable is successfully saved to the JSON file when changes are made.

Next, I tested general inputs for the program to check my program against the objectives set at the start of this stage.

Test	**Expected Result**	**Outcome**	**Further Actions**
Is the user able to edit timetable slots with a subject, teacher, and room?	The user should be able to edit the timetable slots with a subject, teacher, and room. Once the lesson details have been validated, the timetable slot should be updated to display these details.	The user is able to edit the timetable slots with a subject, teacher, and room. Once the lesson details have been validated, the timetable slot is updated to display these details.	N/A.
Is the user able to clear timetable slots to remove lessons?	The user should be able to select a timetable slot and clear it from the lesson details by pressing the 'Clear Timetable Slot' button so that it is empty.	The user is able to select a timetable slot and clear it from the lesson details by pressing the 'Clear Timetable Slot' button so that it is empty.	N/A.
Is the user able to add teacher and room as an option (not a requirement)?	The user should be able to add a teacher and room as optional lesson details. If they do not want to enter these details in a timetable slot, they should not be required to do so. When they attempt to save a timetable slot without either of these details, the lesson should be accepted by the program and saved to the timetable.	The user is able to add a teacher and room as optional lesson details. If they do not want to enter these details in a timetable slot, they are not required to do so. When they attempt to save a timetable slot without either of these details, the lesson is accepted by the program and saved to the timetable.	N/A.
Is the timetable saved to the JSON file, *timetable.json*, whenever changes are made to the timetable?	The JSON file, *timetable.json*, should be updated whenever changes are made to the timetable, such as a timetable slot being edited or cleared. When the user starts up the program, it should read from *timetable.json* and	The JSON file, *timetable.json*, is updated whenever changes are made to the timetable, such as a timetable slot being edited or cleared. When the user starts up the program, it reads from *timetable.json* and loads in the timetable to the *QTableWidget.*	N/A.

	load in the timetable to the *QTableWidget.*		

The general tests demonstrate that my program is fully functioning, meeting all the objectives set at the start of this stage, as well as some extra functionality such as clearing timetable slots.

Review

My objectives for this stage were to develop functionality in Timetable which enables the user to view their timetable in a 5 x 5 table format, and edit timetable slots with a subject, teacher, and room.

I have successfully developed this functionality in this program, as well as some extra functionality such as clearing timetable slots. Functionality is implemented with robustness; there has been consideration for all use cases, such as populating the list with empty lessons if there are missing indexes in the timetable list (which would otherwise cause a missing index error when populating the timetable from the list).

Evaluation

Usability Features Evaluation

In the design section, I stated a plan for the usability features which I had intended to implement into my program. They formed the specific requirements for my program, whilst I also created success criteria which formed more general requirements for my program.

To determine whether I have met the requirements specification stated in the usability features section and the success criteria, I have set up tables with two columns; one states the requirement, and another explains whether it was met and why. In some cases where the requirement was not met, I also explained how it could be developed in the future.

The requirements have been broken down into three tables for the usability features to represent the types of requirement I broke the usability features into previously, and one table for the success criteria.

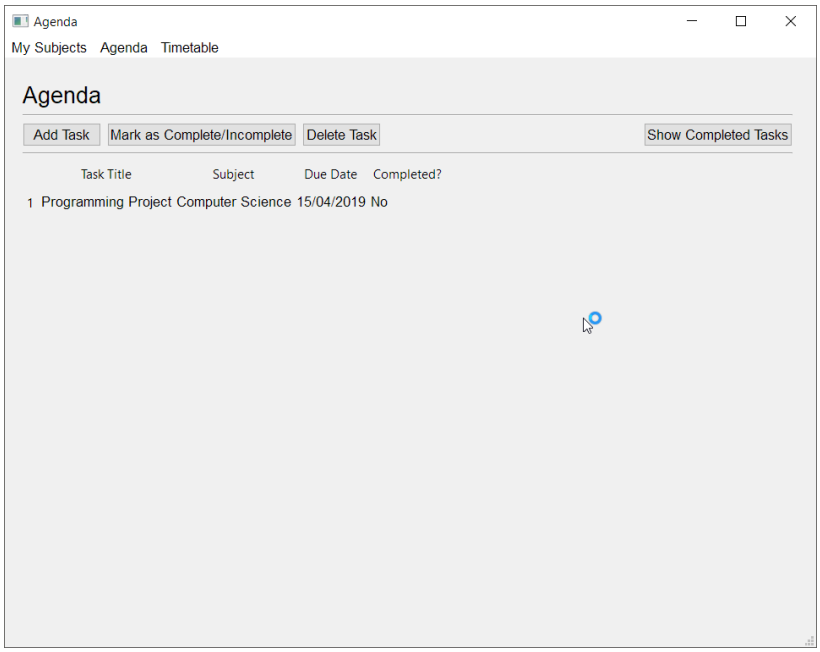
Usability Features: Input

A summary of the usability features evaluation for the inputs is shown below in the table.

Requirement	**Requirement Met?**
Buttons; used for simple actions which only have a single purpose and need to be performed quickly and conveniently.	Yes. Buttons have been used as a simple method for the user to access dialog windows for actions such as adding a new subject or adding a new task. They have also been used to interact with the list and table widgets in My Subjects, Agenda, and Timetable. For example, they are able to delete subjects in My Subjects, they can mark tasks as complete/incomplete in Agenda, and they can delete lessons in Timetable.
Tick boxes; used for marking tasks as complete or incomplete, with the	No.

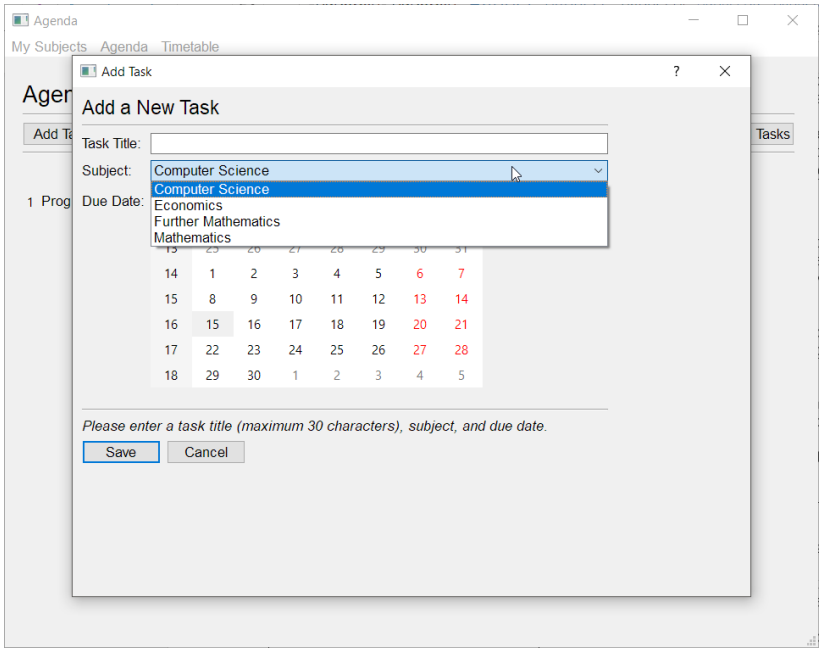
ability to quickly untick/re-tick a task if it has been accidentally pressed, as a way of undoing their previous action.	Tick boxes have been discovered to be obsolete in my application, as they would not fit in with the design aesthetic of my application. I have implemented the features in a more suitable manner which is less prone to accidental use. Rather than ticking or unticking tasks to mark tasks as complete/incomplete, the user can select the task from the *QTableWidget*, and then press a toggle button to mark it as complete/incomplete.
Dropdown menus; used for selection of user input from predetermined options to make it easier and quicker to input/change things repetitively.	Yes. Dropdown menus (also known as combo boxes) have been used in Agenda and Timetable for the dialog when adding a task/editing a timetable slot. They provide a way for the user to quickly select their subject by clicking the option rather than typing it manually. The options are read from *subject_list.txt*, which can be edited through the list in My Subjects, and added to the combo box.
Calendars; used for convenient selection of dates to make it easier for the user to identify which date they are looking for, such as by providing them the day of the week which corresponds to each date, like a traditional calendar.	Yes. The built-in *QCalendarWidget* has been used to enable the user to select the due date for tasks through a graphical user interface, which makes it easier for them to visualise what the date is in exactly one or two weeks' time, which are common due dates for homework tasks.
Text boxes; used to give the user flexibility to add any notes/additional information they want to remind themselves of, as opposed to limiting them with other input mediums. May also be used to input details about subjects and teachers to enable the dropdown menu to be used.	No. The feature for the user to add additional notes has not been implemented, as a survey of sixth form students has shown that 90% of sixth form students would prefer not to have this feature, because they want to keep the application simple, and quick to use. According to the feedback, adding the ability to add notes would add clutter to the user interface, making it more difficult to navigate and less user friendly. Whilst I have not implemented the ability to take notes yet, it would be one of the higher priority features in future development. I could develop this feature to be an opt-in setting, so that it would not annoy the 90% of sixth form students who would not like to use this feature. Subjects can be added through My Subjects to add new options to the dropdown menus, and the teachers can be added to timetable slots using a line edit widget.

Evidence of Usability Features: Input



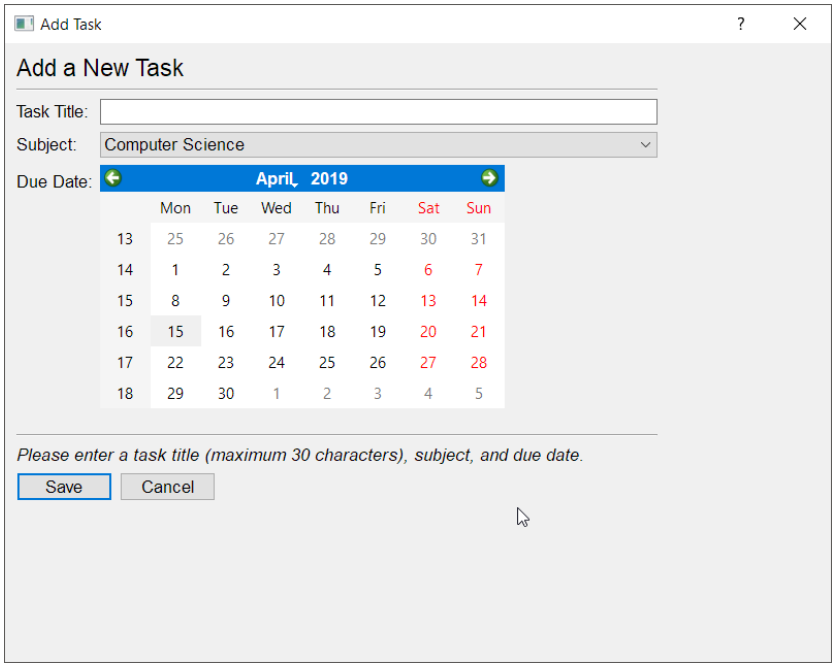
The screenshot above shows buttons being used in my application, as four buttons were included in Agenda.

The first button triggers a dialog window for adding a new task. The second button toggles the selected task as being marked as complete/incomplete. The third button deletes the selected task from the agenda, and the fourth button shows/hides completed tasks from the agenda.



The screenshot above shows a dropdown menu being used in my application, as it was used when adding a new task in Agenda.

This dropdown menu (combo box) has options which were read from *subject_list.txt*, and it allows the user to select a subject from the list so they can add tasks faster than if they had to type their subject manually.



The screenshot above shows a graphical calendar widget being used in my application, as it is used when adding a new task in Agenda.

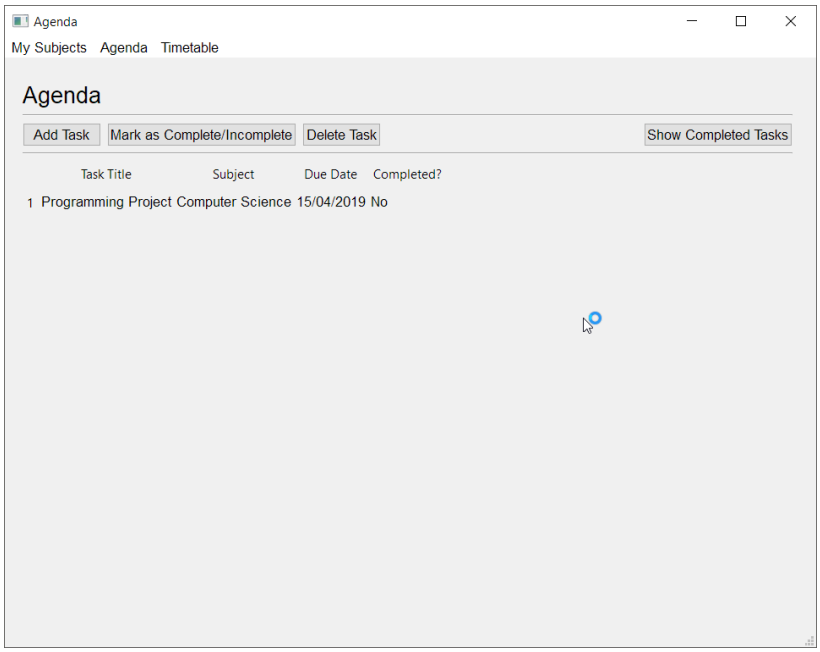
It allows the user to select their due date from a monthly view in the calendar, which makes it quicker for the user to select a due date, and easier for them to see what day to select (especially in cases where students are often set homework with a due date of next week).

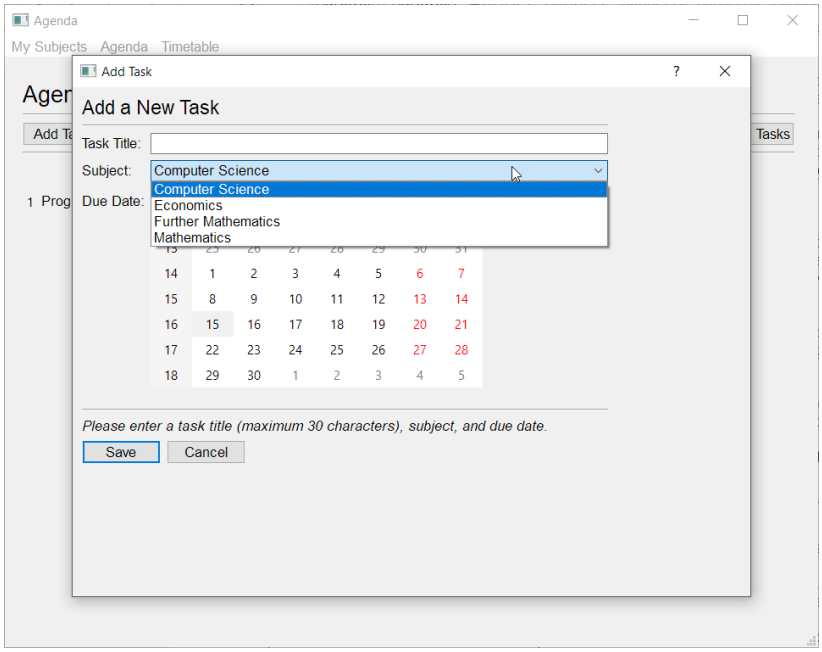
Usability Features: Output

Requirement	**Requirement Met?**
Headers; used to help the user navigate across the application by informing them where in the application they currently are.	Yes. Headers have been consistently applied at the top of each window in a significantly larger font size than the rest of the window, which has font size 10. Further differentiation between main windows and dialogs has also been created subtly using the font size of headers; main windows have a header of font size 16, whereas dialog windows have a header of font size 14.
Sub-headers; used to guide the user within each screen, summarising what information each part of that screen contains and/or the purpose of that part of the screen.	Yes. Sub-headers are not needed as much as I thought they would be needed in my application, but they are used in the *QTableWidget* in Agenda to label the columns, such as 'Task Title', 'Subject', and 'Due Date'.
Text labels; used to provide additional information to enhance the understanding of the user, or as hints for what the user should input.	Yes. Instruction text labels are present in the dialog windows, such as when the user wants to add a new subject to My Subjects or a new task to Agenda. They are formatted in italics to indicate

	that the user should pay special attention to reading this text, and they provide hints to the user as to what they should input, and why their inputs failed validation if they were rejected.
Icons; used to provide the user with visual information about a button.	No. My research shows that modern design language has shifted since this project was originally planned; companies such as Google and Apple are shifting away from the use of iconography to guide the user, instead preferring to create a more minimalistic, easy-to-use user interface. These two companies produce some of the most used software in the world, so using a similar design language would make students familiarise themselves with my application more easily. Furthermore, icons are often used in cases where differentiation between features and sections of an application is more severely needed. It is not often used in applications with high information density, as it takes up more space. My application more closely resembles the latter scenario than the former scenario.
Sound effects; optional audio cues after clicking to provide feedback that the system is processing their click.	No. User feedback from sixth form students (my primary stakeholders) stated that sound effects would be a distraction to other people in the typical environments the program would be used, so this would have been an unnecessary feature to develop and maintain. This is a feature which could be developed in the future, as although it would complicate my program unnecessarily, it could be implemented as an optional setting which is turned off by default in a settings menu. However, it would be a lower priority for future development, as it would likely be used by very few people.

Evidence of Usability Features: Output

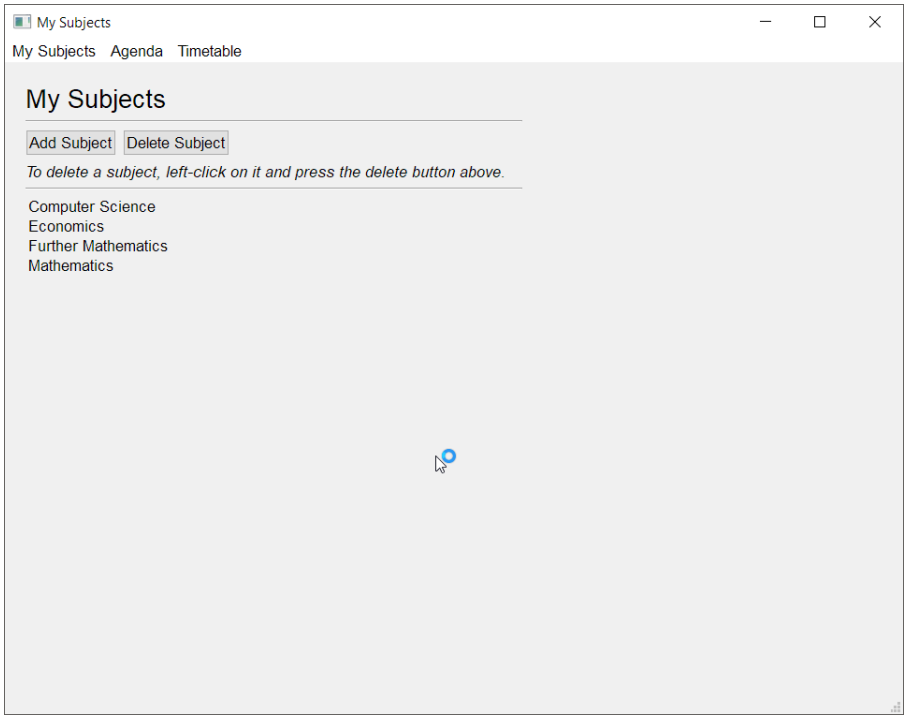




The two screenshots above show headers being used in my application, as there is one for the main window of Agenda and one for the Add Task dialog in Agenda. It also shows sub-headers being used in my application in the first screenshot, as there are several in the *QTableWidget* in Agenda.

It is clear in the second screenshot especially that the dialog header is noticeably smaller than the main window header. This is used to help the user differentiate between the main windows and the dialog windows.

Sub-headers are used for the *QTableWidget* in Agenda to specify to the user what task detail is being displayed in that column. For example, the first column shows the task titles, and the second column shows the subjects of each task.



The screenshot above shows instruction labels being used in my application, as there is one being used in My Subjects.

Instruction labels are used throughout the application to guide the user in cases where it may be easy for them to get confused. In this case, it is not immediately obvious that the user can select a subject from the list by left-clicking on it, so I used an instruction label to guide them on how to select a subject to delete it.

Usability Features: Storage

Requirement	**Requirement Met?**
Temporary storage; used for storing data and transporting it across a program to be used during the execution of a program.	Yes. As my application is based on the organisation of data, it is important to have a quick means of transporting data throughout the execution of a program. Temporary storage is used consistently throughout my application using variables. For example, in My Subjects, the variable *subject* is used to temporarily store the currently iterated subject in the list as the list is being saved to *subject_list.txt*. Without this, it would not be possible to save the list for permanent storage by iteration, meaning it would be a lot more difficult to save the subject list.
Permanent storage (data persistence) using .txt files and .json files; used for data which would need to be kept even after the application is closed so that the user can record things such as their tasks, timetable slots, and grades.	Yes. .txt files have been used to store the user's list of subjects in My Subjects permanently, with data atomicity ensured by using two separate files when sorting the list in alphabetical ascending order. This means data will not be lost even if the

	program crashes midway through the sorting process; the saved data will be stored in either the original file (*subject_list.txt*) if the program crashes during initial sorting, or the temporary sorted file (*subject_list_temp.txt*) if the program crashes during the duplication of the sorted list.

Evidence of Usability Features: Storage

```
55      # Saves the subject list.
56      def save_subject_list(self):
57          with open("subject_list_temp.txt", "w") as outfile:
58              for i in range(self.list_widget_my_subjects.count()):
59                  subject = self.list_widget_my_subjects.item(i).text()
60                  outfile.write(subject + "\n")
61              with open("subject_list.txt", "w") as outfile:
62                  for i in range(self.list_widget_my_subjects.count()):
63                      subject = self.list_widget_my_subjects.item(i).text()
64                      outfile.write(subject + "\n")
```

The screenshot above of the code from *my_subjects.py* shows temporary storage being used in my application, as it is used in My Subjects to save the subject list.

In this method, *save_subject_list()*, the program iterates through each subject in the *QListWidget*, gets the subject on that line, and adds it to the next line in *subject_list.txt*. Without using *subject* as a variable for temporary storage, the line iteration would not be possible.

```
1   [
2       {
3           "task_title": "Programming Project",
4           "subject": "Computer Science",
5           "due_date": "15/04/2019",
6           "completed": "No"
7       },
8       {
9           "task_title": "Practice Paper",
10          "subject": "Mathematics",
11          "due_date": "18/04/2019",
12          "completed": "No"
13      },
14      {
15          "task_title": "Practice Paper",
16          "subject": "Further Mathematics",
17          "due_date": "18/04/2019",
18          "completed": "No"
19      }
20  ]
```

The screenshot above of the contents of *task_list.json* shows permanent storage being used in my application, as it is used in Agenda to store the task list as a list of dictionaries.

In this case, the .json file enables the task details of each task to be stored as a dictionary, and a list to be used to store these dictionaries. It enables the program to read from this file and populate the agenda when the program is started, so the user's progress can be saved. Without permanent storage, the user's tasks in the agenda would be lost after they exit the program.

User Acceptance Testing

To ensure that my application has been developed to meet the needs of my stakeholders, I have performed a wide variety of external user acceptance tests in a real world environment.

This will provide me with feedback about the successes and failures of my application, which will help me evaluate whether I have met the requirements for this application fully, partially, or not at all in the next section.

Choosing Test Criteria

User acceptance testing was planned so that the end-user would test my application against criteria I have set in different scenarios, which are shown below.

Criterion has been set for each part of my application to include both functional and robustness testing throughout the application.

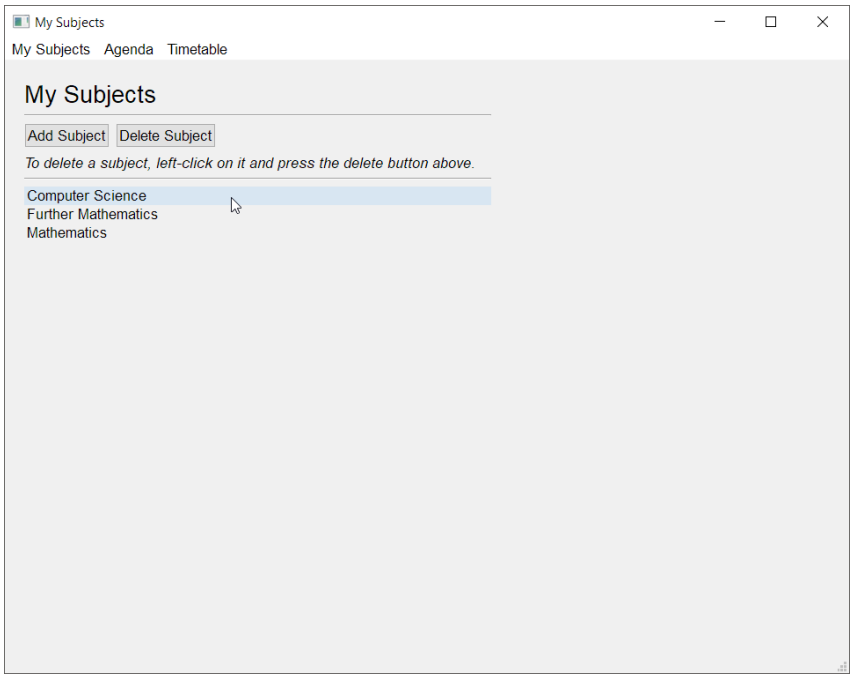
Functional testing is about testing the features of the program which have been implemented. It is related to the original requirements and specifications. This type of testing ensures that the program meets the needs of the stakeholders, and that the program has been designed to specification.

Robustness testing is about testing the limitations of the program. It is related to testing the quality of features implemented in the program and how it can handle unexpected inputs (such as extreme or invalid inputs). This type of testing ensures that the program is highly adaptable to various environments, and that it would be able to cope with the various use cases in a real world environment.

Test Results: My Subjects

Test #	Criterion	Criterion Met?
#1	Main window which displays a list of the user's subjects.	Yes
#2	List of user's subjects sorted in alphabetical ascending order.	Yes
#3	Sleek, simple design which is easy to use.	Yes
#4	High information density.	Yes
#5	Ability to add new subjects to the list through a dialog window.	Yes
#6	Input for new subject name validated by a presence and length check.	Yes
#7	Ability to delete subjects from the list.	Yes
#8	Updated subject list saved to a .txt file for permanent storage whenever a change to the subject list is made.	Yes
#9	List of user's subjects loaded in from the .txt file whenever the program is started.	Yes
#10	Rejects new subject names which are blank or only filled with whitespace.	Yes
#11	Rejects new subject names which exceed 30 characters.	Yes

Test Evidence: My Subjects



The screenshot above demonstrates that test #1, test #2, test #3, test #4, and test #9 were met successfully.

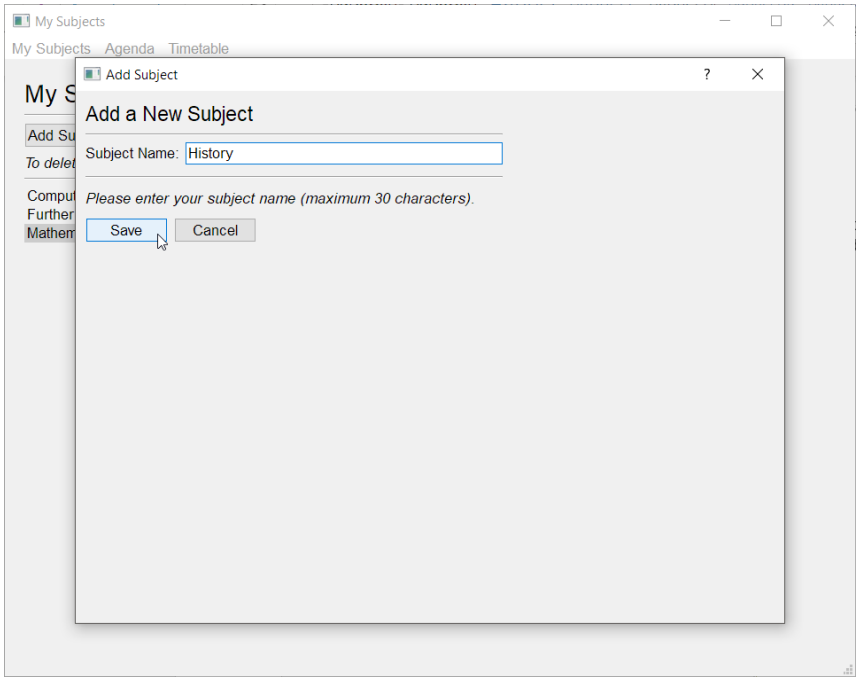
There is a main window which displays the user's subjects in the form of a list which is sorted in alphabetical ascending order. In this screenshot, it displays the user's three subjects; Computer Science, Further Mathematics, and Mathematics.

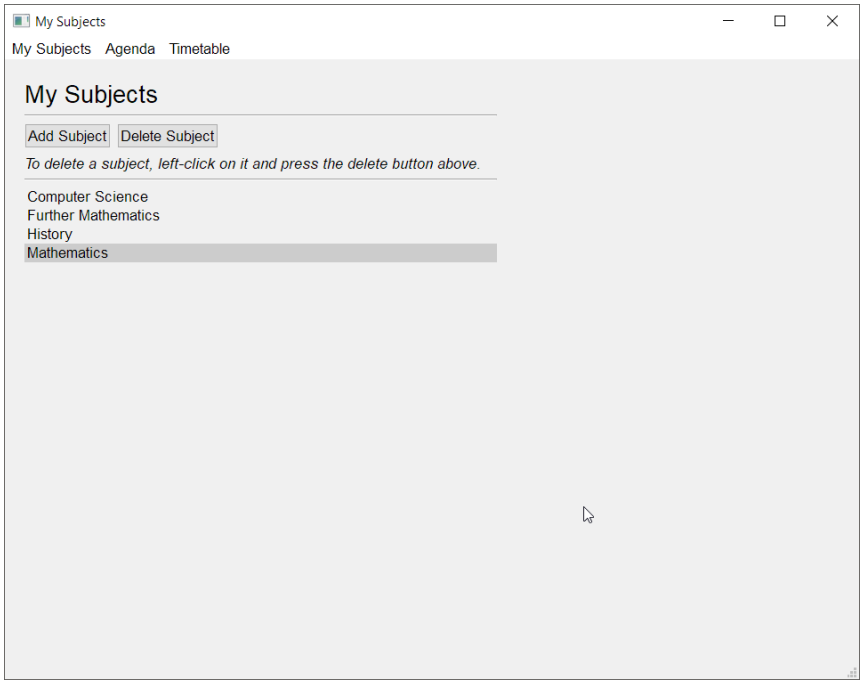
The user interface is sleek, with minimal clutter, and only shows the essential features. There is no unnecessary artwork or design elements which could distract the user from their work. Horizontal lines effectively separate the header of the main window, 'My Subjects', from the buttons, which makes the user interface easier on the eyes.

This program is easy to use, as the buttons are clearly labelled to state their purpose. There is an instruction label below the buttons to explain how they can select which subject they want to delete.

Information density is maximised by using an appropriate amount of whitespace between elements. There is space for many more subjects to be added until the user is required to scroll; the user would likely be able to add 20-30 subjects in that space, which is more subjects than any GCSE or A-Level student takes.

These subjects were loaded in from a previous session, which shows that the program successfully loaded in the user's subjects from the .txt file.

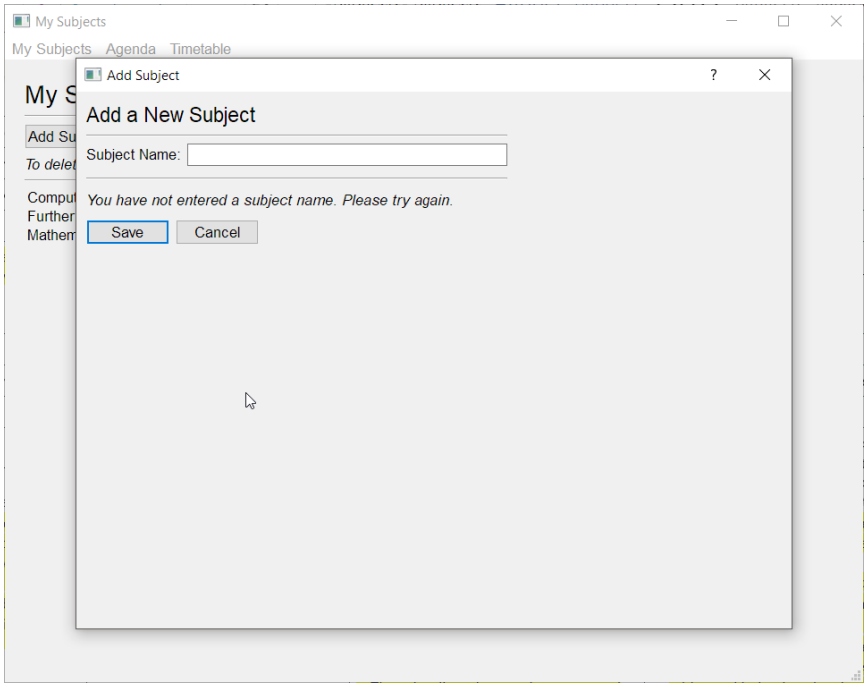




The screenshots above show that test #5 was met successfully.

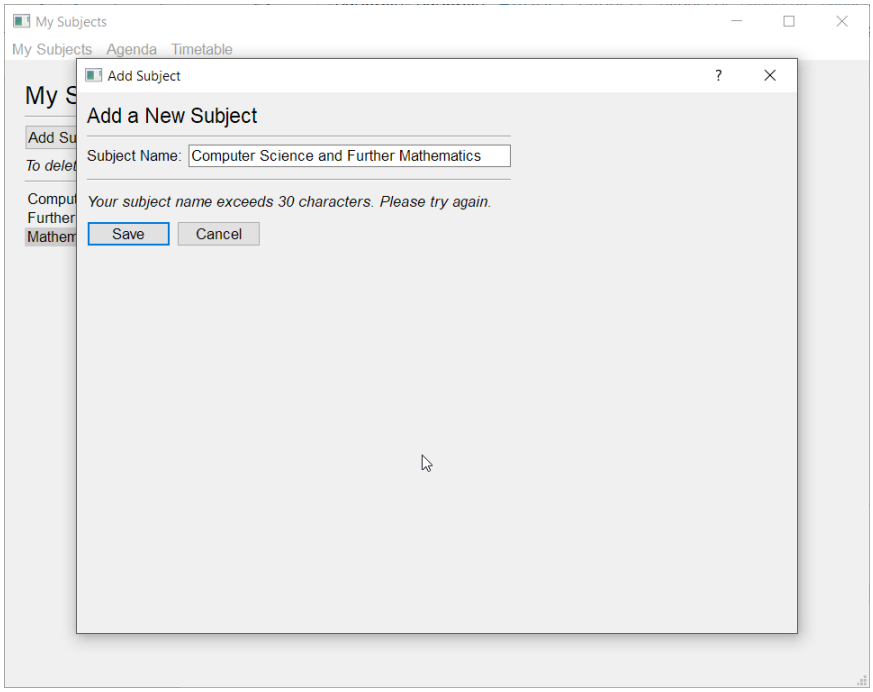
The dialog window enables the user to add a new subject, with a line edit widget for the user to input their new subject name, and the instruction label telling them that it can be a maximum of 30 characters. The user is able to save the subject by pressing the ‘Save’ button, or cancel their input by pressing the ‘Cancel’ button.

After they saved their subject, it is added to the list if it passes the presence and length check validations.



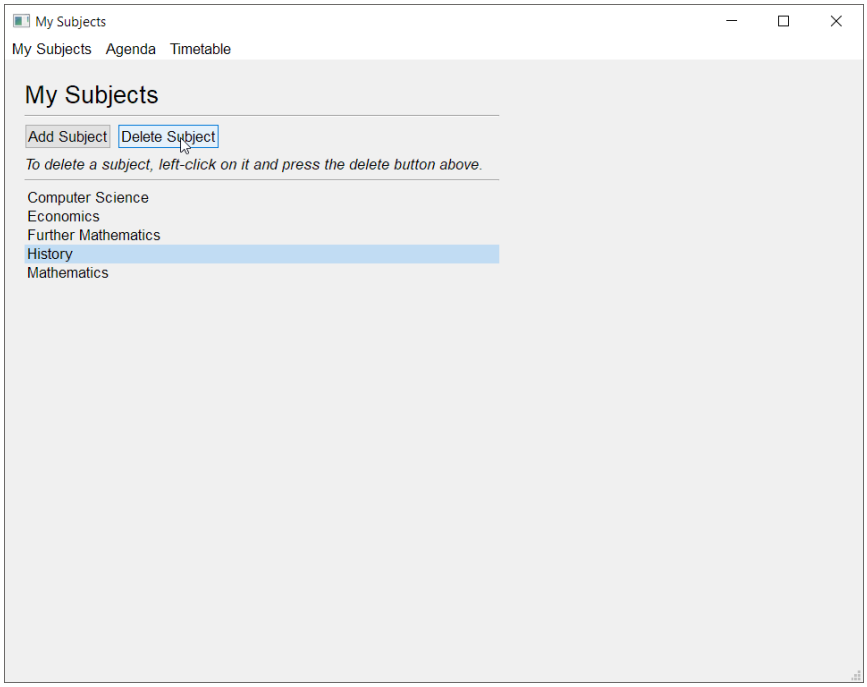
The screenshot above shows that test #6 and test 10 were met successfully.

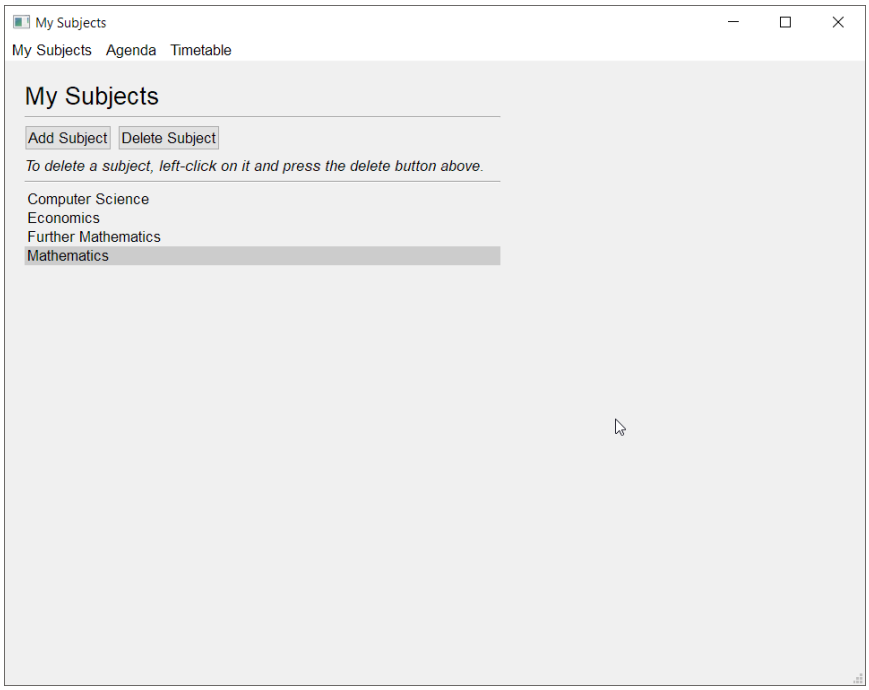
The instruction label is changed to state that the user has not entered a subject name, and the dialog window is kept open, meaning that the subject has not been added to the list of subjects.



The screenshot above shows that test #6 and test #11 were met successfully.

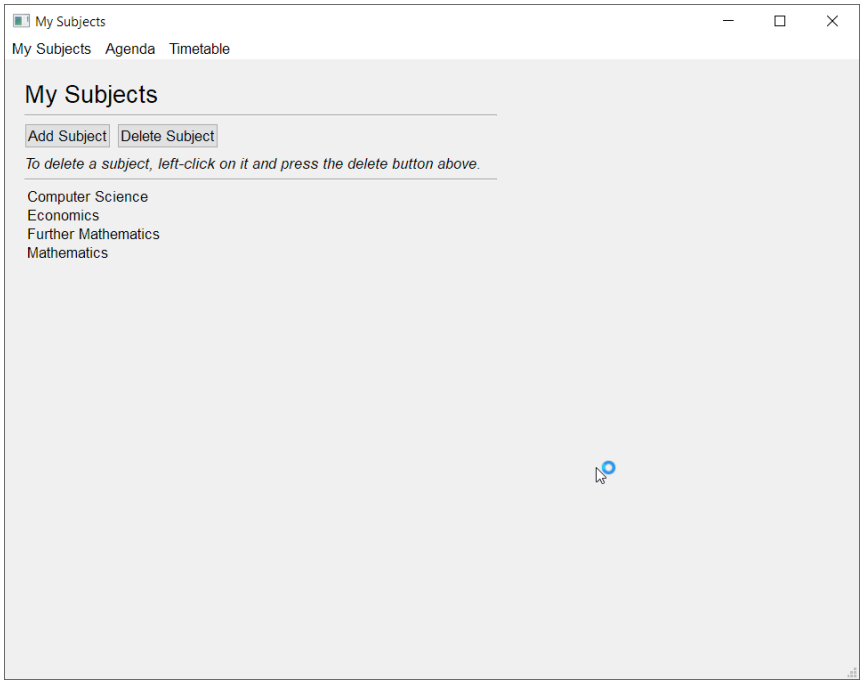
The instruction label is changed to state that the user's subject name exceeds 30 characters, and the dialog window is kept open, meaning that the subject has not been added to the list of subjects.





The screenshots above show that test #7 was successful.

When the user selects a task and then presses the button to delete the subject, as instructed, the subject is deleted from the list. In this case, the user selected ‘History’ and deleted it successfully.



The screenshot above shows that test #8 and test #9 were successful.

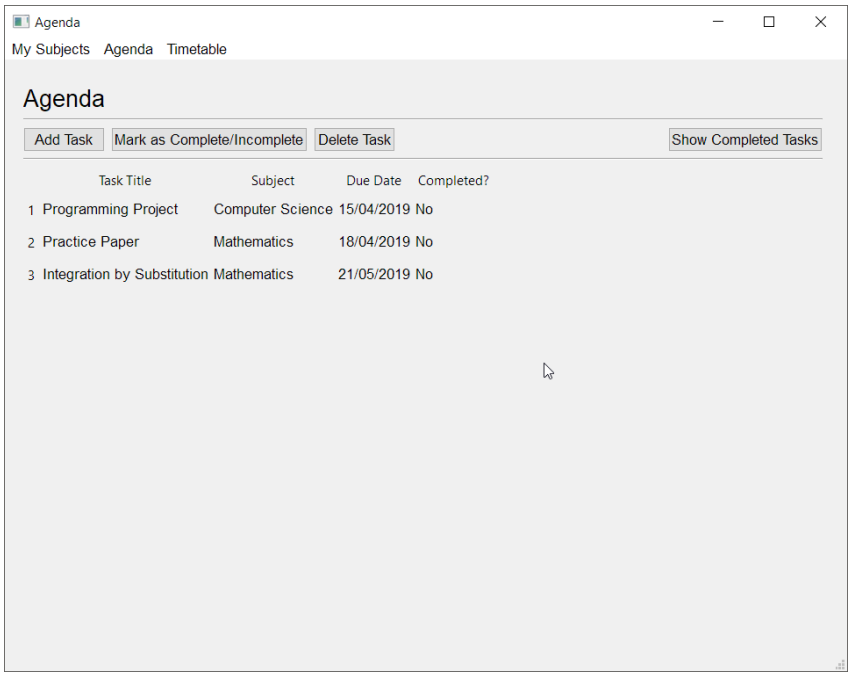
In the first session, the user added 'History' and 'Economics' to their subject list, then deleted 'History' from the list, and closed the program. After they opened the program again, they were presented with the same subject list which they ended the previous session with, suggesting that the changes were saved each time they updated the subject list.

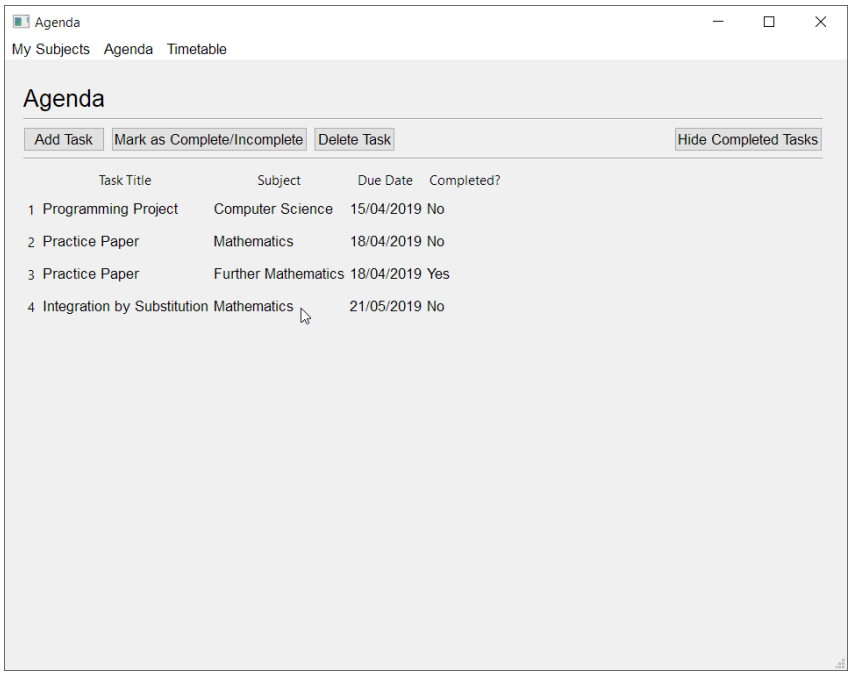
Test Results: Agenda

Test #	Criterion	Criterion Met?
#1	Main window which displays a list of the user's tasks and their task details (task title, subject, due date, and completion), also known as an agenda.	Yes
#2	List of user's tasks sorted by due date ascending order.	Yes
#3	Sleek, simple design which is easy to use.	Yes
#4	High information density.	Yes
#5	Ability to add new tasks to the list through a dialog window.	Yes
#6	Input for new task title validated by a presence and length check.	Yes
#7	Input for new task subject through a combo box which is linked to the subjects list in My Subjects.	Yes
#8	Input for new task due date through a graphical calendar widget.	Yes
#9	New tasks automatically marked as incomplete.	Yes
#10	Ability to mark tasks as complete if they are incomplete, and incomplete if they are complete (toggling completion status).	Yes
#11	Ability to delete tasks from the agenda.	Yes
#12	Ability to show or hide completed tasks from the agenda.	Yes
#13	Updated task list saved to a .json file for permanent storage whenever a change to the agenda is made.	Yes
#14	List of user's tasks and task details loaded in from the .json file whenever the program is started.	Yes
#15	Rejects new task titles which are blank or only filled with whitespace.	Yes

#16	Rejects new task titles which exceed 30 characters.	Yes
#17	Creates task list if no task list exists (new user).	Yes
#18	Ability to add notes about a task.	No
#19	Ability to display notes about a task.	No

Test Evidence: Agenda





The screenshots above demonstrate that test #1, test #2, test #3, test #4, test #12, and test #14 were successful. It also demonstrates that test #19 was unsuccessful.

The main window displays the user's list of tasks and their task title, subject, due date, and completion status in a table, sorted by due date ascending order.

The table has been automatically resized to fit the task details, as shown in the second screenshot, where the ‘Subject’ column is made wider to accommodate for the longer subject, ‘Further Mathematics’.

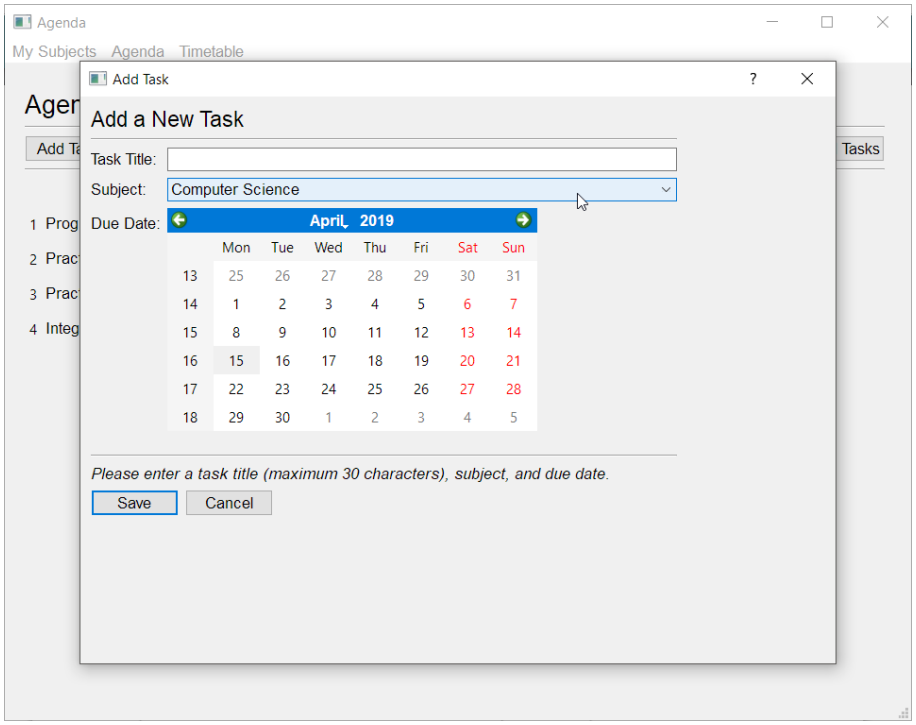
The design is sleek, taking a minimalist approach to show only the essential information. There are no columns or rows which are too big or small for the contents, and there is clear labelling for the columns, which state what task detail they are showing below them.

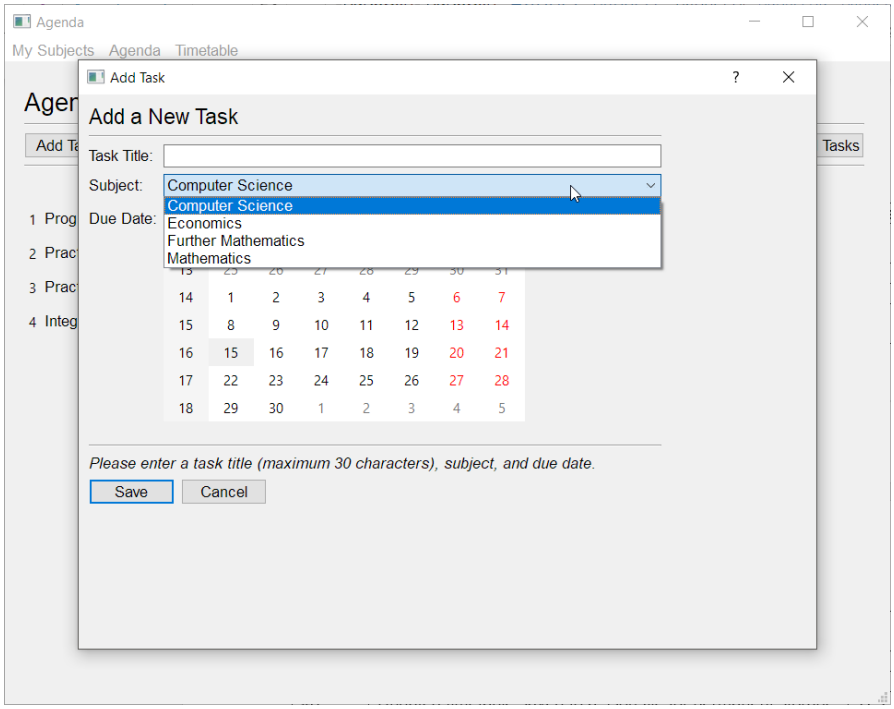
Information is as densely packed as possible in the table widget due to the rows and columns being resized to fit the content.

Tasks can be viewed in two ways. Completed tasks are hidden by default, but the user can press the button on the top right of the screen to switch the task view mode (to hide/show completed tasks).

This program was started, and the table widget was populated with these tasks from a previous session. This shows that test #14 was successful, as the tasks must have been loaded in from the .json file for permanent storage, because variables can only store data within a program session (temporarily).

There is no column for the task notes, and hovering over tasks does not display additional notes either.

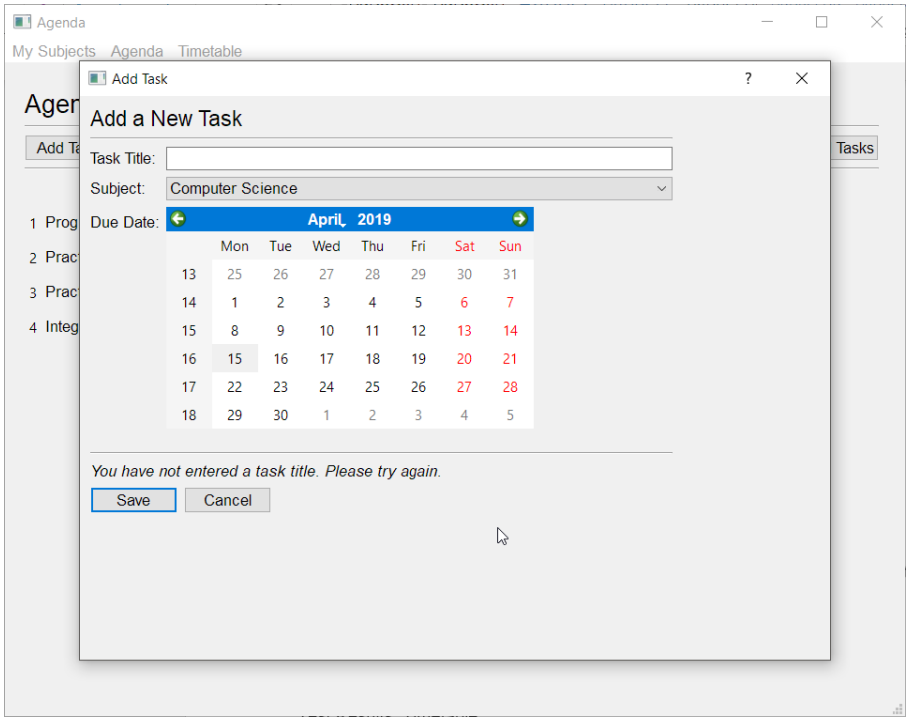


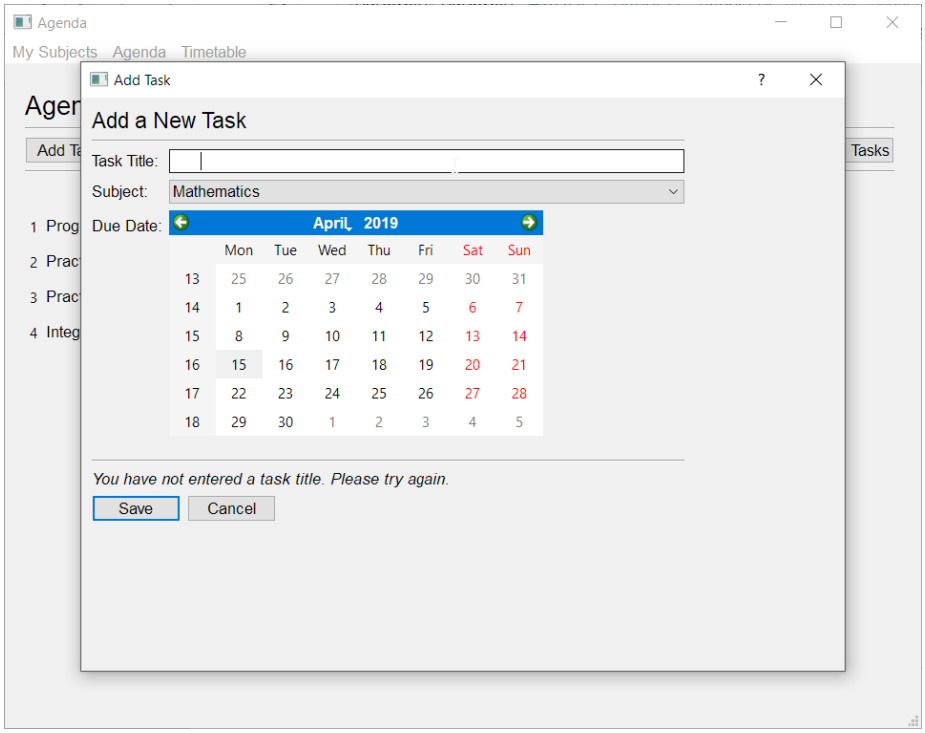


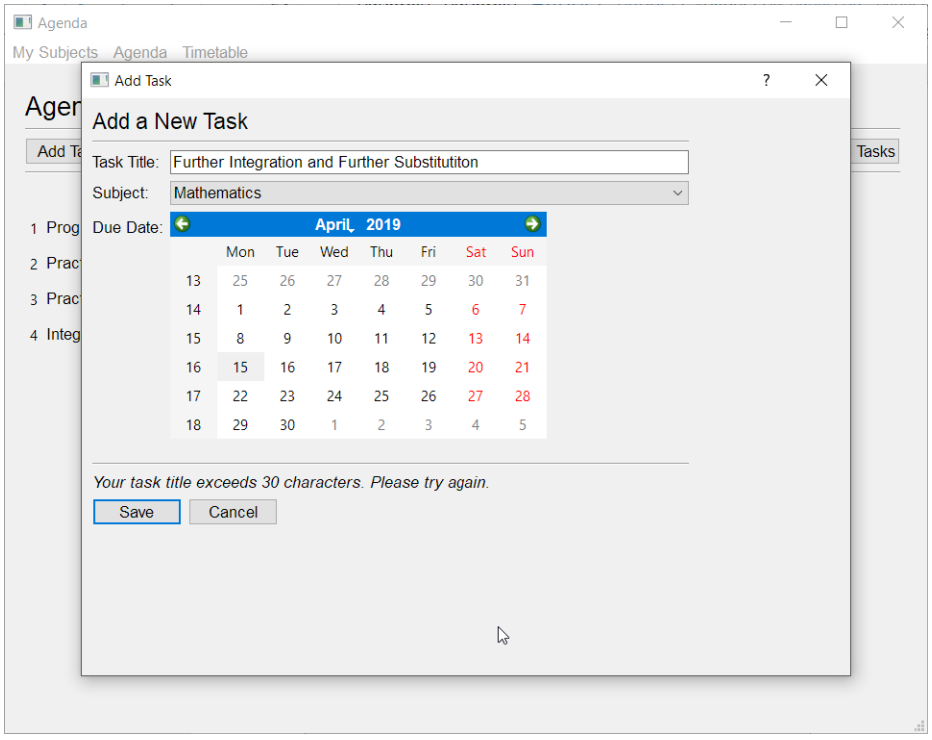
The screenshots above show that test #5, test #7, and test #8 were successful. It also shows that test #18 was unsuccessful.

When the user presses the button to add a new task, a dialog window is opened which provides the user with a line edit widget to enter their task title, a combo box for their subjects (with the options read from the .txt file which stores the user's subjects), and a graphical calendar widget for the due date.

There is no way for the user to add additional notes about their task.



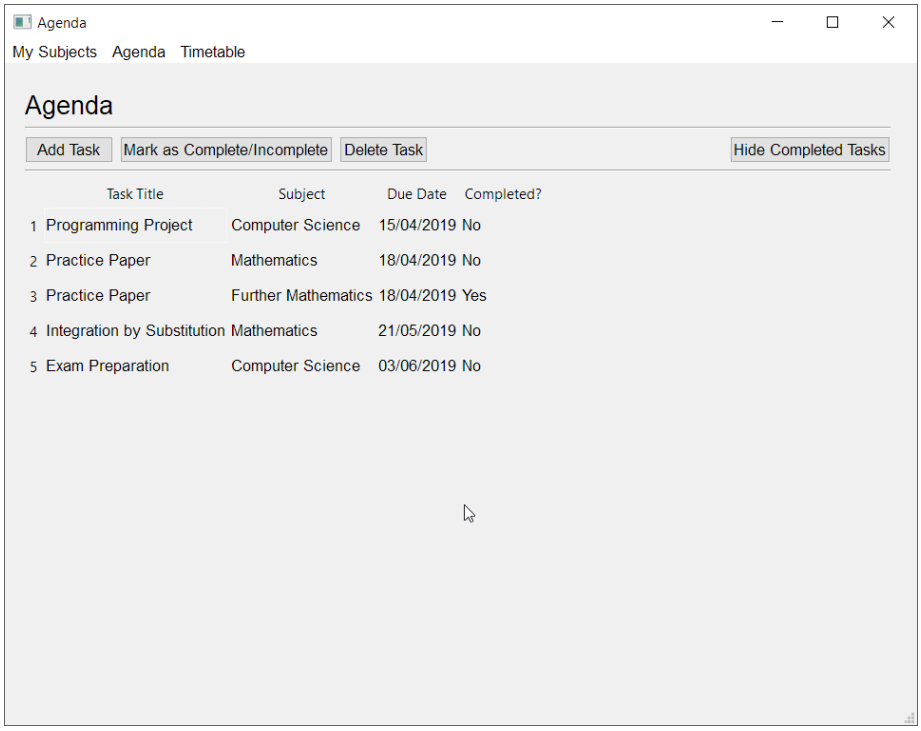




The three screenshots above show that test #6, test #15, and test #16 were successful.

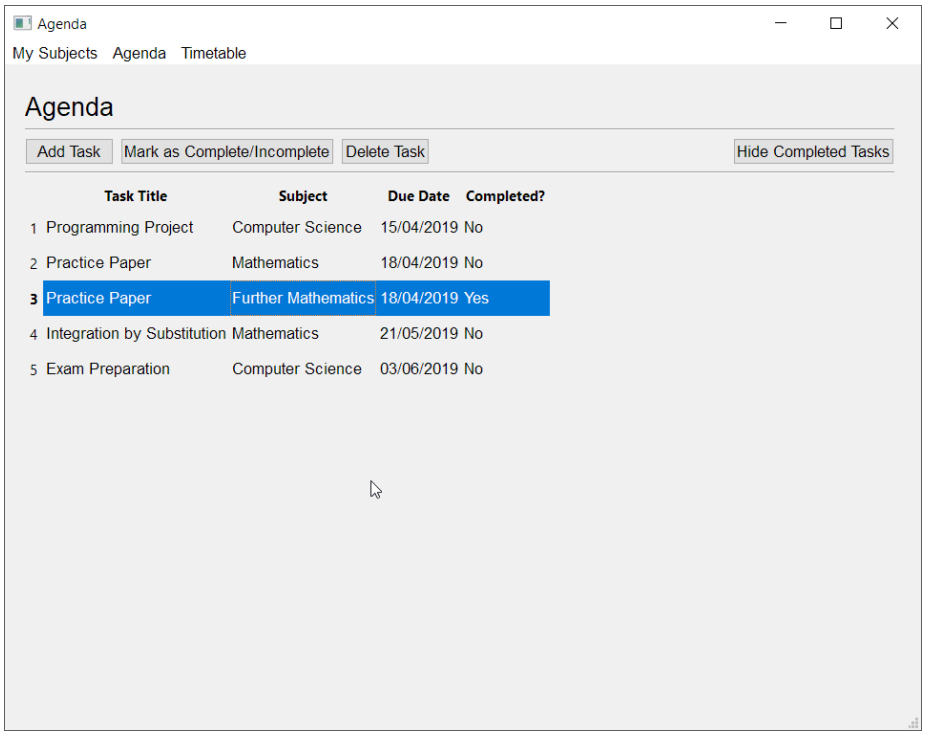
When the user enters a task title which is either empty (like in the first screenshot) or only consists of whitespace (like in the second screenshot), the instruction label is changed to specify that they have not entered a task title, and the dialog window is not closed, which shows that their task has not been added to the agenda.

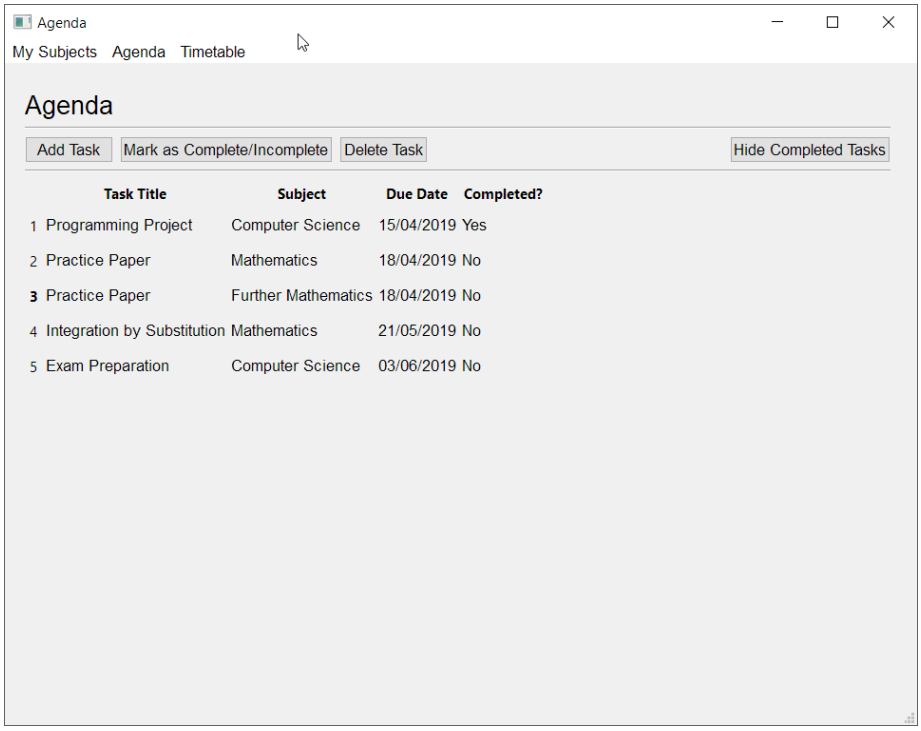
When the user enters a task title which exceeds 30 characters, the instruction label is changed to specify that their task title exceeds 30 characters, and the dialog window is not closed, which shows that their task has not been added to the agenda.



The screenshot above demonstrates that test #9 was successful.

The user entered a new task for exam preparation due on 03/06/2019, and it was automatically marked as incomplete.

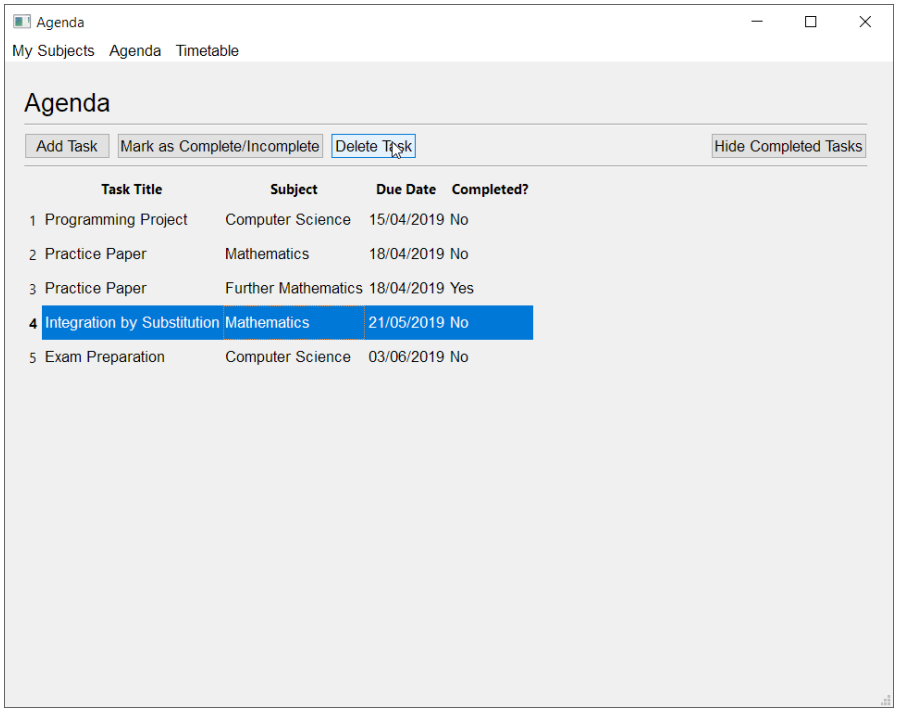


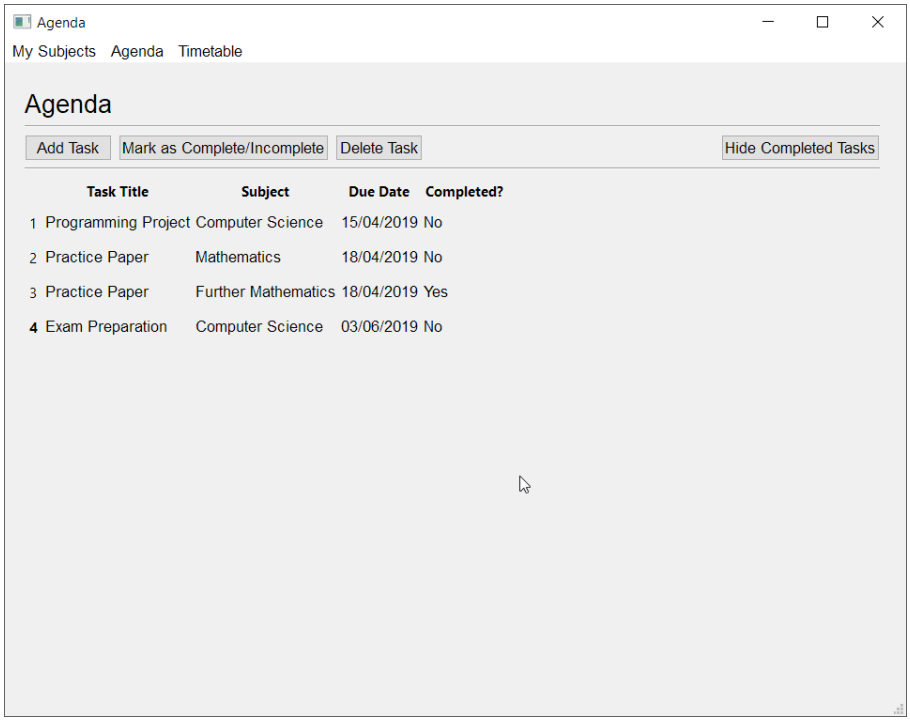


The screenshots above show that test #10 was successful.

When the user selected the first task and pressed the button to mark it as complete/incomplete, its completion status changed from ‘No’ to ‘Yes’.

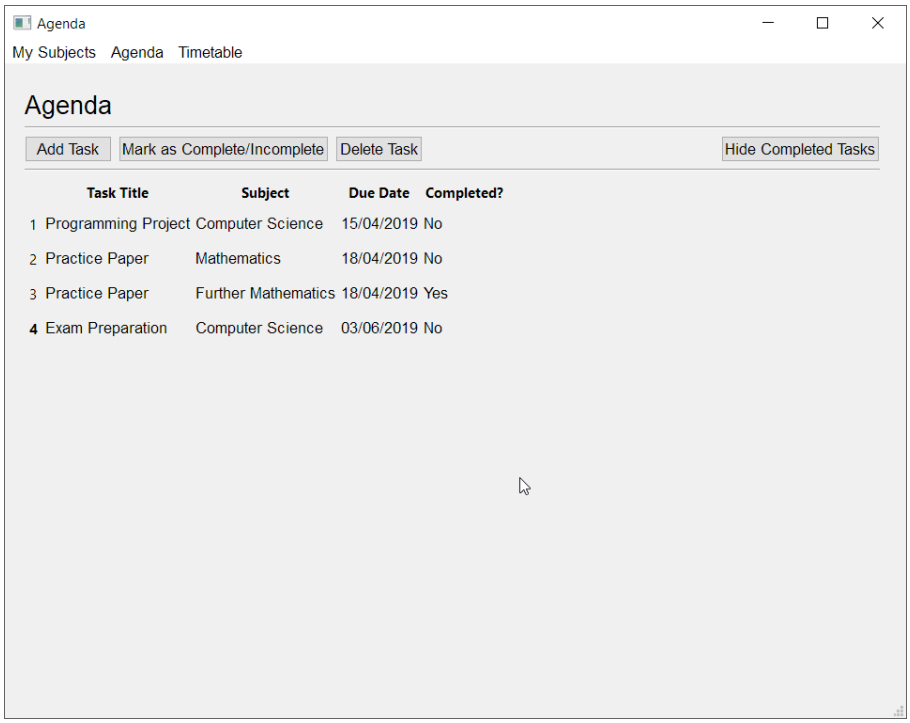
When the user selected the third task and pressed the button to mark it as complete/incomplete, its completion status changed from ‘Yes’ to ‘No’.





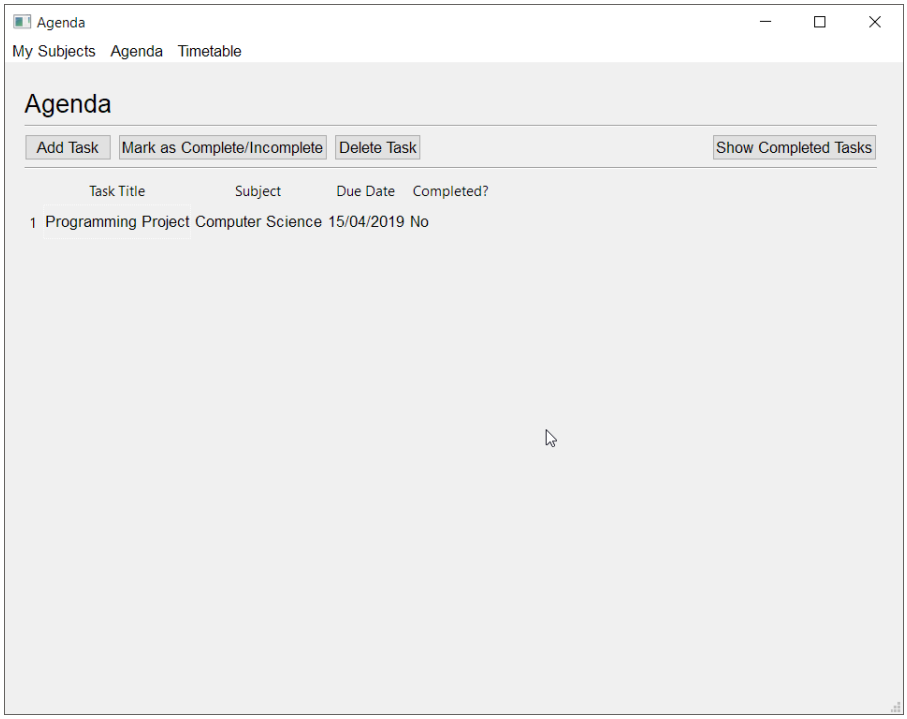
The two screenshots above show that test #11 was successful.

When the user selected the task with the title ‘Integration by Substitution’ and pressed the delete button, the task was deleted from the agenda.



The screenshot above shows that test #13 and test #14 were successful.

The user closed the program, then opened it again. When they opened it, the agenda from their previous session was displayed, which shows that it must have been stored in the .json file successfully.



The screenshot above shows that test #17 was successful.

The user manually deleted the list file, *task_list.json*, from their computer, and then started the program. When they added back their 'Programming Project' task, it was added to the agenda successfully.

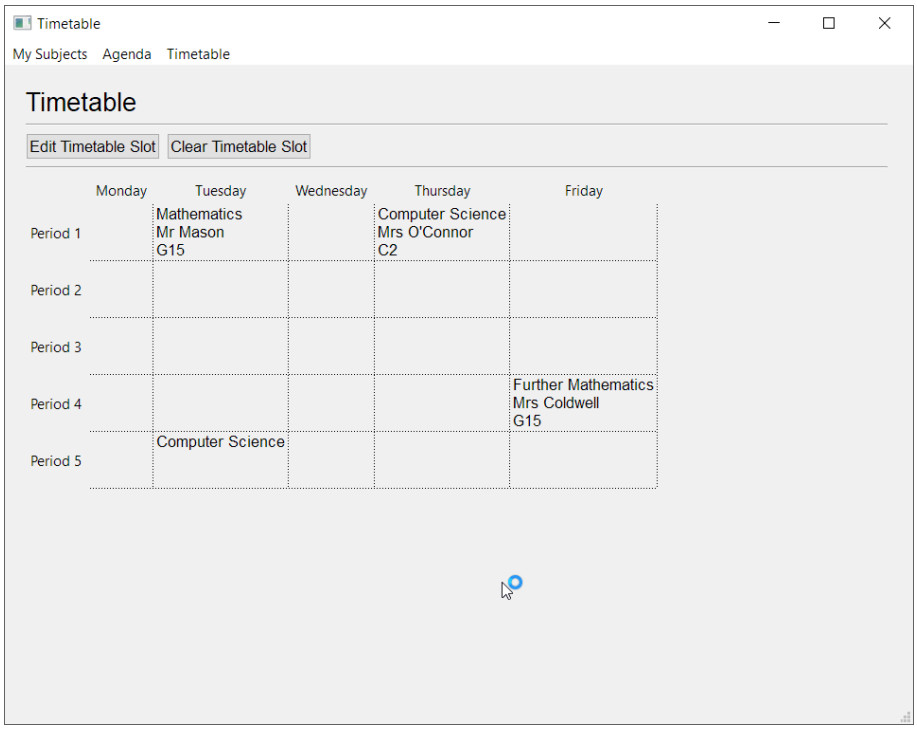
This shows that an empty list was created by the program, because the program would not be able to do this without a list.

Test Results: Timetable

Test #	Criterion	Criterion Met?
#1	Main window which displays a timetable of the user's lessons and their lesson details.	Yes
#2	Sleek, simple design which is easy to use.	Yes
#3	High information density.	Yes
#4	Ability to edit lesson in the timetable using a dialog window by selecting a timetable slot and pressing a button to edit it.	Yes
#5	Input for new task subject through a combo box which is linked to the subjects list in My Subjects.	Yes
#6	Input for teacher for a lesson validated by a length check.	Yes
#7	Input for room for a lesson validated by a length check.	Yes
#8	Updated timetable saved to a .json file for permanent storage whenever a change to the timetable is made.	Yes
#9	List of user's lessons and their lesson details loaded in from the .json file whenever the program is started.	Yes
#10	Rejects new lessons with a teacher input which exceeds 30 characters.	Yes
#11	Rejects new lessons with a room input which exceeds 20 characters.	Yes
#12	Creates timetable list if no timetable list exists (new user).	Yes
#13	Ability to edit the timetable template.	No

#14	Ability for the user to clear a lesson in the timetable by selecting a timetable slot and pressing a button to clear it.	Yes

Test Evidence: Timetable



The screenshot above demonstrates that test #1, test #2, test #3, and test #9 were successful. It also shows that test #13 was unsuccessful.

The main window displays a timetable which shows the user's lessons. In each lesson, it shows the subject, and teacher and room if applicable.

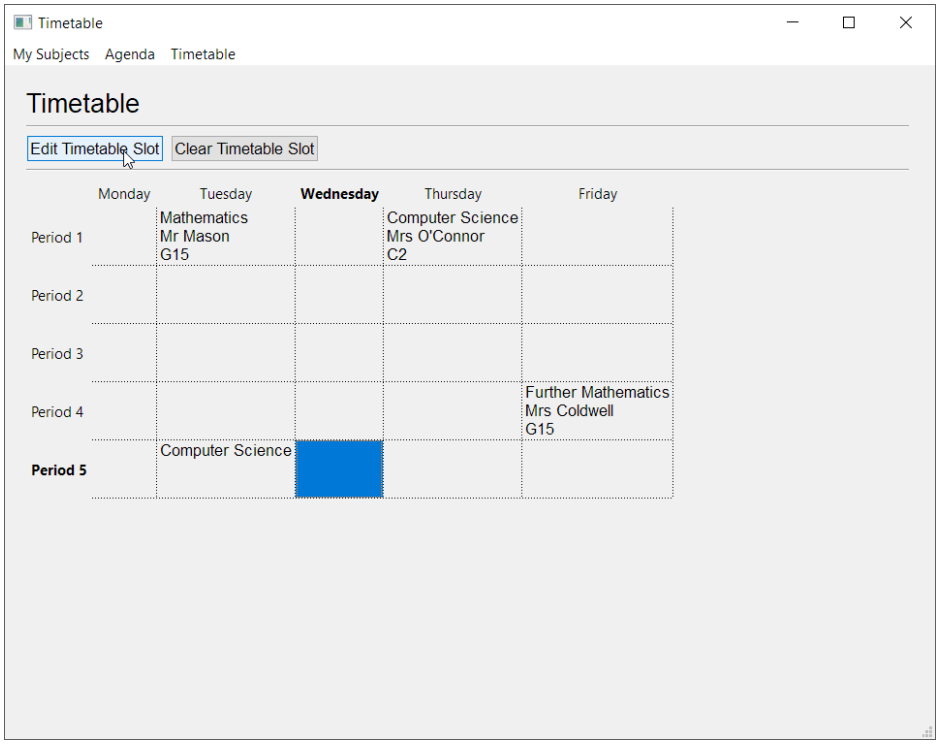
The design is sleek and simple; it contains only the necessary information, and the timetable rows and columns are resized to fit the contents exactly. The cells can be seen easily due to the dotted lines separating them, which is easier on the eyes than using full lines for separation.

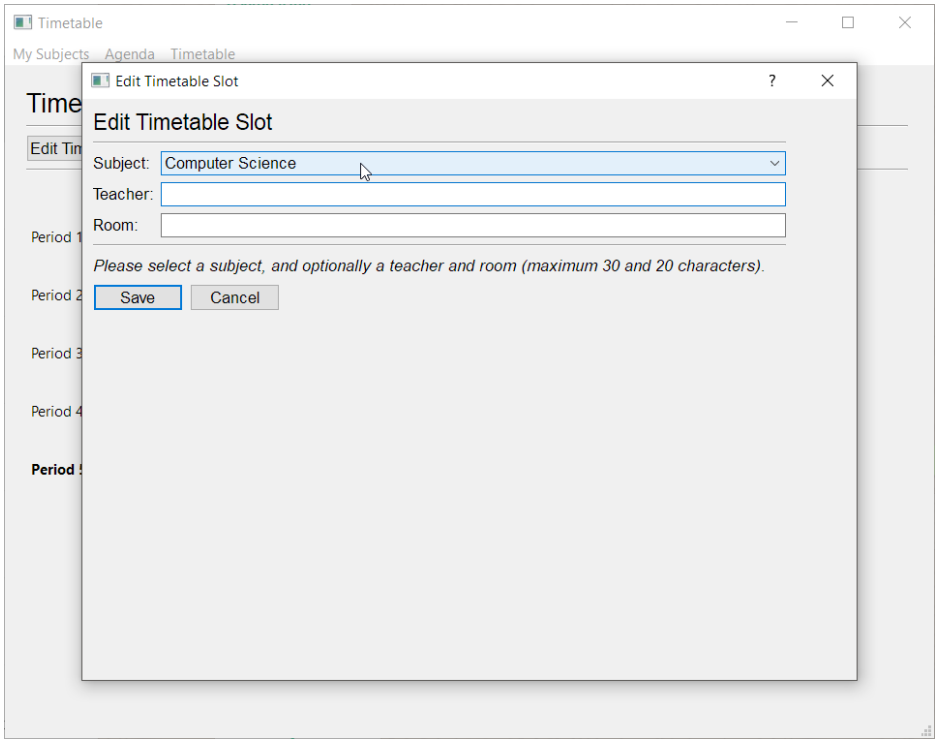
The user interface is easy to use, with the buttons being labelled to describe their purpose.

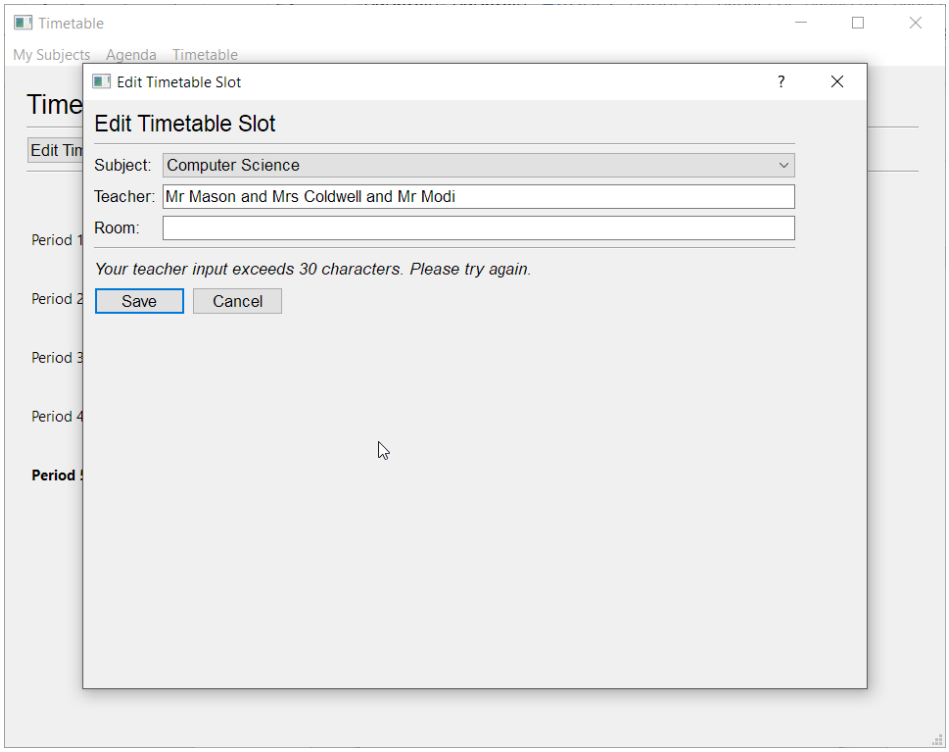
Information is as densely packed as possible due to the resizing of columns and rows to fit contents.

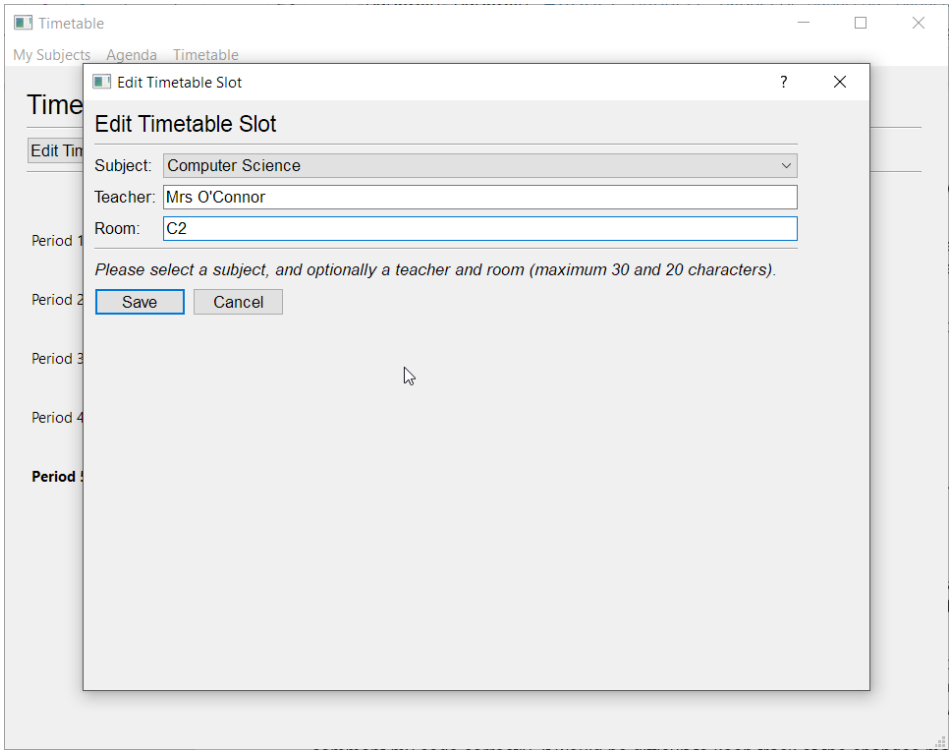
The user loaded the program and it populated their timetable with their lessons from their previous session on the program. This shows that it must have loaded the lessons in from the .json file, as permanent storage cannot be achieved through storage in variables.

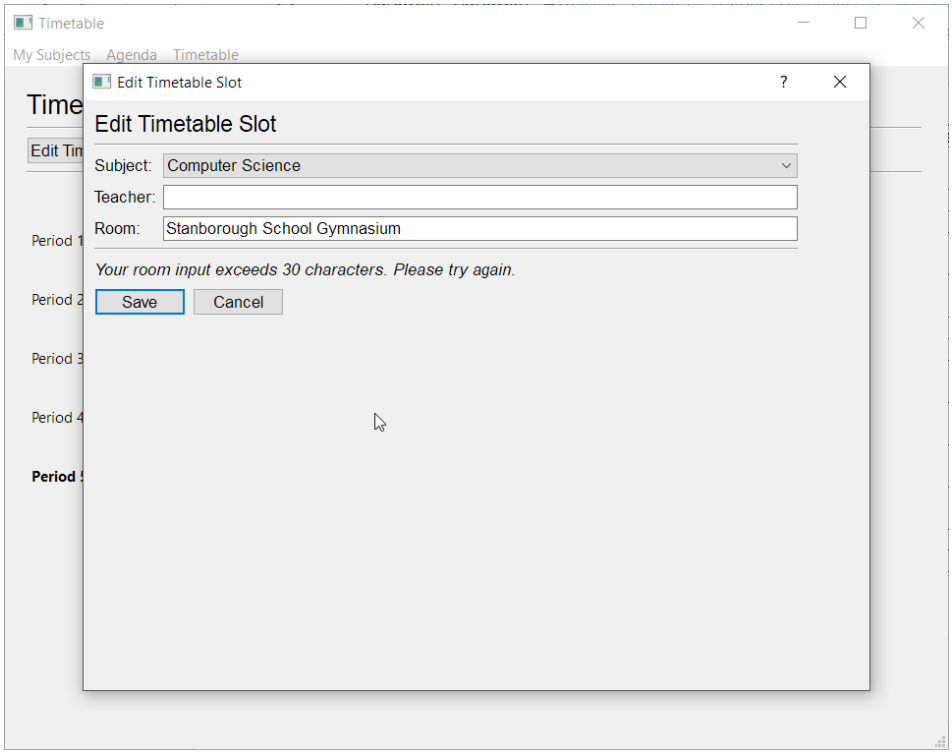
There is no button or option to change the timetable template.


Timetable
My Subjects
Agenda
Timetable
Timetable
Edit Timetable Slot
Clear Timetable Slot
Monday
Tuesday
Wednesday
Thursday
Friday
Period 1
Mathematics
Mr Mason
G15
Computer Science
Mrs O'Connor
C2
Period 2
Period 3
Period 4
Further Mathematics
Mrs Coldwell
G15
Period 5
Computer Science


Timetable
My Subjects
Agenda
Timetable
Edit Timetable Slot
Edit Timetable Slot
Subject:
Computer Science
Teacher:
Room:
Please select a subject, and optionally a teacher and room (maximum 30 and 20 characters).
Save
Cancel


Timetable
My Subjects Agenda Timetable
Edit Timetable Slot
Edit Timetable Slot
Subject: Computer Science
Teacher: Mr Mason and Mrs Coldwell and Mr Modi
Room:
Your teacher input exceeds 30 characters. Please try again.
Save
Cancel


Timetable
My Subjects Agenda Timetable
Edit Timetable Slot
Edit Timetable Slot
Subject: Computer Science
Teacher: Mrs O'Connor
Room: C2
Please select a subject, and optionally a teacher and room (maximum 30 and 20 characters).
Save
Cancel


Timetable
My Subjects
Agenda
Timetable
Edit Timetable Slot
Edit Timetable Slot
Subject:
Computer Science
Teacher:
Room:
Stanborough School Gymnasium
Your room input exceeds 30 characters. Please try again.
Save
Cancel

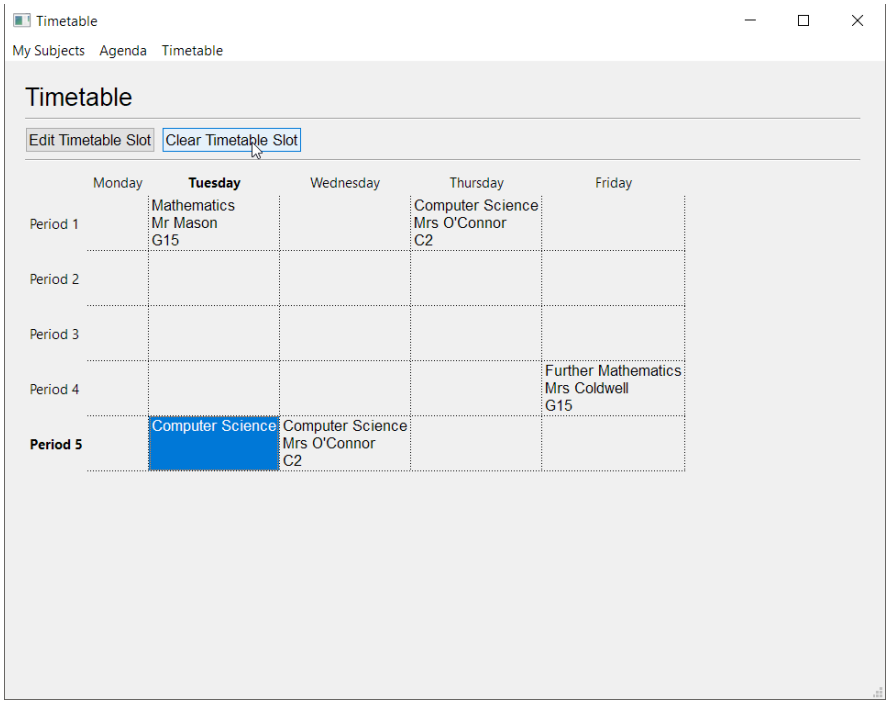
The screenshots above show that test #4, test #5, test #6, test #7, test #10 and test #11 were successful.

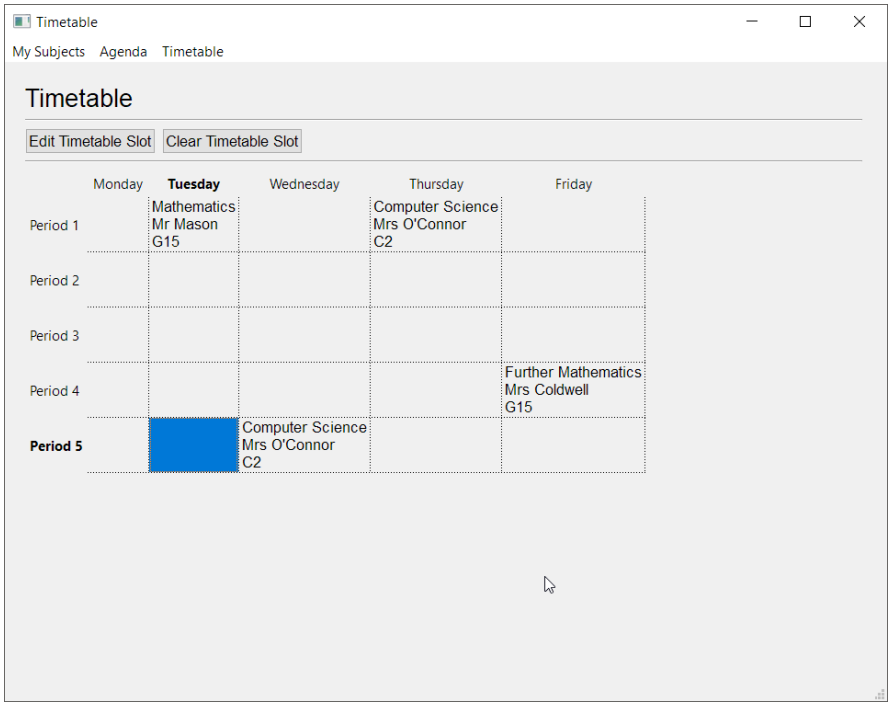
When the user selects a timetable slot and clicks on the button to edit that slot, a dialog window is opened. In that dialog window, the user can select their subject from a combo box, and optionally enter a teacher and room for that lesson.

If the user's teacher input exceeds 30 characters, as in the second screenshot, the instruction label changes to specify that their teacher input exceeds 30 characters, and the dialog window is not closed, which shows that their timetable slot is not updated.

Likewise, if the user's room input exceeds 20 characters, as in the third screenshot, the instruction label changes to specify that their teacher input exceeds 30 characters, and the dialog window is not closed, which shows that their timetable slot is not updated.

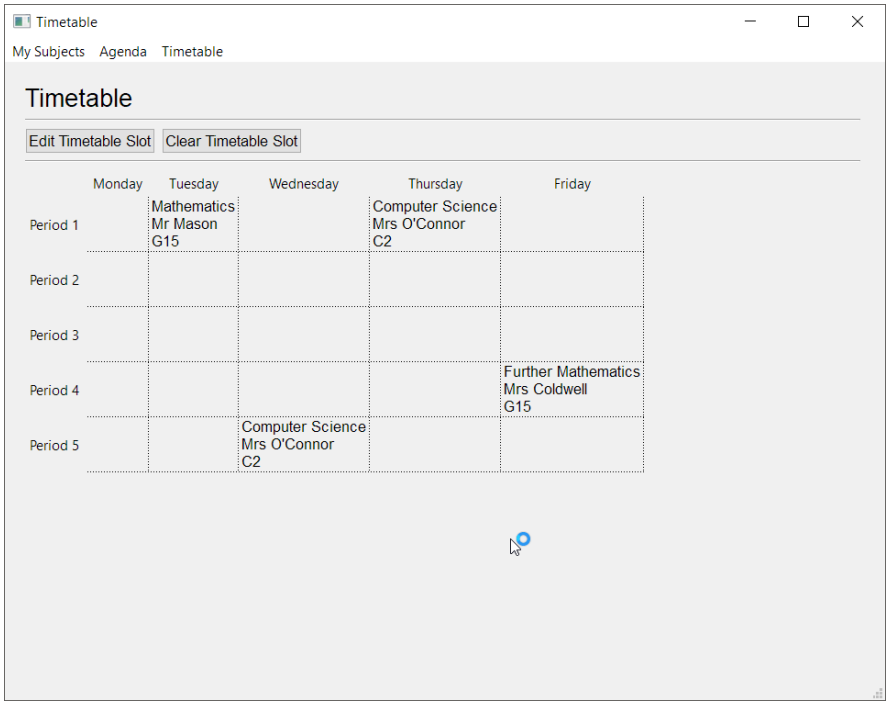
After they enter a valid input (like in the fourth screenshot) and save the lesson, their selected timetable slot is updated with the lesson details.





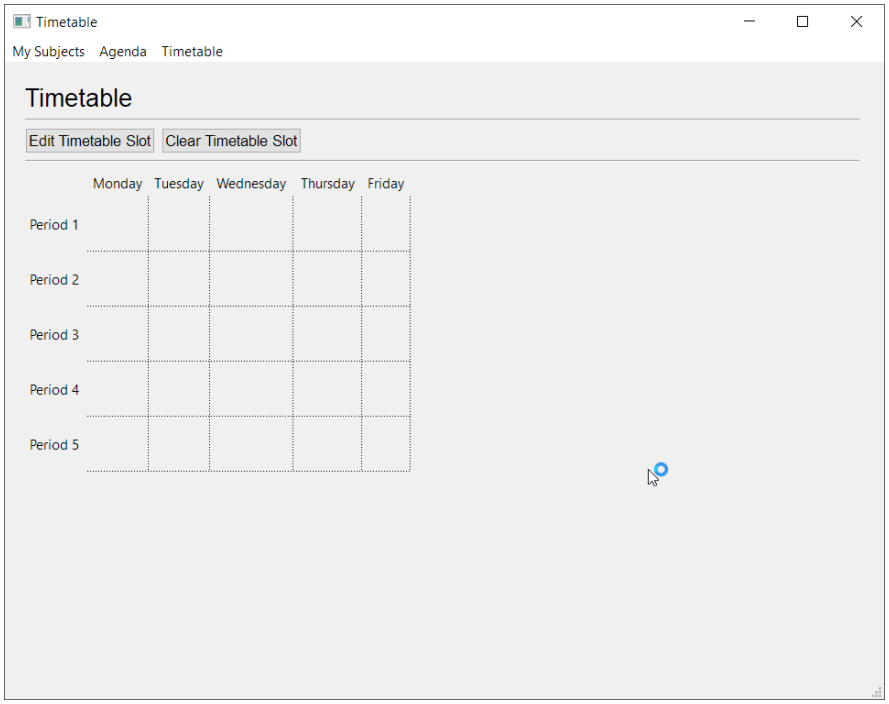
The couple of screenshots above show that test #14 was successful.

When the user selects a timetable slot and presses a button to clear it on the top left (the second button), their selected timetable slot is cleared from the lesson in it.



The screenshot above shows that test #8 and test #9 were successful.

When the user closed the program and opened it again, the timetable from their previous session was displayed, which shows that it was stored permanently in the .json file.



The screenshot above shows that test #12 was successful.

When the user deleted the *timetable.json* file permanently and opened the program, an empty timetable was displayed. This shows that the program created an empty list, as the program would not be able to populate the timetable with empty lessons if there was no timetable list.

Successfulness of Solution

Success Criteria Evaluation

By using the test data from the user acceptance testing, I have been able to evaluate the successfulness of my solution according to the success criteria I set in the Analysis section.

Requirement	**Requirement Met?**
The user must be able to enter their subjects and assign them a colour, which should both be stored in data files. This is because these subjects will be needed for the dropdown menu when adding a task to the agenda, and the colours will be displayed alongside the task on the agenda to help the user distinguish between the tasks from various subjects.	Mostly. The user is able to enter their subjects, as shown in test #5 of the user acceptance testing for Agenda, but colour coding is not used for these subjects. This is based on the feedback from the survey at the end of testing iteration #2 in stage 1, where my primary stakeholders (sixth form students) stated that it would detract too much from the aesthetics of the user interface. Colour coding is likely to be inappropriate for the aesthetic style of a desktop application, but this would be a good idea for future development of a mobile application equivalent, as it would better suit the design style. Further functionality has been implemented in My Subjects, with the user able to delete existing

	subjects from the list (as seen in the user acceptance test #7 for My Subjects), as this is a feature which would be useful to my users, who may switch subjects, such as when dropping one A-Level subject after Year 12.
The user must be able to add a task to the agenda with a title, subject, due date, and additional notes, which should all be stored in data files. This is so that the user can view a list of their tasks, and expand tasks from this list to view details about the task to help organise their workload.	Mostly. The user is unable to input additional notes about their tasks, which is shown in test #18 and test #19 during the user acceptance testing for Agenda. However, the user is able to add tasks to the agenda with a title, subject, and due date, as shown in test #5 of user acceptance testing for Agenda. In some ways, the lack of additional notes about tasks means that Agenda is simpler to navigate and read information from, as information is denser, and there is no need to view tasks in a separate window. All the information about their tasks can be read from the *QTableWidget* in the main window of Agenda. Despite this, it would be a high priority feature for future development to develop the ability for notes to be able to be added about tasks, as it would still prove to be an extremely useful feature for some students.
The user must be able to select the subject which a task is delegated for through a dropdown menu. This is to improve the user experience, allowing the user to create tasks more quickly and with less mistakes.	Yes. The user is able to select the subject for tasks and timetable slots using a dropdown menu (also known as a combo box) which loads in the user's subjects from *subject_list.txt*. This was demonstrated in test #7 of user acceptance testing for Agenda. New subjects can be added to the dropdown menu, and subjects can be deleted from the dropdown menu in My Subjects (as seen in test #7 in user acceptance testing for My Subjects), which provides the user with additional control in an easy manner.
The user must be able to select a timetable template through a user interface, and enter subjects in individual cells of the table, which would be stored to data files. This would allow the user to create a timetable that shows the lessons the user will have on each day, on a period by period basis, to help the	Mostly. The user is not able to select a timetable template (mentioned in test #13 of user acceptance testing for Timetable), as certain timetable configurations would cause scaling issues in the application due to the amount of space the timetable would take. In addition, an extremely small number of sixth form students, my primary stakeholders, have a

user arrive to the correct lessons on time.	schedule which does not consist of five periods over the five weekdays, so this was not a large concern. Despite this, the user is able to not only enter the subjects they have in each slot of their timetable (shown in test #5 of user acceptance testing for Timetable), but they also have the option of entering their teacher and room for that timetable slot (test #6 and test #7 of user acceptance testing for Timetable). This can prevent students from forgetting which room they need to go to or the name of their teacher, both of which are especially useful for new students and students who are starting a new school year. The user is also able to clear timetable slots (demonstrated in test #14 of user acceptance testing for Timetable), which would be useful in cases where their lesson has been permanently rescheduled, or they have dropped a subject which means they no longer need to attend those lessons. Editing the timetable template could be looked into during future development, with different optional ways of displaying the timetable to overcome scrolling issues with larger timetables.
(Optionally, depending on time constraints) The user must be able to enter the grades which they achieve in a subject to a grade book, with the title of the topic, grade, date achieved, and feedback, which would be stored to data files. This would allow the user to access an overview of their grades at another time, so that they can see which topics they need to improve at.	No. Grade book was planned to be an extra part of the application in the first place, and I have surveyed sixth form students (my primary stakeholders). 90% of the respondents stated that grade book would not be used by them, and the remaining 10% stated that they would prefer other parts of the application to have more features instead. This was the case during development, as I developed the ability for the user to organise their subjects more effectively using a list widget in My Subjects, the ability for the user to hide/show completed tasks in Agenda (shown in test #12 in user acceptance testing for Agenda), and the ability for the user to add details about their teacher and room for each timetable slot in Timetable (shown in test #6 and test #7 in user acceptance testing for Timetable). In other words, I focused on improving the quality of existing programs instead of adding a new program to my application which would be very rarely used.

	However, Grade Book would be a feature I would look to implement, especially now that the higher priority extra features have been implemented.

Evaluation of Solution

In the user acceptance test results, a total of 41 out of 44 tests were recorded as successful. This shows that my application was meeting an extremely large majority of the test criteria, which was set based on the original design specification and requirements, as well as some tests set to test the robustness of the application.

In the user acceptance testing, every test for robustness was successful, and almost every functional test was successful. Out of these functional tests, all the fundamental parts of the design specification passed testing, which is a reason why I believe that my solution is successful.

For example, in all my programs which made up the application, I performed functional tests to ensure that all my user interfaces had a sleek, easy to use design, with high information density. All of my programs passed this test completely. This is especially important in my application, because my application is about the processing and organisation of data.

If the user interfaces were not aesthetically pleasing, it would be more difficult for the user to find the information they are looking for, such as tasks due in the next week.

In addition, if the user interfaces were not easy to use, then the user would spend too much time trying to work out how to perform simple actions such as adding a task to the agenda. This would negate the time they would have saved by organising their data electronically.

Meanwhile, a lack of high information density would make viewing information in my programs a slower process, which would negate the advantage of having their important data such as their school timetable stored in one place (on their computer).

Whilst not all success criteria were fully met, I would say that my student planner application is very successful, due to the reasons behind why some of the success criteria was not met.

In all cases where success criteria was not met, it was caused by a decision which I made purposefully after communicating with my primary stakeholders in sixth form students, as well as some younger students studying for their GCSEs.

One example of where success criteria was mostly met, but not fully met, was the timetable functionality. In this case, I did not implement the ability for the user to edit their timetable template, but this was not implemented because it would likely harm the usability of the timetable program. I found that my application would not be able to display a schedule in this timetable format on a single screen if it had many more periods and days, as it would require side scrolling, which is not possible on almost every mouse. This means that it would only be laptop users who would be able to use their pointing device to scroll sideways (desktop users would be required to use arrow keys, which would be a lot more difficult).

In addition, during the development process, I performed research into GCSE and A-Level students which found that an extremely small number of students had a schedule which did not consist of five periods over five weekdays. These two factors provided me a good reason to avoid implementing functionality for the user to edit their timetable template from the 5 x 5 format.

Grade book was the only example of a success criterion which was not met at all. It was already planned as an optional feature during the Design phase, but this decision was entirely down to a survey which I took from sixth form students. In this survey, I found that 90% of sixth form students would not use Grade Book at all, and the remaining 10% of sixth form students would prefer existing programs in Student Planner to be developed further to provide more features. Hence, I took this into consideration, and added more features to My Subjects, Agenda, and Timetable instead.

By taking these matters into account, overall, I would evaluate my student planner solution to be very successful, but not fully successful. Whilst I made the correct decisions based on the needs of my stakeholders, the decisions not to implement certain features could have been made earlier in the process of planning my student planner, as opposed to in the development phase. However, once I made these decisions to not implement some features, I spent the time on developing extra features which would be more useful to my stakeholders.

Limitations of My Application

There are some limitations in my existing application, so it is important to consider what these limitations are, and how my application could be developed to overcome these limitations in the future.

The most significant limitation of my existing application is that the user interface is not easy on the eyes for users using the program in dark environments. This is a common issue with other everyday applications such as web browsers.

To circumvent this, I could implement a dark mode across my application, which could be accessed through a settings menu. This would cause the colour palette of my application to be changed to have darker backgrounds and white text to contrast with this. The decreased contrast of the background with the user's environment would cause less eye strain, making it easier for students to use my application in darker environments, such as in the car.

Another limitation is that there are no accessibility features for blind users, meaning that they would not be able to use my application, as it is reliant on users being able to read the text in the application.

I could overcome this limitation by implementing a text-to-speech feature which could be enabled when the application detects that accessibility devices for blind users have been plugged into the device, such as braille keyboards. When the text-to-speech feature is enabled, the program would read out the text on the screen according to the user's input. For example, it may read the button labels, which would help inform the user about what buttons they could press.

Improvements in Future Development

There were a few features which I did not develop, as I listened to feedback from stakeholders which found that these features would not be a high priority for development when compared to extra features which I could develop, such as the ability to hide completed tasks in Agenda. However, in the future, some of these features could be developed, with measures taken to overcome the limitations of these potential changes.

In terms of improvements, adding the ability for the user to add notes about their tasks would be the top priority. As mentioned in the usability section, a text box widget would be used in the dialog window for adding a task to enable the user to input these notes. These notes would be added as an additional value to each dictionary in the list stored in *task_list.json*.

The limitation behind this improvement is how the user would view these notes, as it would take up too much space in the *QTableWidget*. I could develop around this limitation by creating an additional button in the main window for the user to 'View Notes' about a selected task. This would open a dialog window with the full details about that task, showing the task title, subject, due date, completion, and additional notes, similar to how it was implemented in Egenda (which was mentioned in the Competitor Research section).

Another minor limitation behind this improvement would be that users may not want this feature. Hence, I could add this feature as an opt-in feature in a settings menu for my application.

Implementing the Grade Book would be something which would have lower priority than adding notes, but it would still be something which could be important to some of my stakeholders, so it would be in my interests to develop it in the future. This would allow the user to enter records of tests and assessments they have taken, including the assessment title using a line edit widget, subject using a combo box, date of assessment using a calendar widget, grade using a combo box, and feedback using a text box widget.

The major limitation behind Grade Book would be the grading system. For example, in the current GCSE system, some subjects are still using the old A* to F grading system. Furthermore, teachers may set small class tests which do not provide the user with a grade, only a numerical score from that test. The best way to overcome this limitation would be to add an option in the combo box for 'Other'. If this is selected, a line edit widget could appear, which would enable the user to manually type in their grade/score. This would not be ideal solution, but it would not need to be used by the user very often, and it would be better than alternative solutions.

Maintenance

The maintenance of my application in the future has been carefully considered throughout this programming project through various means.

Screenshots, Annotations, and Comments

Screenshots, annotations, and comments aid future maintenance by making it easier to understand the code for the programs I have written, as it is important to understand the existing code before making updates to programs.

An example of where screenshotting, annotations, and comments will help greatly in understanding the code is in development iteration #2 of stage 6. In this iteration, I made a lot of changes to the code from the previous iteration and previous stages, so it was important that it was documented appropriately. If I did not screenshot, annotate, and comment my code correctly, it would be difficult to keep track of the changes made, and it would likely be confusing to maintain in the future.

During the development of my programs, the operational code has been screenshotted from the view of the IDE's text editor, which ensures that the built-in colour coding and indentation highlighting features of the IDE can be observed for easier reading of the code.

The code has been broken down and annotated through the use of text, explaining what functions and calculations I have used to perform certain actions. This will make future maintenance easier because I will be able to understand why I chose to use particular solutions over others, and how the process of how these solutions work.

Within the code, I have included comments, which serve as a more concise explanation about what each block of code does. This also makes future maintenance easier by saving time which would otherwise be spent switching between the development logs and the text editor of the IDE.

Code Modularity

My application has been split into several modular programs to improve the organisation, reusability, and ease of testing of my code.

Examples of code modularity are how there are different programs for My Subjects and Agenda, and how My Subjects is split into multiple programs; one which is the automatically generated user interface code for the My Subjects main window, one for the automatically generated user interface code for the Add Subject dialog window, and one for the operational code.

By splitting the application into modules for each section, my application will be able to be developed further in an easy manner, because the chances of breaking one part of the application whilst developing another part of the program are lower. Even in cases where changing one module breaks the codebase, it is likely that only small changes will be needed to fix the issue.

Code reusability is improved through modular programming by enabling changes in programs to be made more easily, such as when making changes to the user interface's aesthetic layout without changing the operational code. In this case, generating the code using pyuic5 will overwrite the code, so the additional operational code would be lost if the automatically generated code and the operational code weren't saved as separate programs. By separating these two types of code with a modular programming approach, I will be able to make changes to the aesthetics of the user interface without losing my operational code.

Modular programming can also improve code reusability when used alongside the use of classes because new classes can inherit the attributes and methods of other classes. This means that newly written code can be shorter, and only the unique attributes and methods need to be declared.

Whilst having modular code is largely advantageous, there is the drawback that having so many more programs can make things seem more complicated initially. This means that maintaining my application would be difficult if it has not been worked on for a few months.

One solution would be to ensure that the application is being constantly updated and maintained. However, this is time consuming, and it may not be necessary if the application meets all the needs of the stakeholders.

Data Atomicity

Data atomicity is a major example of the good coding practices I have implemented in my application. It improves the robustness of my code, as my program is able to handle crashes with minimal data loss.

For example, when sorting the subject list in My Subjects, the data must be overwritten to a file. This causes the original data in the .txt file to be wiped before new data is written, which can result in data loss if the program crashes during the overwriting of the file. To work around this, I sorted the subject list, then wrote it to a temporary text file, *subject_list_temp.txt*, before writing it to the original text file, *subject_list.txt*.

This means that if the program crashes during the writing of data, the original unsorted data will be retained in one of the files. If I did not code my program with the concept of data atomicity, the original unsorted data would be lost during a crash.

Maintenance is improved through the concept of having data atomicity because it ensures that program crashes can be recovered from properly, as data would still be able to be obtained from either the temporary or original files.

The only downside of ensuring data atomicity in my programs is that the code in these programs is longer than it would be without. However, the system performance impact is negligible from this, as the code for writing the data to an extra text file does not include strenuous actions, so it is largely advantageous to include data atomicity.

Adherence to the PEP 8 Conventions for Python

By adhering to the PEP 8 conventions for Python, which introduces standards for aspects such as variable names and class names, my application will be coded more consistently.

Examples of how I have adhered to the PEP 8 conventions for Python in my application are the use of all lower case letters with underscores as spaces for naming variables (in the form *variable_name*), and camel case with no underscores (in the form *VariableName*) for naming classes.

Consistency of coding throughout my application will help future maintenance because it helps make my code in programs easier to read and understand. The different naming conventions for variables and classes helps the reader differentiate between the two, so they will be able to see what is happening in the code more quickly, and how they can potentially make changes to the code.

One issue related to the adherence to these conventions is that there are no stated conventions on how you should be naming variables and classes (only the conventions for structure of names), so it is currently up to the developer to use their intuition to ensure that variables and classes are named clearly and concisely. Lack of intuition could lead to unnecessarily long variable and class names, which would make the programs more difficult to understand.

To account for this, it may be a good idea to create a written record of naming conventions for how names should be chosen, which would complement the existing PEP 8 conventions for the structure of names. However, this would be difficult to produce in a way which still allows for flexibility of development, and it would also be time consuming, so it may take a lower priority compared to other potential improvements in development such as developing new features.

Built-In Console Debugging

My application has been coded to print some information in the console when it is running. This improves the ease of maintenance of my application, because it makes it easier for developers to track what is occurring in the backend, so they can perform backtracking if there are any problems.

For example, when *my_subjects.py* is executed, the program will print the contents of *subject_list.txt* and the length of this list. This helps the developer by telling them what they should expect to be displayed in the *QListWidget*, and how many items are expected to be in the list widget.

By being able to backtrack, the developer is able to more easily discover the root cause of the problem, as they will be able to follow steps which led to the problem. In the example I gave, the developer would be able to know what part of the subject list is being displayed incorrectly if the *QListWidget* was not being displayed properly. Without console debugging built-in to the program, the developer would have to manually open *subject_list.txt*, which would be more time consuming.

The downside of this maintenance feature is that it adds more code to programs, and it may confuse the user if they see the console debugging.

However, these are insignificant concerns, because printing a few lines each time will have a negligible performance impact on the user's system, and the user is unlikely to be viewing the terminal, as they will be interacting with the application using the graphical user interface.

Completed Code

Throughout the development process, many versions of different programs for my application were created, so I have included screenshots of the final completed versions of the programs below. Together, these programs form the student planner application.

Each program has been displayed on a new page for easier navigation of this documentation.

My Subjects

my_subjects.py

```python
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QDialog, QMainWindow

from add_subject_setup import Ui_dialog_new_subject
from my_subjects_setup import Ui_mwindow_my_subjects

# Sets up the Add Subject dialog.


class AddSubjectDialog(QDialog, Ui_dialog_new_subject):
    def __init__(self):
        super().__init__()
        self.setupUi(self)

# Sets up the My Subjects main window.


class MySubjectsWindow(QMainWindow, Ui_mwindow_my_subjects):
    def __init__(self):
        super().__init__()
        self.setupUi(self)

        # Connects 'Add Subject' button to the Add Subject dialog.
        self.btn_add_subject.clicked.connect(self.open_dialog_add_subject)

        # Connects 'Delete Subject' button to the method for deleting the selected subject.
        self.btn_delete_subject.clicked.connect(self.delete_subject)

        # Populates the list widget on window startup.
        with open("subject_list.txt", "r") as data_file:
            subject_list = data_file.readlines()
            print(subject_list)
            print(len(subject_list))
            for line in subject_list:
                self.list_widget_my_subjects.addItem(line.strip("\n"))
            self.list_widget_my_subjects.sortItems()
            self.sort_subject_list()
```

```python
    # Opens the dialog for the user to add a subject.
    def open_dialog_add_subject(self):
        self.Dialog = AddSubjectDialog()
        # Connects 'Save' button to save the user input.
        self.Dialog.button_box_new_subject.accepted.disconnect()
        self.Dialog.button_box_new_subject.accepted.connect(self.save_subject)
        self.Dialog.open()

    # Deletes the selected subject.
    def delete_subject(self):
        selected_item = self.list_widget_my_subjects.selectedItems()
        for item in selected_item:
            self.list_widget_my_subjects.takeItem(
                self.list_widget_my_subjects.row(item))
        self.save_subject_list()

    # Saves the subject list.
    def save_subject_list(self):
        with open("subject_list_temp.txt", "w") as outfile:
            for i in range(self.list_widget_my_subjects.count()):
                subject = self.list_widget_my_subjects.item(i).text()
                outfile.write(subject + "\n")
            with open("subject_list.txt", "w") as outfile:
                for i in range(self.list_widget_my_subjects.count()):
                    subject = self.list_widget_my_subjects.item(i).text()
                    outfile.write(subject + "\n")

    # Sorts the subject list text file alphanumerically.
    def sort_subject_list(self):
        with open("subject_list.txt", "r") as outfile:
            lines = outfile.readlines()
            lines.sort()
            with open("subject_list_temp.txt", "w") as outfile:
                for line in lines:
                    outfile.write(line)
                with open("subject_list.txt", "w") as outfile:
                    for line in lines:
                        outfile.write(line)
```

```
77
78      # Saves input to a .txt file for the list of subjects.
79      def save_subject(self):
80          new_subject_name = self.Dialog.line_edit_subject_name.text()
81          if len(new_subject_name) <= 30 and len((new_subject_name).strip(" ")) > 0:
82              print(new_subject_name)
83              self.list_widget_my_subjects.addItem(new_subject_name)
84              self.list_widget_my_subjects.sortItems()
85              with open('subject_list.txt', 'a') as outfile:
86                  outfile.write(new_subject_name + "\n")
87              self.sort_subject_list()
88              self.Dialog.close()
89          elif len((new_subject_name).strip(" ")) == 0:
90              self.Dialog.lbl_instruction.setText(
91                  "You have not entered a subject name. Please try again.")
92          else:
93              self.Dialog.lbl_instruction.setText(
94                  "Your subject name exceeds 30 characters. Please try again.")
95
96
97  if __name__ == "__main__":
98      import sys
99      app = QtWidgets.QApplication(sys.argv)
100     mwindow_my_subjects = MySubjectsWindow()
101     mwindow_my_subjects.show()
102     sys.exit(app.exec_())
103
```

my_subjects_setup.py

```python
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'my_subjects.ui'
#
# Created by: PyQt5 UI code generator 5.12.1
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_mwindow_my_subjects(object):
    def setupUi(self, mwindow_my_subjects):
        mwindow_my_subjects.setObjectName("mwindow_my_subjects")
        mwindow_my_subjects.resize(960, 720)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        mwindow_my_subjects.setFont(font)
        self.central_widget = QtWidgets.QWidget(mwindow_my_subjects)
        self.central_widget.setObjectName("central_widget")
        self.gridLayout = QtWidgets.QGridLayout(self.central_widget)
        self.gridLayout.setObjectName("gridLayout")
        self.scrl_area_my_subjects_1 = QtWidgets.QScrollArea(
            self.central_widget)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.scrl_area_my_subjects_1.setFont(font)
        self.scrl_area_my_subjects_1.setFrameShape(QtWidgets.QFrame.NoFrame)
        self.scrl_area_my_subjects_1.setLineWidth(0)
        self.scrl_area_my_subjects_1.setWidgetResizable(True)
        self.scrl_area_my_subjects_1.setObjectName("scrl_area_my_subjects_1")
        self.scrl_area_my_subjects_2 = QtWidgets.QWidget()
        self.scrl_area_my_subjects_2.setGeometry(QtCore.QRect(0, 0, 938, 648))
        self.scrl_area_my_subjects_2.setObjectName("scrl_area_my_subjects_2")
        self.layoutWidget = QtWidgets.QWidget(self.scrl_area_my_subjects_2)
        self.layoutWidget.setGeometry(QtCore.QRect(11, 11, 531, 611))
        self.layoutWidget.setObjectName("layoutWidget")
        self.vert_layout_my_subjects = QtWidgets.QVBoxLayout(self.layoutWidget)
        self.vert_layout_my_subjects.setContentsMargins(0, 0, 0, 0)
        self.vert_layout_my_subjects.setObjectName("vert_layout_my_subjects")
        self.lbl_my_subjects = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setPointSize(16)
        self.lbl_my_subjects.setFont(font)
        self.lbl_my_subjects.setObjectName("lbl_my_subjects")
        self.vert_layout_my_subjects.addWidget(self.lbl_my_subjects)
        self.hori_line_my_subjects = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_my_subjects.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_my_subjects.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_my_subjects.setObjectName("hori_line_my_subjects")
        self.vert_layout_my_subjects.addWidget(self.hori_line_my_subjects)
        self.hori_layout_buttons = QtWidgets.QHBoxLayout()
        self.hori_layout_buttons.setObjectName("hori_layout_buttons")
        self.btn_add_subject = QtWidgets.QPushButton(self.layoutWidget)
        self.btn_add_subject.setObjectName("btn_add_subject")
        self.hori_layout_buttons.addWidget(
            self.btn_add_subject, 0, QtCore.Qt.AlignLeft)
        self.btn_delete_subject = QtWidgets.QPushButton(self.layoutWidget)
        self.btn_delete_subject.setAutoDefault(False)
        self.btn_delete_subject.setDefault(False)
        self.btn_delete_subject.setFlat(False)
        self.btn_delete_subject.setObjectName("btn_delete_subject")
        self.hori_layout_buttons.addWidget(self.btn_delete_subject)
        spacerItem = QtWidgets.QSpacerItem(
            40, 20, QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Minimum)
        self.hori_layout_buttons.addItem(spacerItem)
        self.vert_layout_my_subjects.addLayout(self.hori_layout_buttons)
        self.lbl_instruction = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setItalic(True)
        self.lbl_instruction.setFont(font)
        self.lbl_instruction.setObjectName("lbl_instruction")
        self.vert_layout_my_subjects.addWidget(self.lbl_instruction)
        self.hori_line_add_subject = QtWidgets.QFrame(self.layoutWidget)
```

```
        self.hori_line_add_subject.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_add_subject.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_add_subject.setObjectName("hori_line_add_subject")
        self.vert_layout_my_subjects.addWidget(self.hori_line_add_subject)
        self.list_widget_my_subjects = QtWidgets.QListWidget(self.layoutWidget)
        palette = QtGui.QPalette()
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active,
                         QtGui.QPalette.WindowText, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 255, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Button, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 255, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Light, brush)
        brush = QtGui.QBrush(QtGui.QColor(247, 247, 247, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Midlight, brush)
        brush = QtGui.QBrush(QtGui.QColor(120, 120, 120, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Dark, brush)
        brush = QtGui.QBrush(QtGui.QColor(160, 160, 160, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Mid, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Text, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 255))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active,
                         QtGui.QPalette.BrightText, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active,
                         QtGui.QPalette.ButtonText, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.NoBrush)
```

```
        palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Base, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 255, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Window, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Shadow, brush)
        brush = QtGui.QBrush(QtGui.QColor(247, 247, 247, 127))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active,
                         QtGui.QPalette.AlternateBase, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 220))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active,
                         QtGui.QPalette.ToolTipBase, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active,
                         QtGui.QPalette.ToolTipText, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0, 128))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active,
                         QtGui.QPalette.PlaceholderText, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive,
                         QtGui.QPalette.WindowText, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 255, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Button, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 255, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Light, brush)
        brush = QtGui.QBrush(QtGui.QColor(247, 247, 247, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive,
                         QtGui.QPalette.Midlight, brush)
        brush = QtGui.QBrush(QtGui.QColor(120, 120, 120, 0))
```

```
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Dark, brush)
        brush = QtGui.QBrush(QtGui.QColor(160, 160, 160, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Mid, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Text, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 255))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive,
                         QtGui.QPalette.BrightText, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive,
                         QtGui.QPalette.ButtonText, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.NoBrush)
        palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Base, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 255, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Window, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Shadow, brush)
        brush = QtGui.QBrush(QtGui.QColor(247, 247, 247, 127))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive,
                         QtGui.QPalette.AlternateBase, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 220))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive,
                         QtGui.QPalette.ToolTipBase, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive,
                         QtGui.QPalette.ToolTipText, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0, 128))
```

```
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive,
                         QtGui.QPalette.PlaceholderText, brush)
        brush = QtGui.QBrush(QtGui.QColor(120, 120, 120, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled,
                         QtGui.QPalette.WindowText, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 255, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Button, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 255, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Light, brush)
        brush = QtGui.QBrush(QtGui.QColor(247, 247, 247, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled,
                         QtGui.QPalette.Midlight, brush)
        brush = QtGui.QBrush(QtGui.QColor(120, 120, 120, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Dark, brush)
        brush = QtGui.QBrush(QtGui.QColor(160, 160, 160, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Mid, brush)
        brush = QtGui.QBrush(QtGui.QColor(120, 120, 120, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Text, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 255))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled,
                         QtGui.QPalette.BrightText, brush)
        brush = QtGui.QBrush(QtGui.QColor(120, 120, 120, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled,
                         QtGui.QPalette.ButtonText, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.NoBrush)
        palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Base, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 255, 0))
```

```
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Window, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Shadow, brush)
        brush = QtGui.QBrush(QtGui.QColor(240, 240, 240, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled,
                         QtGui.QPalette.AlternateBase, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 220))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled,
                         QtGui.QPalette.ToolTipBase, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled,
                         QtGui.QPalette.ToolTipText, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0, 128))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled,
                         QtGui.QPalette.PlaceholderText, brush)
        self.list_widget_my_subjects.setPalette(palette)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.list_widget_my_subjects.setFont(font)
        self.list_widget_my_subjects.setAutoFillBackground(False)
        self.list_widget_my_subjects.setStyleSheet(
            "background-color: rgba(255, 255, 255, 0)")
        self.list_widget_my_subjects.setFrameShape(QtWidgets.QFrame.NoFrame)
        self.list_widget_my_subjects.setFrameShadow(QtWidgets.QFrame.Plain)
        self.list_widget_my_subjects.setObjectName("list_widget_my_subjects")
        self.vert_layout_my_subjects.addWidget(self.list_widget_my_subjects)
        self.scrl_area_my_subjects_1.setWidget(self.scrl_area_my_subjects_2)
        self.gridLayout.addWidget(self.scrl_area_my_subjects_1, 0, 0, 1, 1)
        mwindow_my_subjects.setCentralWidget(self.central_widget)
        self.menu_bar = QtWidgets.QMenuBar(mwindow_my_subjects)
        self.menu_bar.setGeometry(QtCore.QRect(0, 0, 960, 25))
```

```
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_bar.setFont(font)
        self.menu_bar.setObjectName("menu_bar")
        self.menu_my_subjects = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_my_subjects.setFont(font)
        self.menu_my_subjects.setObjectName("menu_my_subjects")
        self.menu_agenda = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_agenda.setFont(font)
        self.menu_agenda.setObjectName("menu_agenda")
        self.menu_timetable = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_timetable.setFont(font)
        self.menu_timetable.setObjectName("menu_timetable")
        mwindow_my_subjects.setMenuBar(self.menu_bar)
        self.status_bar = QtWidgets.QStatusBar(mwindow_my_subjects)
        self.status_bar.setObjectName("status_bar")
        mwindow_my_subjects.setStatusBar(self.status_bar)
        self.actionAgenda = QtWidgets.QAction(mwindow_my_subjects)
        self.actionAgenda.setObjectName("actionAgenda")
        self.menu_bar.addAction(self.menu_my_subjects.menuAction())
        self.menu_bar.addAction(self.menu_agenda.menuAction())
        self.menu_bar.addAction(self.menu_timetable.menuAction())

        self.retranslateUi(mwindow_my_subjects)
        QtCore.QMetaObject.connectSlotsByName(mwindow_my_subjects)
```

```
302
303     def retranslateUi(self, mwindow_my_subjects):
304         _translate = QtCore.QCoreApplication.translate
305         mwindow_my_subjects.setWindowTitle(
306             _translate("mwindow_my_subjects", "My Subjects"))
307         self.lbl_my_subjects.setText(_translate(
308             "mwindow_my_subjects", "My Subjects"))
309         self.btn_add_subject.setText(_translate(
310             "mwindow_my_subjects", "Add Subject"))
311         self.btn_delete_subject.setText(_translate(
312             "mwindow_my_subjects", "Delete Subject"))
313         self.lbl_instruction.setText(_translate(
314             "mwindow_my_subjects", "To delete a subject, left-click on it and press the delete button above."))
315         self.menu_my_subjects.setTitle(_translate(
316             "mwindow_my_subjects", "My Subjects"))
317         self.menu_agenda.setTitle(_translate("mwindow_my_subjects", "Agenda"))
318         self.menu_timetable.setTitle(_translate(
319             "mwindow_my_subjects", "Timetable"))
320         self.actionAgenda.setText(_translate("mwindow_my_subjects", "Agenda"))
321
```

add_subject_setup.py

```python
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'add_subject.ui'
#
# Created by: PyQt5 UI code generator 5.12.1
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_dialog_new_subject(object):
    def setupUi(self, dialog_new_subject):
        dialog_new_subject.setObjectName("dialog_new_subject")
        dialog_new_subject.resize(800, 600)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        font.setItalic(False)
        dialog_new_subject.setFont(font)
        self.layoutWidget = QtWidgets.QWidget(dialog_new_subject)
        self.layoutWidget.setGeometry(QtCore.QRect(11, 11, 471, 160))
        self.layoutWidget.setObjectName("layoutWidget")
        self.vert_layout_new_subject = QtWidgets.QVBoxLayout(self.layoutWidget)
        self.vert_layout_new_subject.setContentsMargins(0, 0, 0, 0)
        self.vert_layout_new_subject.setObjectName("vert_layout_new_subject")
        self.lbl_new_subject = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setPointSize(14)
        self.lbl_new_subject.setFont(font)
        self.lbl_new_subject.setObjectName("lbl_new_subject")
        self.vert_layout_new_subject.addWidget(self.lbl_new_subject)
        self.hori_line_new_subject = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_new_subject.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_new_subject.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_new_subject.setObjectName("hori_line_new_subject")
        self.vert_layout_new_subject.addWidget(self.hori_line_new_subject)
        self.form_layout_subject_details = QtWidgets.QFormLayout()
        self.form_layout_subject_details.setObjectName(
            "form_layout_subject_details")
        self.lbl_subject_name = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_subject_name.setObjectName("lbl_subject_name")
        self.form_layout_subject_details.setWidget(
            1, QtWidgets.QFormLayout.LabelRole, self.lbl_subject_name)
        self.line_edit_subject_name = QtWidgets.QLineEdit(self.layoutWidget)
        self.line_edit_subject_name.setObjectName("line_edit_subject_name")
        self.form_layout_subject_details.setWidget(
            1, QtWidgets.QFormLayout.FieldRole, self.line_edit_subject_name)
        self.vert_layout_new_subject.addLayout(
            self.form_layout_subject_details)
        self.hori_line_subject_details = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_subject_details.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_subject_details.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_subject_details.setObjectName(
            "hori_line_subject_details")
        self.vert_layout_new_subject.addWidget(self.hori_line_subject_details)
        self.lbl_instruction = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setItalic(True)
        self.lbl_instruction.setFont(font)
        self.lbl_instruction.setObjectName("lbl_instruction")
        self.vert_layout_new_subject.addWidget(self.lbl_instruction)
        self.button_box_new_subject = QtWidgets.QDialogButtonBox(
            self.layoutWidget)
        font = QtGui.QFont()
        font.setItalic(False)
        self.button_box_new_subject.setFont(font)
        self.button_box_new_subject.setOrientation(QtCore.Qt.Horizontal)
        self.button_box_new_subject.setStandardButtons(
            QtWidgets.QDialogButtonBox.Cancel | QtWidgets.QDialogButtonBox.Save)
        self.button_box_new_subject.setObjectName("button_box_new_subject")
        self.vert_layout_new_subject.addWidget(
            self.button_box_new_subject, 0, QtCore.Qt.AlignLeft)

        self.retranslateUi(dialog_new_subject)
        self.button_box_new_subject.accepted.connect(dialog_new_subject.accept)
```

```
        self.button_box_new_subject.rejected.connect(dialog_new_subject.reject)
        QtCore.QMetaObject.connectSlotsByName(dialog_new_subject)

    def retranslateUi(self, dialog_new_subject):
        _translate = QtCore.QCoreApplication.translate
        dialog_new_subject.setWindowTitle(
            _translate("dialog_new_subject", "Add Subject"))
        self.lbl_new_subject.setText(_translate(
            "dialog_new_subject", "Add a New Subject"))
        self.lbl_subject_name.setText(_translate(
            "dialog_new_subject", "Subject Name:"))
        self.lbl_instruction.setText(_translate(
            "dialog_new_subject", "Please enter your subject name (maximum 30 characters)."))

```

Agenda

agenda.py

```python
import json
import sys
from datetime import datetime

from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QDialog, QMainWindow

from add_task_setup import Ui_dialog_new_task
from agenda_setup import Ui_mwindow_agenda

# Hides completed tasks by default.
hidden_tasks = True

# Reads existing JSON files for lists of tasks.
with open('task_list.json', 'r') as outfile:
    try:
        task_list = json.load(outfile)
        json.dump(task_list, sys.stdout, ensure_ascii=False, indent=4)

        with open('task_list_hidden.json', 'r') as outfile:
            task_list_hidden = json.load(outfile)
    except ValueError:
        print("Empty JSON file.")
        task_list = []
        task_list_hidden = []

# Sets up the Add Task dialog.


class AddTaskDialog(QDialog, Ui_dialog_new_task):
    def __init__(self):
        super().__init__()
        self.setupUi(self)

# Sets up the Agenda main window.


class AgendaWindow(QMainWindow, Ui_mwindow_agenda):
```

```python
    def __init__(self):
        super().__init__()
        self.setupUi(self)

        # Sets the list of tasks to have a transparent background.
        self.setStyleSheet("""QTableWidget {background-color: transparent;}
            QHeaderView::section {background-color: transparent;}
            QHeaderView {background-color: transparent;}
            QTableCornerButton::section{background-color: transparent;}""")

        # Connects 'Add Task' button to the Add Task dialog.
        self.btn_add_task.clicked.connect(self.open_dialog_add_task)
        # Connects 'Mark as Complete/Incomplete' button to mark selected task.
        self.btn_complete_task.clicked.connect(self.mark_task_complete)
        # Connects 'Delete Task' button to delete selected task.
        self.btn_delete_task.clicked.connect(self.delete_task)
        # Connects 'Hide/Show Completed Tasks' button to hide/show tasks.
        self.btn_hide_completed.clicked.connect(self.hide_completed_tasks)

        # Sets text of task view button to 'Show Completed Tasks'.
        self.btn_hide_completed.setText("Show Completed Tasks")

        # Populates list on start-up.
        self.update_list()

    # Opens the dialog for the user to add a task.  u
    def open_dialog_add_task(self):
        self.Dialog = AddTaskDialog()

        # Connects 'Save' button to save the user's task to Agenda.
        self.Dialog.button_box_new_task.accepted.disconnect()
        self.Dialog.button_box_new_task.accepted.connect(self.save_task)

        # Populates the combo box with subject options.
        with open("subject_list.txt", "r") as data_file:
            subject_list = data_file.readlines()
            for line in subject_list:
                self.Dialog.comb_box_subject.addItem(line.strip("\n"))
```

```
        self.Dialog.open()

    # Populates the agenda with all tasks.
    def update_list(self):
        # Displays completed tasks if user selected the option.
        if hidden_tasks is True:
            current_list = task_list_hidden
        else:
            current_list = task_list

        # Sets number of rows, column size, and row size.
        self.table_widget_task_list.setRowCount(len(current_list))
        self.table_widget_task_list.horizontalHeader().setSectionResizeMode(
            QtWidgets.QHeaderView.ResizeToContents)
        self.table_widget_task_list.verticalHeader().setSectionResizeMode(
            QtWidgets.QHeaderView.Fixed)

        # Sorts the task list by due date.
        self.sort_task_list()

        # Adds tasks to the agenda.
        current_row = 0
        for task in current_list:
            current_column = 0

            # Adds task details to the agenda.
            for task_details in task.values():
                self.table_widget_task_list.setItem(
                    current_row, current_column,
                    QtWidgets.QTableWidgetItem(task_details))
                current_column += 1

            current_row += 1

    # Saves new tasks to the agenda and JSON file.
    def save_task(self):
        # Gets user inputs for subject and due date, and assigns task as incomplete.
        new_subject = self.Dialog.comb_box_subject.currentText()
        new_due_date = self.Dialog.calendar_due_date.selectedDate().toString("dd/MM/yyyy")
        new_completed = ("No")

        # Validates task title input and adds task if validation passed.
        new_task_title = self.Dialog.line_edit_task_title.text()
        if len(new_task_title) <= 30 and len((new_task_title).strip(" ")) > 0:
            # Appends new task to the task list as a dictionary.
            task = {
                "task_title": new_task_title,
                "subject": new_subject,
                "due_date": new_due_date,
                "completed": new_completed}
            task_list.append(dict(task))

            self.save_task_list()

            # Closes the dialog and updates the task list.
            self.Dialog.close()
            self.update_list()

        # Rejects input if no task title entered.
        elif len((new_task_title).strip(" ")) == 0:
            self.Dialog.lbl_instruction.setText(
                "You have not entered a task title. Please try again.")

        # Rejects input if task title exceeds 30 characters.
        else:
            self.Dialog.lbl_instruction.setText(
                "Your task title exceeds 30 characters. Please try again.")

    # Updates the JSON file with the current task list.
    def save_task_list(self):
        with open('task_list.json', 'w') as outfile:
            json.dump(task_list, outfile, ensure_ascii=False, indent=4)

        task_list_hidden[:] = [
            task for task in task_list if task["completed"] == ("No")]
```

```
        with open('task_list_hidden.json', 'w') as outfile:
            json.dump(task_list_hidden, outfile, ensure_ascii=False, indent=4)

    # Sorts the task list by due date.
    def sort_task_list(self):
        task_list.sort(key=lambda task: datetime.strptime(
            task["due_date"], '%d/%m/%Y'))
        self.save_task_list()

    # Hides/shows completed tasks.
    def hide_completed_tasks(self):
        # Calls for use of the global variable.
        global hidden_tasks

        # Changes variable value and sets text according to previous value.
        if hidden_tasks is False:
            hidden_tasks = True
            self.btn_hide_completed.setText("Show Completed Tasks")
        else:
            hidden_tasks = False
            self.btn_hide_completed.setText("Hide Completed Tasks")

        self.update_list()

    # Marks selected task as complete/incomplete.
    def mark_task_complete(self):
        selected_row = self.table_widget_task_list.currentRow()
        if self.table_widget_task_list.item(selected_row, 3).text() == "No":
            self.table_widget_task_list.item(selected_row, 3).setText("Yes")
        else:
            self.table_widget_task_list.item(selected_row, 3).setText("No")

        # Obtains the task title, subject, and due date of selected task.
        selected_task_title = self.table_widget_task_list.item(
            selected_row, 0).text()
        selected_subject = self.table_widget_task_list.item(
            selected_row, 1).text()
        selected_due_date = self.table_widget_task_list.item(
            selected_row, 2).text()

        # Iterates through task list for matching task and updates completion.
        for num, task in enumerate(task_list):
            if (task["task_title"] == selected_task_title
                    and task["subject"] == selected_subject
                    and task["due_date"] == selected_due_date):
                task_list[num]["completed"] = self.table_widget_task_list.item(
                    selected_row, 3).text()

        self.save_task_list()
        self.update_list()

    # Deletes selected task from agenda.
    def delete_task(self):
        selected_row = self.table_widget_task_list.currentRow()

        # Obtains the task title, subject, and due date of selected task.
        selected_task_title = self.table_widget_task_list.item(
            selected_row, 0).text()
        selected_subject = self.table_widget_task_list.item(
            selected_row, 1).text()
        selected_due_date = self.table_widget_task_list.item(
            selected_row, 2).text()

        # Iterates through task list and deletes matching task.
        for task in task_list:
            if (task["task_title"] == selected_task_title
                    and task["subject"] == selected_subject
                    and task["due_date"] == selected_due_date):
                task_list.remove(task)

        self.save_task_list()
        self.update_list()


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    mwindow_agenda = AgendaWindow()
    mwindow_agenda.show()
    sys.exit(app.exec_())
```

agenda_setup.py

```python
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'agenda.ui'
#
# Created by: PyQt5 UI code generator 5.12.1
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_mwindow_agenda(object):
    def setupUi(self, mwindow_agenda):
        mwindow_agenda.setObjectName("mwindow_agenda")
        mwindow_agenda.resize(960, 720)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        mwindow_agenda.setFont(font)
        self.central_widget = QtWidgets.QWidget(mwindow_agenda)
        self.central_widget.setObjectName("central_widget")
        self.gridLayout = QtWidgets.QGridLayout(self.central_widget)
        self.gridLayout.setObjectName("gridLayout")
        self.scrl_area_agenda_1 = QtWidgets.QScrollArea(self.central_widget)
        self.scrl_area_agenda_1.setFrameShape(QtWidgets.QFrame.NoFrame)
        self.scrl_area_agenda_1.setWidgetResizable(True)
        self.scrl_area_agenda_1.setObjectName("scrl_area_agenda_1")
        self.scrl_area_agenda_2 = QtWidgets.QWidget()
        self.scrl_area_agenda_2.setGeometry(QtCore.QRect(0, 0, 938, 648))
        self.scrl_area_agenda_2.setObjectName("scrl_area_agenda_2")
        self.layoutWidget = QtWidgets.QWidget(self.scrl_area_agenda_2)
        self.layoutWidget.setGeometry(QtCore.QRect(10, 16, 911, 611))
        self.layoutWidget.setObjectName("layoutWidget")
        self.vert_layout_agenda = QtWidgets.QVBoxLayout(self.layoutWidget)
        self.vert_layout_agenda.setContentsMargins(0, 0, 0, 0)
        self.vert_layout_agenda.setObjectName("vert_layout_agenda")
        self.lbl_agenda = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setPointSize(16)
        self.lbl_agenda.setFont(font)
        self.lbl_agenda.setObjectName("lbl_agenda")
        self.vert_layout_agenda.addWidget(self.lbl_agenda)
        self.hori_line_agenda = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_agenda.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_agenda.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_agenda.setObjectName("hori_line_agenda")
        self.vert_layout_agenda.addWidget(self.hori_line_agenda)
        self.hori_layout_buttons = QtWidgets.QHBoxLayout()
        self.hori_layout_buttons.setObjectName("hori_layout_buttons")
        self.btn_add_task = QtWidgets.QPushButton(self.layoutWidget)
        self.btn_add_task.setObjectName("btn_add_task")
        self.hori_layout_buttons.addWidget(
            self.btn_add_task, 0, QtCore.Qt.AlignLeft)
        self.btn_complete_task = QtWidgets.QPushButton(self.layoutWidget)
        self.btn_complete_task.setObjectName("btn_complete_task")
        self.hori_layout_buttons.addWidget(self.btn_complete_task)
        self.btn_delete_task = QtWidgets.QPushButton(self.layoutWidget)
        self.btn_delete_task.setObjectName("btn_delete_task")
        self.hori_layout_buttons.addWidget(self.btn_delete_task)
        spacerItem = QtWidgets.QSpacerItem(
            40, 20, QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Minimum)
        self.hori_layout_buttons.addItem(spacerItem)
        self.btn_hide_completed = QtWidgets.QPushButton(self.layoutWidget)
        self.btn_hide_completed.setObjectName("btn_hide_completed")
        self.hori_layout_buttons.addWidget(self.btn_hide_completed)
        self.vert_layout_agenda.addLayout(self.hori_layout_buttons)
        self.hori_line_add_task = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_add_task.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_add_task.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_add_task.setObjectName("hori_line_add_task")
        self.vert_layout_agenda.addWidget(self.hori_line_add_task)
        self.table_widget_task_list = QtWidgets.QTableWidget(self.layoutWidget)
        palette = QtGui.QPalette()
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Button, brush)
```

```
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.NoBrush)
        palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Base, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Window, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Button, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.NoBrush)
        palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Base, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Window, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Button, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.NoBrush)
        palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Base, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Window, brush)
        self.table_widget_task_list.setPalette(palette)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.table_widget_task_list.setFont(font)
        self.table_widget_task_list.setAutoFillBackground(False)
        self.table_widget_task_list.setStyleSheet(
            "background-color: transparent")
        self.table_widget_task_list.setFrameShape(QtWidgets.QFrame.NoFrame)
        self.table_widget_task_list.setLineWidth(0)
        self.table_widget_task_list.setSizeAdjustPolicy(
            QtWidgets.QAbstractScrollArea.AdjustToContents)
        self.table_widget_task_list.setEditTriggers(
            QtWidgets.QAbstractItemView.NoEditTriggers)
```

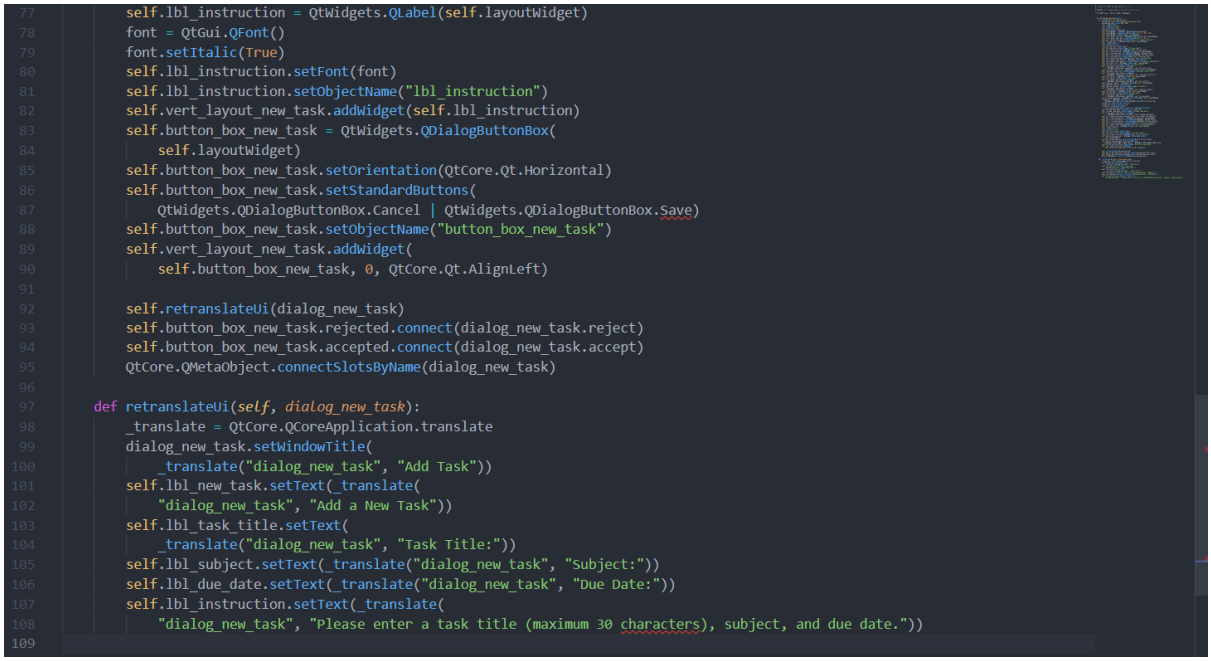
```
        self.table_widget_task_list.setTabKeyNavigation(False)
        self.table_widget_task_list.setProperty("showDropIndicator", True)
        self.table_widget_task_list.setSelectionMode(
            QtWidgets.QAbstractItemView.SingleSelection)
        self.table_widget_task_list.setSelectionBehavior(
            QtWidgets.QAbstractItemView.SelectRows)
        self.table_widget_task_list.setShowGrid(False)
        self.table_widget_task_list.setCornerButtonEnabled(False)
        self.table_widget_task_list.setObjectName("table_widget_task_list")
        self.table_widget_task_list.setColumnCount(4)
        self.table_widget_task_list.setRowCount(0)
        item = QtWidgets.QTableWidgetItem()
        item.setBackground(QtGui.QColor(0, 0, 0, 0))
        self.table_widget_task_list.setHorizontalHeaderItem(0, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_task_list.setHorizontalHeaderItem(1, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_task_list.setHorizontalHeaderItem(2, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_task_list.setHorizontalHeaderItem(3, item)
        self.table_widget_task_list.horizontalHeader().setVisible(True)
        self.table_widget_task_list.horizontalHeader().setHighlightSections(True)
        self.table_widget_task_list.horizontalHeader().setStretchLastSection(False)
        self.table_widget_task_list.verticalHeader().setVisible(True)
        self.vert_layout_agenda.addWidget(self.table_widget_task_list)
        self.scrl_area_agenda_1.setWidget(self.scrl_area_agenda_2)
        self.gridLayout.addWidget(self.scrl_area_agenda_1, 0, 0, 1, 1)
        mwindow_agenda.setCentralWidget(self.central_widget)
        self.menu_bar = QtWidgets.QMenuBar(mwindow_agenda)
        self.menu_bar.setGeometry(QtCore.QRect(0, 0, 960, 25))
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_bar.setFont(font)
        self.menu_bar.setObjectName("menu_bar")
        self.menu_agenda = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
```

```
        font.setPointSize(10)
        self.menu_agenda.setFont(font)
        self.menu_agenda.setObjectName("menu_agenda")
        self.menu_my_subjects = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_my_subjects.setFont(font)
        self.menu_my_subjects.setObjectName("menu_my_subjects")
        self.menu_timetable = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_timetable.setFont(font)
        self.menu_timetable.setObjectName("menu_timetable")
        mwindow_agenda.setMenuBar(self.menu_bar)
        self.status_bar = QtWidgets.QStatusBar(mwindow_agenda)
        self.status_bar.setObjectName("status_bar")
        mwindow_agenda.setStatusBar(self.status_bar)
        self.menu_bar.addAction(self.menu_my_subjects.menuAction())
        self.menu_bar.addAction(self.menu_agenda.menuAction())
        self.menu_bar.addAction(self.menu_timetable.menuAction())

        self.retranslateUi(mwindow_agenda)
        QtCore.QMetaObject.connectSlotsByName(mwindow_agenda)

    def retranslateUi(self, mwindow_agenda):
        _translate = QtCore.QCoreApplication.translate
        mwindow_agenda.setWindowTitle(_translate("mwindow_agenda", "Agenda"))
        self.lbl_agenda.setText(_translate("mwindow_agenda", "Agenda"))
        self.btn_add_task.setText(_translate("mwindow_agenda", "Add Task"))
        self.btn_complete_task.setText(_translate(
            "mwindow_agenda", "Mark as Complete/Incomplete"))
        self.btn_delete_task.setText(
            _translate("mwindow_agenda", "Delete Task"))
        self.btn_hide_completed.setText(_translate(
            "mwindow_agenda", "Hide/Show Completed Tasks"))
        self.table_widget_task_list.setSortingEnabled(False)
```

```
        item = self.table_widget_task_list.horizontalHeaderItem(0)
        item.setText(_translate("mwindow_agenda", "Task Title"))
        item = self.table_widget_task_list.horizontalHeaderItem(1)
        item.setText(_translate("mwindow_agenda", "Subject"))
        item = self.table_widget_task_list.horizontalHeaderItem(2)
        item.setText(_translate("mwindow_agenda", "Due Date"))
        item = self.table_widget_task_list.horizontalHeaderItem(3)
        item.setText(_translate("mwindow_agenda", "Completed?"))
        self.menu_agenda.setTitle(_translate("mwindow_agenda", "Agenda"))
        self.menu_my_subjects.setTitle(
            _translate("mwindow_agenda", "My Subjects"))
        self.menu_timetable.setTitle(_translate("mwindow_agenda", "Timetable"))

```

add_task_setup.py

```python
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'add_task.ui'
#
# Created by: PyQt5 UI code generator 5.12.1
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_dialog_new_task(object):
    def setupUi(self, dialog_new_task):
        dialog_new_task.setObjectName("dialog_new_task")
        dialog_new_task.resize(800, 600)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        dialog_new_task.setFont(font)
        self.layoutWidget = QtWidgets.QWidget(dialog_new_task)
        self.layoutWidget.setGeometry(QtCore.QRect(11, 11, 621, 433))
        self.layoutWidget.setObjectName("layoutWidget")
        self.vert_layout_new_task = QtWidgets.QVBoxLayout(self.layoutWidget)
        self.vert_layout_new_task.setContentsMargins(0, 0, 0, 0)
        self.vert_layout_new_task.setObjectName("vert_layout_new_task")
        self.lbl_new_task = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setPointSize(14)
        self.lbl_new_task.setFont(font)
        self.lbl_new_task.setObjectName("lbl_new_task")
        self.vert_layout_new_task.addWidget(self.lbl_new_task)
        self.hori_line_new_task = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_new_task.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_new_task.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_new_task.setObjectName("hori_line_new_task")
        self.vert_layout_new_task.addWidget(self.hori_line_new_task)
        self.form_layout_task_details = QtWidgets.QFormLayout()
        self.form_layout_task_details.setObjectName("form_layout_task_details")
        self.lbl_task_title = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_task_title.setObjectName("lbl_task_title")
        self.form_layout_task_details.setWidget(
            1, QtWidgets.QFormLayout.LabelRole, self.lbl_task_title)
        self.line_edit_task_title = QtWidgets.QLineEdit(self.layoutWidget)
        self.line_edit_task_title.setObjectName("line_edit_task_title")
        self.form_layout_task_details.setWidget(
            1, QtWidgets.QFormLayout.FieldRole, self.line_edit_task_title)
        self.lbl_subject = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_subject.setObjectName("lbl_subject")
        self.form_layout_task_details.setWidget(
            2, QtWidgets.QFormLayout.LabelRole, self.lbl_subject)
        self.comb_box_subject = QtWidgets.QComboBox(self.layoutWidget)
        self.comb_box_subject.setFrame(True)
        self.comb_box_subject.setObjectName("comb_box_subject")
        self.form_layout_task_details.setWidget(
            2, QtWidgets.QFormLayout.FieldRole, self.comb_box_subject)
        self.lbl_due_date = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_due_date.setObjectName("lbl_due_date")
        self.form_layout_task_details.setWidget(
            3, QtWidgets.QFormLayout.LabelRole, self.lbl_due_date)
        self.calendar_due_date = QtWidgets.QCalendarWidget(self.layoutWidget)
        sizePolicy = QtWidgets.QSizePolicy(
            QtWidgets.QSizePolicy.Fixed, QtWidgets.QSizePolicy.Preferred)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(
            self.calendar_due_date.sizePolicy().hasHeightForWidth())
        self.calendar_due_date.setSizePolicy(sizePolicy)
        self.calendar_due_date.setObjectName("calendar_due_date")
        self.form_layout_task_details.setWidget(
            3, QtWidgets.QFormLayout.FieldRole, self.calendar_due_date)
        self.vert_layout_new_task.addLayout(self.form_layout_task_details)
        self.hori_line_task_details = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_task_details.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_task_details.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_task_details.setObjectName("hori_line_task_details")
        self.vert_layout_new_task.addWidget(self.hori_line_task_details)
```



```
        self.lbl_instruction = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setItalic(True)
        self.lbl_instruction.setFont(font)
        self.lbl_instruction.setObjectName("lbl_instruction")
        self.vert_layout_new_task.addWidget(self.lbl_instruction)
        self.button_box_new_task = QtWidgets.QDialogButtonBox(
            self.layoutWidget)
        self.button_box_new_task.setOrientation(QtCore.Qt.Horizontal)
        self.button_box_new_task.setStandardButtons(
            QtWidgets.QDialogButtonBox.Cancel | QtWidgets.QDialogButtonBox.Save)
        self.button_box_new_task.setObjectName("button_box_new_task")
        self.vert_layout_new_task.addWidget(
            self.button_box_new_task, 0, QtCore.Qt.AlignLeft)

        self.retranslateUi(dialog_new_task)
        self.button_box_new_task.rejected.connect(dialog_new_task.reject)
        self.button_box_new_task.accepted.connect(dialog_new_task.accept)
        QtCore.QMetaObject.connectSlotsByName(dialog_new_task)

    def retranslateUi(self, dialog_new_task):
        _translate = QtCore.QCoreApplication.translate
        dialog_new_task.setWindowTitle(
            _translate("dialog_new_task", "Add Task"))
        self.lbl_new_task.setText(_translate(
            "dialog_new_task", "Add a New Task"))
        self.lbl_task_title.setText(
            _translate("dialog_new_task", "Task Title:"))
        self.lbl_subject.setText(_translate("dialog_new_task", "Subject:"))
        self.lbl_due_date.setText(_translate("dialog_new_task", "Due Date:"))
        self.lbl_instruction.setText(_translate(
            "dialog_new_task", "Please enter a task title (maximum 30 characters), subject, and due date."))

```

Timetable

timetable.py

```python
import json

from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QDialog, QMainWindow

from edit_timetable_setup import Ui_dialog_edit_timetable
from timetable_setup import Ui_mwindow_timetable

# Reads existing JSON files for list of lessons.
with open('timetable.json', 'r') as outfile:
    try:
        timetable = json.load(outfile)
    except ValueError:
        print("Empty JSON file.")
        timetable = []

    # Adds empty dictionary values to missing indexes in the list.
    missing_lessons = 25 - len(timetable)
    for index in range(missing_lessons):
        lesson = {"subject": " ",
                  "teacher": " ",
                  "room": " "}
        timetable.append(lesson)

# Sets up the Edit Timetable dialog.


class EditTimetableDialog(QDialog, Ui_dialog_edit_timetable):
    def __init__(self):
        super().__init__()
        self.setupUi(self)

# Sets up the Timetable main window.


class TimetableWindow(QMainWindow, Ui_mwindow_timetable):
    def __init__(self):
        super().__init__()
        self.setupUi(self)

        # Sets the timetable to have a transparent background.
        self.setStyleSheet("""QTableWidget {background-color: transparent;}
            QHeaderView::section {background-color: transparent;}
            QHeaderView {background-color: transparent;}
            QTableCornerButton::section{background-color: transparent;}""")

        # Connects 'Edit Timetable Slot' button to the Edit Timetable dialog.
        self.btn_edit_timetable.clicked.connect(
            self.open_dialog_edit_timetable)
        # Connects 'Clear Timetable Slot' button to clear the slot.
        self.btn_clear_timetable.clicked.connect(self.clear_timetable_slot)

        self.update_timetable()

    # Opens the dialog for Edit Timetable.
    def open_dialog_edit_timetable(self):
        self.Dialog = EditTimetableDialog()

        # Connects 'Save' button to save the lesson to the timetable slot.
        self.Dialog.button_box_edit_timetable.accepted.disconnect()
        self.Dialog.button_box_edit_timetable.accepted.connect(
            self.save_lesson)

        # Populates the combo box with subject options.
        with open("subject_list.txt", "r") as data_file:
            subject_list = data_file.readlines()
            for line in subject_list:
                self.Dialog.comb_box_subject.addItem(line.strip("\n"))

        self.Dialog.open()

    # Populates the timetable with all lessons.
    def update_timetable(self):
        # Resizes columns and rows to fit the contents.
        self.table_widget_timetable.horizontalHeader().setSectionResizeMode(
            QtWidgets.QHeaderView.ResizeToContents)
```

```
        self.table_widget_timetable.verticalHeader().setSectionResizeMode(
            QtWidgets.QHeaderView.ResizeToContents)

        # Adds lesson details to each cell in the timetable.
        for row_index, row in enumerate(timetable[::5]):
            for column_index, column in enumerate(range(5)):
                index = (row_index * 5) + column_index
                lesson_details = ((timetable[index]["subject"]) + "\n" +
                                  (timetable[index]["teacher"]) + "\n" +
                                  (timetable[index]["room"]))
                self.table_widget_timetable.setItem(
                    row_index, column_index,
                    QtWidgets.QTableWidgetItem(lesson_details))

    # Updates the JSON file with the current timetable list.
    def save_timetable_list(self):
        with open('timetable.json', 'w') as outfile:
            json.dump(timetable, outfile, ensure_ascii=False, indent=4)

        self.update_timetable()

    # Saves the lesson to the selected timetable slot.
    def save_lesson(self):
        # Gets the row and column of the selected timetable slot.
        selected_row = self.table_widget_timetable.currentRow()
        selected_column = self.table_widget_timetable.currentColumn()

        # Gets the user inputs for subject, teacher, and room.
        lesson_subject = self.Dialog.comb_box_subject.currentText()
        lesson_teacher = self.Dialog.line_edit_teacher.text()
        lesson_room = self.Dialog.line_edit_room.text()

        # Saves lesson to selected timetable slot if length validations passed.
        if len(lesson_teacher) > 30:
            self.Dialog.lbl_instruction.setText(
                "Your teacher input exceeds 30 characters. Please try again.")
        elif len(lesson_room) > 20:
            self.Dialog.lbl_instruction.setText(
                "Your room input exceeds 30 characters. Please try again.")
        else:
            lesson = {"subject": lesson_subject,
                      "teacher": lesson_teacher,
                      "room": lesson_room}
            index = (selected_row * 5) + selected_column
            timetable[index] = lesson

            self.save_timetable_list()
            self.Dialog.close()

    # Clears the lesson from the selected timetable slot.
    def clear_timetable_slot(self):
        # Gets the row and column of the selected timetable slot.
        selected_row = self.table_widget_timetable.currentRow()
        selected_column = self.table_widget_timetable.currentColumn()

        # Removes the lesson details from the timetable slot.
        lesson = {"subject": "",
                  "teacher": "",
                  "room": ""}
        index = (selected_row * 5) + selected_column
        timetable[index] = lesson

        self.save_timetable_list()


# Opens the main window when the program is executed.
if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    mwindow_timetable = TimetableWindow()
    mwindow_timetable.show()
    sys.exit(app.exec_())
```

timetable_setup.py

```python
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'timetable.ui'
#
# Created by: PyQt5 UI code generator 5.12.1
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_mwindow_timetable(object):
    def setupUi(self, mwindow_timetable):
        mwindow_timetable.setObjectName("mwindow_timetable")
        mwindow_timetable.resize(960, 720)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        font.setKerning(True)
        mwindow_timetable.setFont(font)
        self.central_widget = QtWidgets.QWidget(mwindow_timetable)
        self.central_widget.setObjectName("central_widget")
        self.gridLayout = QtWidgets.QGridLayout(self.central_widget)
        self.gridLayout.setObjectName("gridLayout")
        self.scrl_area_timetable_1 = QtWidgets.QScrollArea(self.central_widget)
        self.scrl_area_timetable_1.setFrameShape(QtWidgets.QFrame.NoFrame)
        self.scrl_area_timetable_1.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.scrl_area_timetable_1.setLineWidth(0)
        self.scrl_area_timetable_1.setWidgetResizable(True)
        self.scrl_area_timetable_1.setObjectName("scrl_area_timetable_1")
        self.scrl_area_timetable_2 = QtWidgets.QWidget()
        self.scrl_area_timetable_2.setGeometry(QtCore.QRect(0, 0, 938, 647))
        font = QtGui.QFont()
        font.setKerning(True)
        self.scrl_area_timetable_2.setFont(font)
        self.scrl_area_timetable_2.setAutoFillBackground(True)
        self.scrl_area_timetable_2.setObjectName("scrl_area_timetable_2")
        self.layoutWidget = QtWidgets.QWidget(self.scrl_area_timetable_2)
        self.layoutWidget.setGeometry(QtCore.QRect(11, 11, 911, 631))
        self.layoutWidget.setObjectName("layoutWidget")
        self.vert_layout_timetable = QtWidgets.QVBoxLayout(self.layoutWidget)
        self.vert_layout_timetable.setContentsMargins(0, 0, 0, 0)
        self.vert_layout_timetable.setObjectName("vert_layout_timetable")
        self.lbl_timetable = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setPointSize(16)
        self.lbl_timetable.setFont(font)
        self.lbl_timetable.setObjectName("lbl_timetable")
        self.vert_layout_timetable.addWidget(self.lbl_timetable)
        self.hori_line_timetable = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_timetable.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_timetable.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_timetable.setObjectName("hori_line_timetable")
        self.vert_layout_timetable.addWidget(self.hori_line_timetable)
        self.hori_layout_buttons = QtWidgets.QHBoxLayout()
        self.hori_layout_buttons.setObjectName("hori_layout_buttons")
        self.btn_edit_timetable = QtWidgets.QPushButton(self.layoutWidget)
        self.btn_edit_timetable.setObjectName("btn_edit_timetable")
        self.hori_layout_buttons.addWidget(
            self.btn_edit_timetable, 0, QtCore.Qt.AlignLeft)
        self.btn_clear_timetable = QtWidgets.QPushButton(self.layoutWidget)
        self.btn_clear_timetable.setObjectName("btn_clear_timetable")
        self.hori_layout_buttons.addWidget(
            self.btn_clear_timetable, 0, QtCore.Qt.AlignLeft)
        spacerItem = QtWidgets.QSpacerItem(
            40, 20, QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Minimum)
        self.hori_layout_buttons.addItem(spacerItem)
        self.vert_layout_timetable.addLayout(self.hori_layout_buttons)
        self.hori_line_edit_timetable = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_edit_timetable.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_edit_timetable.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_edit_timetable.setObjectName("hori_line_edit_timetable")
        self.vert_layout_timetable.addWidget(self.hori_line_edit_timetable)
        self.table_widget_timetable = QtWidgets.QTableWidget(self.layoutWidget)
        font = QtGui.QFont()
        font.setFamily("Arial")
```

```
        font.setPointSize(10)
        self.table_widget_timetable.setFont(font)
        self.table_widget_timetable.setStyleSheet(
            "background-color: transparent")
        self.table_widget_timetable.setFrameShape(QtWidgets.QFrame.NoFrame)
        self.table_widget_timetable.setEditTriggers(
            QtWidgets.QAbstractItemView.NoEditTriggers)
        self.table_widget_timetable.setTabKeyNavigation(False)
        self.table_widget_timetable.setProperty("showDropIndicator", True)
        self.table_widget_timetable.setSelectionMode(
            QtWidgets.QAbstractItemView.SingleSelection)
        self.table_widget_timetable.setShowGrid(True)
        self.table_widget_timetable.setGridStyle(QtCore.Qt.DotLine)
        self.table_widget_timetable.setCornerButtonEnabled(False)
        self.table_widget_timetable.setObjectName("table_widget_timetable")
        self.table_widget_timetable.setColumnCount(5)
        self.table_widget_timetable.setRowCount(5)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_timetable.setVerticalHeaderItem(0, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_timetable.setVerticalHeaderItem(1, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_timetable.setVerticalHeaderItem(2, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_timetable.setVerticalHeaderItem(3, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_timetable.setVerticalHeaderItem(4, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_timetable.setHorizontalHeaderItem(0, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_timetable.setHorizontalHeaderItem(1, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_timetable.setHorizontalHeaderItem(2, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_timetable.setHorizontalHeaderItem(3, item)
        item = QtWidgets.QTableWidgetItem()
        self.table_widget_timetable.setHorizontalHeaderItem(4, item)
        item = QtWidgets.QTableWidgetItem()
```

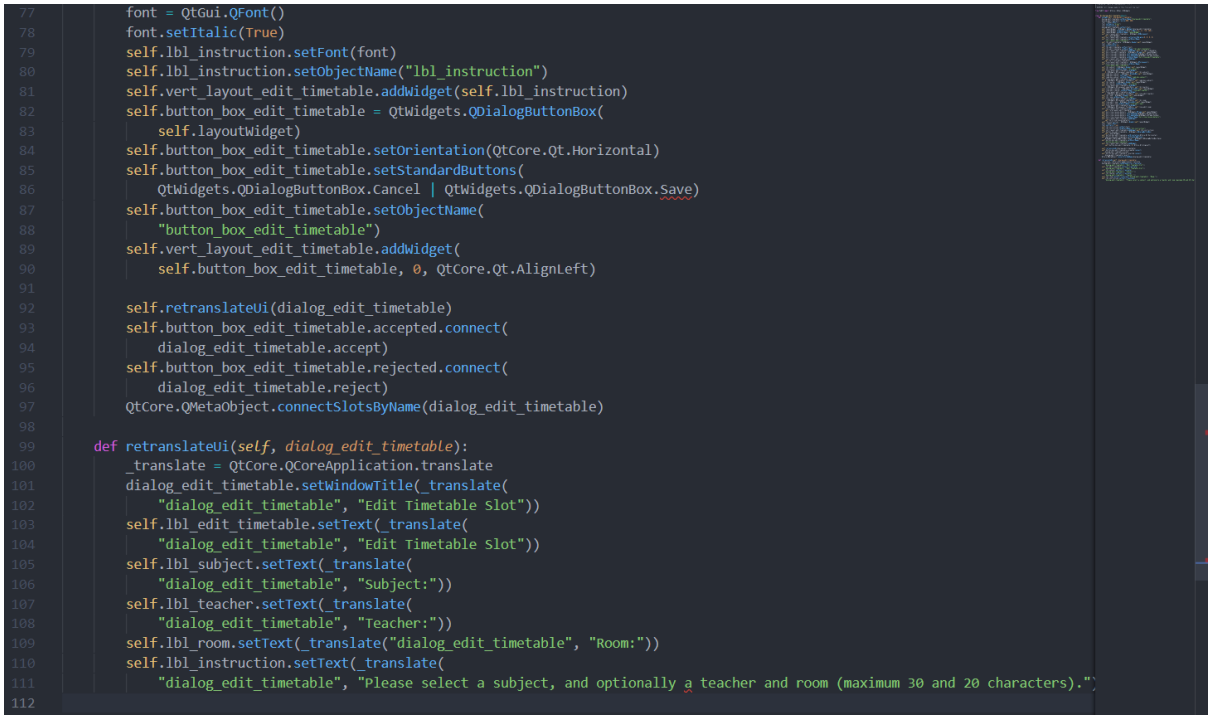
```
        self.table_widget_timetable.setItem(0, 0, item)
        self.table_widget_timetable.horizontalHeader().setHighlightSections(True)
        self.table_widget_timetable.horizontalHeader().setStretchLastSection(False)
        self.vert_layout_timetable.addWidget(self.table_widget_timetable)
        self.scrl_area_timetable_1.setWidget(self.scrl_area_timetable_2)
        self.gridLayout.addWidget(self.scrl_area_timetable_1, 0, 0, 1, 1)
        mwindow_timetable.setCentralWidget(self.central_widget)
        self.menu_bar = QtWidgets.QMenuBar(mwindow_timetable)
        self.menu_bar.setGeometry(QtCore.QRect(0, 0, 960, 26))
        self.menu_bar.setObjectName("menu_bar")
        self.menu_my_subjects = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_my_subjects.setFont(font)
        self.menu_my_subjects.setObjectName("menu_my_subjects")
        self.menu_agenda = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_agenda.setFont(font)
        self.menu_agenda.setObjectName("menu_agenda")
        self.menu_timetable = QtWidgets.QMenu(self.menu_bar)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        self.menu_timetable.setFont(font)
        self.menu_timetable.setObjectName("menu_timetable")
        mwindow_timetable.setMenuBar(self.menu_bar)
        self.status_bar = QtWidgets.QStatusBar(mwindow_timetable)
        self.status_bar.setObjectName("status_bar")
        mwindow_timetable.setStatusBar(self.status_bar)
        self.actionAgenda = QtWidgets.QAction(mwindow_timetable)
        self.actionAgenda.setObjectName("actionAgenda")
        self.menu_bar.addAction(self.menu_my_subjects.menuAction())
        self.menu_bar.addAction(self.menu_agenda.menuAction())
        self.menu_bar.addAction(self.menu_timetable.menuAction())

```

```
        self.retranslateUi(mwindow_timetable)
        QtCore.QMetaObject.connectSlotsByName(mwindow_timetable)

    def retranslateUi(self, mwindow_timetable):
        _translate = QtCore.QCoreApplication.translate
        mwindow_timetable.setWindowTitle(
            _translate("mwindow_timetable", "Timetable"))
        self.lbl_timetable.setText(_translate(
            "mwindow_timetable", "Timetable"))
        self.btn_edit_timetable.setText(_translate(
            "mwindow_timetable", "Edit Timetable Slot"))
        self.btn_clear_timetable.setText(_translate(
            "mwindow_timetable", "Clear Timetable Slot"))
        item = self.table_widget_timetable.verticalHeaderItem(0)
        item.setText(_translate("mwindow_timetable", "Period 1"))
        item = self.table_widget_timetable.verticalHeaderItem(1)
        item.setText(_translate("mwindow_timetable", "Period 2"))
        item = self.table_widget_timetable.verticalHeaderItem(2)
        item.setText(_translate("mwindow_timetable", "Period 3"))
        item = self.table_widget_timetable.verticalHeaderItem(3)
        item.setText(_translate("mwindow_timetable", "Period 4"))
        item = self.table_widget_timetable.verticalHeaderItem(4)
        item.setText(_translate("mwindow_timetable", "Period 5"))
        item = self.table_widget_timetable.horizontalHeaderItem(0)
        item.setText(_translate("mwindow_timetable", "Monday"))
        item = self.table_widget_timetable.horizontalHeaderItem(1)
        item.setText(_translate("mwindow_timetable", "Tuesday"))
        item = self.table_widget_timetable.horizontalHeaderItem(2)
        item.setText(_translate("mwindow_timetable", "Wednesday"))
        item = self.table_widget_timetable.horizontalHeaderItem(3)
        item.setText(_translate("mwindow_timetable", "Thursday"))
        item = self.table_widget_timetable.horizontalHeaderItem(4)
        item.setText(_translate("mwindow_timetable", "Friday"))
        __sortingEnabled = self.table_widget_timetable.isSortingEnabled()
        self.table_widget_timetable.setSortingEnabled(False)
        self.table_widget_timetable.setSortingEnabled(__sortingEnabled)
        self.menu_my_subjects.setTitle(
            _translate("mwindow_timetable", "My Subjects"))
```

```
        self.menu_agenda.setTitle(_translate("mwindow_timetable", "Agenda"))
        self.menu_timetable.setTitle(
            _translate("mwindow_timetable", "Timetable"))
        self.actionAgenda.setText(_translate("mwindow_timetable", "Agenda"))

```

edit_timetable_setup.py

```python
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'edit_timetable.ui'
#
# Created by: PyQt5 UI code generator 5.12.1
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_dialog_edit_timetable(object):
    def setupUi(self, dialog_edit_timetable):
        dialog_edit_timetable.setObjectName("dialog_edit_timetable")
        dialog_edit_timetable.resize(800, 600)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(10)
        dialog_edit_timetable.setFont(font)
        self.layoutWidget = QtWidgets.QWidget(dialog_edit_timetable)
        self.layoutWidget.setGeometry(QtCore.QRect(11, 11, 716, 208))
        self.layoutWidget.setObjectName("layoutWidget")
        self.vert_layout_edit_timetable = QtWidgets.QVBoxLayout(
            self.layoutWidget)
        self.vert_layout_edit_timetable.setContentsMargins(0, 0, 0, 0)
        self.vert_layout_edit_timetable.setObjectName(
            "vert_layout_edit_timetable")
        self.lbl_edit_timetable = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setPointSize(14)
        self.lbl_edit_timetable.setFont(font)
        self.lbl_edit_timetable.setObjectName("lbl_edit_timetable")
        self.vert_layout_edit_timetable.addWidget(self.lbl_edit_timetable)
        self.hori_line_edit_timetable = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_edit_timetable.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_edit_timetable.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_edit_timetable.setObjectName("hori_line_edit_timetable")
        self.vert_layout_edit_timetable.addWidget(
            self.hori_line_edit_timetable)
        self.form_layout_edit_timetable = QtWidgets.QFormLayout()
        self.form_layout_edit_timetable.setObjectName(
            "form_layout_edit_timetable")
        self.lbl_subject = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_subject.setObjectName("lbl_subject")
        self.form_layout_edit_timetable.setWidget(
            0, QtWidgets.QFormLayout.LabelRole, self.lbl_subject)
        self.comb_box_subject = QtWidgets.QComboBox(self.layoutWidget)
        self.comb_box_subject.setFrame(True)
        self.comb_box_subject.setObjectName("comb_box_subject")
        self.form_layout_edit_timetable.setWidget(
            0, QtWidgets.QFormLayout.FieldRole, self.comb_box_subject)
        self.lbl_teacher = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_teacher.setObjectName("lbl_teacher")
        self.form_layout_edit_timetable.setWidget(
            1, QtWidgets.QFormLayout.LabelRole, self.lbl_teacher)
        self.line_edit_teacher = QtWidgets.QLineEdit(self.layoutWidget)
        self.line_edit_teacher.setObjectName("line_edit_teacher")
        self.form_layout_edit_timetable.setWidget(
            1, QtWidgets.QFormLayout.FieldRole, self.line_edit_teacher)
        self.lbl_room = QtWidgets.QLabel(self.layoutWidget)
        self.lbl_room.setObjectName("lbl_room")
        self.form_layout_edit_timetable.setWidget(
            2, QtWidgets.QFormLayout.LabelRole, self.lbl_room)
        self.line_edit_room = QtWidgets.QLineEdit(self.layoutWidget)
        self.line_edit_room.setObjectName("line_edit_room")
        self.form_layout_edit_timetable.setWidget(
            2, QtWidgets.QFormLayout.FieldRole, self.line_edit_room)
        self.vert_layout_edit_timetable.addLayout(
            self.form_layout_edit_timetable)
        self.hori_line_lesson_details = QtWidgets.QFrame(self.layoutWidget)
        self.hori_line_lesson_details.setFrameShape(QtWidgets.QFrame.HLine)
        self.hori_line_lesson_details.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.hori_line_lesson_details.setObjectName("hori_line_lesson_details")
        self.vert_layout_edit_timetable.addWidget(
            self.hori_line_lesson_details)
        self.lbl_instruction = QtWidgets.QLabel(self.layoutWidget)
```



```
        font = QtGui.QFont()
        font.setItalic(True)
        self.lbl_instruction.setFont(font)
        self.lbl_instruction.setObjectName("lbl_instruction")
        self.vert_layout_edit_timetable.addWidget(self.lbl_instruction)
        self.button_box_edit_timetable = QtWidgets.QDialogButtonBox(
            self.layoutWidget)
        self.button_box_edit_timetable.setOrientation(QtCore.Qt.Horizontal)
        self.button_box_edit_timetable.setStandardButtons(
            QtWidgets.QDialogButtonBox.Cancel | QtWidgets.QDialogButtonBox.Save)
        self.button_box_edit_timetable.setObjectName(
            "button_box_edit_timetable")
        self.vert_layout_edit_timetable.addWidget(
            self.button_box_edit_timetable, 0, QtCore.Qt.AlignLeft)

        self.retranslateUi(dialog_edit_timetable)
        self.button_box_edit_timetable.accepted.connect(
            dialog_edit_timetable.accept)
        self.button_box_edit_timetable.rejected.connect(
            dialog_edit_timetable.reject)
        QtCore.QMetaObject.connectSlotsByName(dialog_edit_timetable)

    def retranslateUi(self, dialog_edit_timetable):
        _translate = QtCore.QCoreApplication.translate
        dialog_edit_timetable.setWindowTitle(_translate(
            "dialog_edit_timetable", "Edit Timetable Slot"))
        self.lbl_edit_timetable.setText(_translate(
            "dialog_edit_timetable", "Edit Timetable Slot"))
        self.lbl_subject.setText(_translate(
            "dialog_edit_timetable", "Subject:"))
        self.lbl_teacher.setText(_translate(
            "dialog_edit_timetable", "Teacher:"))
        self.lbl_room.setText(_translate("dialog_edit_timetable", "Room:"))
        self.lbl_instruction.setText(_translate(
            "dialog_edit_timetable", "Please select a subject, and optionally a teacher and room (maximum 30 and 20 characters)."))

```

Bibliography

Article Resources

- BBC: Tablet computers in '70% of schools'
  - https://www.bbc.co.uk/news/education-30216408
- Google: Design – Material Design
  - https://material.io/design/
- Python: PEP 8 - Style Guide for Python Code
  - https://www.python.org/dev/peps/pep-0008/
- Scribe Consulting: Serif and Sans-Serif Fonts
  - http://www.scribe.com.au/tip-w017.html
- UX Planet: Infinite Paging vs. Pagination
  - https://uxplanet.org/ux-infinite-scrolling-vs-pagination-1030d29376f1
- SkyLogic Projects: How to Install and Build Your First GUI in Python 3.4
  - http://projects.skylogic.ca/blog/how-to-install-pyqt5-and-build-your-first-gui-in-python-3-4/
- W3Schools: Python JSON
  - https://www.w3schools.com/python/python_json.asp
- Stack Abuse: Reading and Writing JSON to a File in Python
  - https://stackabuse.com/reading-and-writing-json-to-a-file-in-python/

Community Forum Resources

- Stack Overflow: When should I be using classes in Python?
  - https://stackoverflow.com/questions/33072570/when-should-i-be-using-classes-in-python
- Stack Overflow: Classes vs. Functions
  - https://stackoverflow.com/questions/18202818/classes-vs-functions
- Stack Overflow: How to Open Windows in PyQt
  - https://stackoverflow.com/questions/27567208/how-do-i-open-sub-window-after-i-click-on-button-on-main-screen-in-pyqt4
- Stack Overflow: How to Open New Window in PyQt with a Button
  - https://stackoverflow.com/questions/36768033/pyqt-how-to-open-new-window
- Stack Overflow: Sorting Lists by Date
  - https://stackoverflow.com/questions/23158766/sort-a-python-date-string-list

Video Resources

- YouTube: Let's Learn Python #24 – UI with Python, PyQt & Qt Designer
  - https://www.youtube.com/watch?v=GLqrzLIIW2E