**Car rental problem – coursework**

**Contents**

Analysis

Problem identification 4

Why is it suited to a computational solution? 4

Stakeholders 4

Computational methods 5

Problem recognition 5

Problem decomposition 5

Divide and conquer 5

Abstraction 5

Interview 6

Interview questions 6

Interview 7

Research 11

Existing similar solution 11

Features of proposed solution 14

Further meeting with stakeholders 14

Requirements 16

Software and hardware requirements 16

Stakeholder requirements 16

Success criteria 18

# Design

User interface design 19

Loading screen 22

Usability features 22

Stakeholder input 23

Algorithms 24

Diagram showing how subroutines link 27

Subroutines 28

Inputs and Outputs 33

Key variables 34

Validation 35

Testing method 36

# Development and testing

Stage 1 38

Stage 1 Review 41

Stage 2 38

Stage 2 Review 41

Stage 3 45

Stage 3 Review 48

Stage 4 49

Stage 4 Review 53

Stage 5 54

Stage 5 Review 58

Stage 6 60

Stage 6 Review 65

Stage 7 66

Stage 7 Review 75

Stage 8 78

# Evaluation

Criteria 80

Success criteria 80

Evidence 81

Limitations 85

# Final code

Car+Rental+Company.py 86

**Analysis**

**Problem identification**

Cars have become a necessity in everyday life to the point where they have become the most popular mode of transport worldwide. Whether needed for business or personal reasons they are currently very essential. Even though this is true a large percentage of the population (around 23.9% in England for example) do not own cars due to financial or ethical reasons etc, which creates problems for these people as we live in a time where owning a car is common and relied upon. So, creating a way for people to temporarily own a car for their personal or commercial reasons would solve this problem as these people in need of an automobile would have a cheap option available to them.

Renting a car isn’t the most popular option as it isn’t properly presented to these people who have issues with buying a car outright this is where the problem mainly is. There aren’t many reliable and cheap cars on the market to rent for these people. In addition, the procedure to rent a car for someone who is inexperienced would prove to be confusing and time consuming. So, solving that problem and having such options would open opportunities for the lower and working class as it would help with commuting and overall financial stability.

Why is it suited to a computational solution?

This problem is suited to being solved through computational methods because there are many factors involved in renting a car. The main problem is the vast number of models and makes of cars and how different details are involved (e.g., engine size, year the car was released etc). So, as there are many different types of cars the customer’s personal details can be used as a sort of filter, so when they are given, in response, the cars they can rent would be provided. This is where the problem is suitable for being solved through computational methods, as, if an algorithm was to be coded where it asks for the customers details and then outputs cars that are available and suitable for those personal details then the whole system would be a lot more efficient. The more efficient and simpler the better for the user as they do not have to be experienced in the automobile market and would just have to follow the steps given on screen.

Stakeholders

The demographic for this software ranges from people who use cars regularly for commuting or people who use cars for road trips etc. There are many rental companies out there so the stakeholders will be a just a small sample of people who use these companies. This sample will range from people who are inexperienced with the use of computers and the process of renting a car (most likely older people) to people who are more experienced and are looking for a user-friendly application they can rely on and come back to constantly. **Computational methods:**

Problem recognition

The overall problem is finding a way to output a set of cars that the user can choose to rent based off their own personal details. However, the underlaying problem is comparing these details to the database of cars and outputting the correct ones (that aren’t already being rented) so there is no confusion with the user. For example, if the user is only 18 and has 2 years of experience, only 1.2L – 1.6L cars like fiats 500s or Renault Clios should be shown not 4L Jeeps. Once this matching of data is solved the rest of the system is simply just updating and changing 2 separate databases constantly depending on information inputted.

Problem decomposition:

This problem can be decomposed into smaller steps:

* Asking the user for their personal details
* Saving the details to a database named “personal details”
* Comparing the details to another database already created “Cars”
* Outputting available cars depending on the details
* Updating the database “Cars” depending on whether the user chose to rent

This process would be completed in seconds and would seem effortless to the user each time a new or returning user starts the program.

Divide and conquer

The algorithm would be challenging to complete through a head on approach, so splitting up the problem into these smaller steps, solving them then combining each module into one algorithm (a modular program) would be a perfect example of the divide and conquer method of problem solving.

Abstraction

In this solution abstraction is essential as a lot of unnecessary data is being handled, the goal is to be simplistic so said data will be removed and only the useful data will be kept and recalled when necessary.

**Interview**

I asked four different stakeholders about their experiences and opinions on rental cars and their use. The first two stakeholders use cars for road trips and holidays whereas the second two use their cars for commuting. In this section I will show the questions I asked, why I asked those questions, and the answer to the questions

Holidays and road trips

These are the questions I asked Edward and Leo:

1. Have you ever rented a car under your own name?
2. If yes, what was your reasoning behind renting?
3. What was your experience like going through the process and return the car?
4. What are your thoughts on renting over buying?
5. What do you think is the best devices to go through the process on (for example a mobile phone or a laptop etc)?
6. Do you have anything you would like to add?

In this interview, I asked a range of questions based on different aspects on the problem. For questions one and two I focused on the reasoning behind why some people may use automobiles for holidays etc. Identifying customer reasoning is essential for solving this problem as I can use that information to make the algorithm appeal to those people.

With questions three, four and five I wanted to know what his opinion and experience was on the systems he has used before when renting and what he would like to see in a new one. This is also crucial information as I can now make the program fit his needs and I can avoid any problems he faced before.

The last question was asked in the case that I missed any important detail.

Commuting

Here are the questions I asked Joaquin and John:

1. Do you own a car, or do you rent one for work?
2. (Depending on their answer) why did you choose that over the other option?
3. Have you ever rented before if not would you ever want to?
4. Do you believe everyone should own cars or is renting the better option for the future?
5. Have you saved money on traveling to work?
6. Do you have anything you would like to add?

For this interview I wanted to know to opinions and views on a person who uses a car for commuting and find out what the disadvantages are. For questions one, two and three I wanted to know his overall history with cars and what was stopping them from renting over buying if that’s what they had chosen.

For question four finding out what their personal view on the problem was important so I knew what kind of views people who drive to work have on cars and their effect on the environment.

And finally for questions five and six I wanted to know what his financial situation was like and if he wanted to add any more details

**Interview**

Holidays and road trips

Leo

1. **Have you ever rented a car under your own name?**

“Yes, all the time actually. Whenever I go on road trips, I rent so I can have the right car for what I need”

1. **If yes, what was your reasoning behind renting, if not why?**

“It was mainly so that I have the right car for the job. I wouldn’t have to buy and sell constantly just because one specific trip was different.”

1. **What was your experience like going through the process to rent and return the car?**

“Every service I use is different and hard to get my head around at first. There are so many steps involved and if you entered the wrong detail once you would have to start again. The main reason why I have used so many is because I’m trying to find one that I can just always come back to and is simple to use with no errors. There have been many times before where my details have been mixed up with someone else’s in the system, which is the worst thing.”

1. **What are your thoughts on renting over buying?**

“I thinking renting is such a great thing for people like me who don’t have enough money or a good income to pay of a suitable car for road trips. The reason why I can go on these trips is because I can rent otherwise, I wouldn’t be able to go anywhere. Financially they are the best option for the lower class”

1. **What do you think is the best devices to go through the process on (for example a mobile phone or a laptop etc)?**

“I think the easiest to use is probably a laptop or a PC, but I would like it on my mobile phone because I’m always traveling it would be easier if I ever have a problem on the road”

1. **Do you have anything else you would like to add?**

“No, I think I have said all I want to say”

Edward

1. **Have you ever rented a car under your own name?**

“Yes, only once, as I usually cycle everywhere”

1. **If yes, what was your reasoning behind renting, if not why?**

“It was the cheaper option at the time as I was just going away for the week, and I don’t really need a permanent car”

1. **What was your experience like going through the process to rent and return the car?**

“It was very confusing and difficult. Took me a couple days but I got it in the end. If I ever rent again, it won’t be with that same service it was too time consuming. I would like it to be easier and quicker to use.”

1. **What are your thoughts on renting over buying?**

“I haven’t bought a car before so I can’t really say much but I can say that the prices were a lot better and choosing how long I can use it for is a lot better than buying and worrying about selling it after”

1. **What do you think is the best devices to go through the process on (for example a mobile phone or a laptop etc)?**

“Personally, I think the best is a computer as it’s just easier, there’s a large screen, physical keyboard and mouse etc. I do see a mobile phone being useful however as it is more portable.”

1. **Do you have anything else you would like to add?**

“Not really just that I would like to rent again in the future if I can find a system that’s a lot quicker and easier to use”

Analysis

From their answers the main thing that stands out is that renting is very helpful for their use and that making the program is simple and easy to use as I can is very important for users. Leo has used many different services purely because he is trying to find one that he can keep using and is efficient. Edward however has only needed to use it once, likes the idea of renting but also needs a good service that uses a user-friendly program. Also, both stated that being able to use this across multiple devices would prove to be useful especially for travellers.

Commuting

Joaquin

1. **Do you own a car, or do you rent one for work?**

“I own a car for work, but I am still paying off a loan”

1. **Why did you choose to buy a car over renting?**

“I needed a car for work, so I took a loan out for it. It was what all my friends where doing so I thought It was a smart idea too and I heard renting was unreliable”

1. **Have you ever rented before if not would you ever want to?**

“No, I haven’t before this is my first car, but I would really like to know after experiencing what it’s like to own a car. It seems so much easier to rent as it is cheaper etc. I had tried before but just didn’t understand the system I had to go through it was too confusing, so I gave up and just bought a car instead”

1. **Do you believe everyone should own cars or is renting the better option for the future?**

“I believe renting is the way to go for the future. A lot of people own and use cars for no reason, and it harms the environment so I think we should all have to rent them and use them for important reasons like work only. So, to achieve this we need a reliable simple system to rent rather than the ones out there that are confusing for people who are inexperienced like me”

1. **Have you saved money on travelling to work?**

“Overall, yes I have as I work in the centre where bus and train fairs are at their highest. But there is still the chance to save more which is very important for me as my job isn’t a high paying one at all”

1. **Do you have anything else you would like to add?**

“Only that I would really like to have a simple easy way to rent cars because I would like to start doing so in the future”

John

1. **Do you own a car, or do you rent one for work?**

“I own a car”

1. **Why did you choose to buy a car over renting?**

“Because I work 6 days a week every week with barely any holiday time, so I need a permanent car”

1. **Have you ever rented before if not would you ever want to?**

“No, I have not but I can see the benefits of it just isn’t a suitable option for me as I am constantly driving everywhere. However, I would like to in the future as long as it is a quick and simple process”

1. **Do you believe everyone should own cars or is renting the better option for the future?**

“I do believe that renting may be the better option due to the environment as people would only rent and use them for specific reasons, but they have become too much of a necessity now especially for me”

1. **Have you saved money on travelling to work?**

“No, I have not as fuel prices are very expensive especially for my car as it is more of a premium car compared to others, which is why I am considering renting as I would save a lot”

1. **Do you have anything else you would like to add?**

“No, I have said everything I would like to say”

Analysis

For commuting there are two slightly different sides. On one side you have Joaquin who has tried to rent before but has been unable to because the system they used was to time consuming and confusing and would like to have a service that is a lot easier. Joaquin also believes that renting is a much better option for the future and wants to save money by going for that option. Whereas on the other hand John has never rented a car to use for work as he needs a permanent mode of transport. However, he does see the financial benefits and would like to rent in the future too if the process is easier and quick.

Overview

Overall, from the four interviews I can conclude that this is a suitable problem where being solved computationally is the best option. The main goal is to create a user-friendly, reliable piece of software that anyone (experienced or not) can use effortlessly.

**Research**

**Existing similar solution 1: “Rental cars.com”**

(Even though these examples are websites they can be useful to be for comparison)

A screenshot of a computer

Description automatically generated with medium confidenceOverview

This website uses multiple computational features like a built-in calendar to select dates for returning the car or check boxes etc. It also advertises its own offers and features that are shown to the user straight away. Techniques like this make the whole system more user-friendly. The text, the boxes, the background all follow a consistent colour scheme makes the website better to look at.

Graphical user interface, application

Description automatically generated

When details are entered the program starts a search. Usually this would just be a blank screen as the program searches the database but this system outputs this animation to cover for that blank screen and keep the user from thinking something went wrong or they must refresh etc.

Graphical user interface, text, application, email

Description automatically generated

Once the search is complete the available cars are outputted alongside the information the user inputted first so that if any mistakes were made, they can check again and edit if they need to. In this case there were no

cars available so instead of an error message showing a different message telling the user there are no cars available and what to do next. This is an example of validation.

Features I can apply to my solution

My solution will not be written in html and CSS however I can still use some parts. For example, text boxes, buttons and check boxes are all common and simple features. But also keeping that consistent colour scheme and having some sort of validation is very important in keeping the program user-friendly. Validation can be achieved by checking user details entered in each box and creating an output depending on if the information entered is valid. In addition, a implementing a loading screen would be a good idea as this program will be used on a wide range of devices that vary in performance and may take long on the searching part of the algorithm.

**Existing similar solution 2: “enterprise.co.uk”**

Overview

Graphical user interface, text, application

Description automatically generated

Enterprise uses a similar setup where the main area of interaction is in the middle of the page and is surrounded by extra information that could be useful for first time users. They have labeled each box 1 and 2 so that the user first enters their location. This is done to filter out any dates specific locations can’t do so that the user doesn’t have to waste time inputting dates that arent available.

Table

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Additional Features like drop down menus are used here to make the system more interactive with the user, instead of them entering lines of data instead they can click on certain days on the calender. This reduces the chance of error and makes the timings easier to visualise. In addition to drop down menus this site also includes an option to search at the users location. This feature has many benefits such as time saving as the user does not need to type out their whole address line by line. However as information about precise location can be considered sensitive data some users may not agree with the fact that this site uses their location. Also next some subheadings there are information buttons that when pressed take the user to a page that explains that part of the process, this removes any confusion or questions about the procedure.

Features I can apply to my solution

From this site it is clear that the features mentioned beforehand make the system look more professional and effortless. These attributes, such as; drop down menus, use of precise location and the extra information buttons, are crucial in creating my solution as they all help making the software more user-friendly, interactive and simple for any user experienced or not. The characteristics mentioned solve the problems that the interviewees mentioned and will make the for the best version of this solution.

**Features of the purposed solution**

Initial concept of my solution and approach based on research

The suitable approach to this problem, based on research of similar problems and solutions, would be to create an algorithm that asks the user to input their details into fields of which are pre-conditioned to make sure the user inputs correct details (for example if they were to enter an invalid email in the email field then a message would pop up instructing them to enter an email that has a correct format). This would be done through a GUI that consists of multiple text boxes to input into and buttons that the user will press to continue to the next step. This is to make the program more user-friendly to attract customers (as said before). In addition to this the program saves to an external database where it can be called back and read from. Depending on this information a new window will show up listing all the cars that can be rented by the user. The “personal details” database and the “available cars” database will be created as separate csv files and data will be called and compared. Additionally extra features like drop down menus and built-in calendars will be implemented where necessary.

Limitations of my solution

The main limitation of my system will most likely be that it will not be linked to external rental companies. Therefore I would have to create my own database of cars which is not difficult but will be a significantly smaller databse compared to the examples shown above during the research section. Also users will most likely have to download this application which may be inconvenient for devices with smaller storage space etc.

**Further meetings with stakeholders**

An email I sent to my stakeholders:

*“Hi,*

*I have come up with the idea of creating an application that opens a new window when ran and asks users to enter their data into the boxes show. This data will be saved in its own file named, “Personal details.csv” and will be kept at maximum security. Depending on these details certain cars from an existing database will be outputted in a new window for the user and they will choose which car and how long they want to rent for. The goal is to make the software as simple and easy to use as possible so that anyone could use it. Are there any changes or anything new you would like to add to this initial plan?*

*Thanks”*

These are the responses that I received:

**Leo:**

*“Overall it sounds like a great plan! However I would suggest adding some sort of login system where returning customers (like myself) are able to just enter a username and password and their details are already saved in the system. This would prevent the user having to constantly enter the same details”*

**Edward:**

*“I think this proposal is perfect, I think this is the best solution and you should go forward with it. Nothing to add, just that you should make the interface as simple as possible”*

**Joaquin:**

*“As long as the application is carried out step by step I’ll be fine with it. I would like on-screen instructions on what is happening when I press buttons or interact at all but other than that I think this is a good proposal.”*

**John:**

*“The main problem I have with this is the security of the “personal details.csv” file. There must not be any data leaks as sensitive information is being used. I’d suggest a form of encryption that’s the best option for this program. Also do not overuse features like drop down menus etc make sure to keep it simple that is the main goal. If these targets are hit I will be perfectly fine with this application”***Requirements**

**Sofware and Hardware requirements:**

Hardware:

* A computer, mobile phone or laptop – Any device with average performance benchamarks. Nothing powerful is needed to run this application just a standard device. They must not have any limitations (for example company or school computers) so that certain software can be ran.

Software:

* Windows, Mac Operating Sytem, Android, Iphone Operating system or Linux – These are common operating systems that are used on most latops, mobile phones and PCs.
* Python iInterpreter – A python interpreter is needed as the program will be written in this language
* “Pandas” – this is a databse modual for python which needs to be installed

**Stakeholder Requirements**

Design

|  |  |
| --- | --- |
| Requirement | Explanantion |
| Simple startup window, with minimal information | This allows for new users who have never rented before to not be bombared with information on things they do not have experience with |
| Consistent design | Following the same colour scheme and dimentions for text boxes, buttons etc. This is to make the program better to look at so the user does not click off. |
| Step-by-step setup | For new inexperienced users there must be a step by step procedure that tells them what is occuring at each stage. |
| Multiple screens | Each stage could close the old windows and open a new one or clear the original and replace with new content for the next stage. This is to keep the lightweight simplistic design so users arent scrolling though all the information at once |

Fucntionality

|  |  |
| --- | --- |
| **Requirement** | **Explanation** |
| “Continue” and “Back” buttons | This is too allow the user to continue when ready and also go back when they think they inputted the wrong details |
| Built-in calender | This would make inputting dates quicker and easier to visualise as it is far better to look at a calender than a set of numbers |
| Loading screens | Instead of just searching through the databse whilst on the same screen, a new screen can show up letting the user know the program is searching not frozen |
| Validation | Checking the data that is entered into the initial fields is the correct data type. For example if a string was enetered into the “Years of experience” text box then a message will be outputted rather than saving that invalid data. |

**Success Criteria**

|  |  |
| --- | --- |
| **Criteria** | **How to evidence** |
| Main window that pops up when ran | Screen shot the first window that shows up when ran |
| Buttons and text boxes that have funtions | Screen shot where the buttons are and what happens when pressed |
| Lightweight consistent design on each window | Screen shot each screen and compare against to see if they are consistent |
| Drop down menus | Screen shot before and after arrow is clicked to show size and colour of the drop down etc |
| Built-in calender | Screen show the whole callender before and after dates are chosen |
| A databse for personal details that can be read from and updated | Databse can be viewed in excel so data can be screen shotted before and after added or changed |
| Instructions at each stage | Screen shot all instructions |
| Output relevent cars after checking against data | Screenshot the cars outputted and all the cars in the database to show they are the correct ones |
| Validation | Screenshot all validation at each stage for any inputs. |

**Design**

Graphical user interface

Description automatically generatedUser interface design

This is the main design of the window that will open when the program is ran. This design provides a simple and modern look to the program making it better to look at for the user. A constant colour scheme alongside a consistent text font will be followed throughout the program and its multiple screens/pages.

The design I created links to the success criteria in multiple ways. For example:

* Main window that pops up when ran
* Buttons and text boxes that have functions
* Lightweight consistent design on each window

Page information

As there will be multiple pages each with different buttons and text boxes, it is important that the user is informed on what is happening on that screen and what they are required to do. In the “page information” box instructions and extra details about the process at that point will be shown to the user to clear any confusion for first time users.

Links to success criteria:

* Instructions at each stage

Labels

The sections labelled “Label” will have text like “Name:”, “Email:”, “Postcode”, “Date of birth”, “Years of experience” shown there as markers for the corresponding box next to them. They will just act as guidelines for where and what data the user must enter and will have no border or background colour.

Entry boxes

Each label has an entry box placed in the next column for the user to enter their details. In the case of incorrect format being entered by the user, for example, if the user enters their experience (which will be set to an integer) into the name box (which will be set to a string data type), the boxes’ outline will turn red, implying that the entry was not accepted. This is a form of validation.

Error messages

The sections labelled “error messages” are also a form of validation as the ones in the next column alongside entry boxes will appear when the user enters the incorrect data. These will be red messages that will inform the user that the data is invalid. The error message below the entry boxes alongside the button will be an extra message also in red text informing the user to go back and check the data entered in case there is any confusion on why the program is not continuing (this message will only appear if the user made a mistake in the first place).

Links to success criteria:

* Validation

Button

The button shown in the bottom right is a way for the user to save and continue when ready. This button will have multiple uses. First if any data is incorrectly inputted one of the functions of the button will be to output the error message to the user and change the border colour of the entry box letting the user know why they can’t continue. Next the correct data that is entered will be stored to an external database once the button is clicked. Also, when saved to the database the data that was correct will be deleted from the box to show the user that it was correctly saved. Finally, a new screen will pop up above the original moving the process to the next step.

Links to success criteria:

* Buttons and text boxes that have functions

Graphical user interface, treemap chart

Description automatically generatedLoading screen

This is the design of the second window (a “Loading screen”)

After the data inputted is saved to the database and everything is correct the button will open a second window whilst a search is running in the background of all the cars available. This is a method of informing the user that the program is working and is not in a frozen condition.

The label in the centre in this case will have a text along the lines of “searching please wait” alongside an animation of the common three dots. This is the main point if the screen, to inform the user that the program is running perfectly fine.

The image boxes surrounding the label will most likely be random images every time of cars that are available or current offers the company is running.

Usability Features

Graphical user interface

Description automatically generatedIn my design I have considered the use of multiple usability features. These features are there to make sure any type of user can use this software with ease. An example of usability features included is the easy to ready text font and colours. All text will follow colour schemes and sizes relevant to their importance, so information is still available to users with visual difficulties. In addition, this is overall a large design so all buttons, text boxes and other interactive features will be large. This is to make sure nothing is missed even if the user has any difficulties.

Although this is not the final design with the exact colours and sizes, it is an accurate presentation of what the program will roughly look like

The colours of the text and borders will most likely not be the same, but it will follow the same simple consistent design shown above.

In addition, the sizes of the buttons and the entry boxes will follow all have the same dimensions and will not be random. This is to make the program look cleaner to the user making it easier to use rather than having multiple shapes of random sizes on the screen.

This alongside the loading screen will increase communication with the user as through these designs the user is informed of what is happening and what to do at that moment.

Links to success criteria:

* Lightweight consistent design on each window

Stakeholder input

This is the stage where I have a good idea of what the program will look like and how it will work. I also had another meeting with the stakeholders to confirm my designs and if it follows what they required. I also asked them if they would like to add anything extra.

This is the email I sent to them:

*“Hi,*

*Below you can find diagrams of what the software will look like. The usability features I have included are large clear text, consistent colour schemes, large buttons, large entry boxes. I am really set on this design as I am sure it hits all the requirements however if there were anything you would like to change or add please would you let me know. Also, I would like to know your specific opinion on the types of images used in the loading screen?*

*Thanks,”*

Here are the replies I received:

**Leo:**

*“I think this design and the way it functions is perfect. It seems like the perfect approach to this, and it seems like you have really thought it through. As someone who does have visual difficulties, I can say the large sizes and the plan you have for the colours are both good ideas. However, for the loading screen it would be good to add images of the latest cars available and promote them as much as possible”*

**Edward:**

*“These diagrams attached are great. You have made the gui as simple as possible which is exactly what I wanted. The loading screen looks great as well including images of the latest offers would be a great idea”*

**Joaquin:**

*“The on-screen instructions really do make this design a great idea its what’s crucial as this is a confusing process sometimes and I’m glad you included it. I would say however, that the colours do look a bit bland and more popping ones should be included especially on the loading screen to make it look more interesting.”*

**John:**

*“This design is a good start but lacks detail. It is a lot of white space and even though the plan is to keep it simple I would suggest including a bit more on each screen to make it less bare. In addition, one thing to note is that you should make the the error messages aren’t the same and just repeats. You need to output messages that inform the user of what they are getting incorrect and how to fix it. For the loading screen images of cars with no background would look the best (avoid 4 square images all the same size).”*

Algorithms

The main challenge of this solution is the saving and updating of the external database. The main objective for this is:

Take the user data that have been inputted and save it into a table

Use that data to search and output cars that are available depending on the users’ personal details

Chart, box and whisker chart

Description automatically generatedThis can be broken down into smaller problems shown in the diagram below:

The process overall is mainly a few steps of saving to database, retrieving from the database, comparing, and retrieving from different database and finally updating second database depending on what the user chose to do.

Each step in more detail:

**Saving**

Table

Description automatically generatedThe first step is to save data entered by the user through the entry boxes as soon as the user clicks the button to do so. The best way sorts the details in the csv file is to order it in a table which looks like this:

**Retrieval:**

The second step will be retrieval of the data. For instance, data like name and years of experience will be retrieved so that the user can be referred to (for example “Hi ‘insert name’”). Therefore, the table above is crucial. If the data was not stored in an orderly fashion, it would be stored as one long string which would be a lot harder to retrieve and sperate.

**Search:**

The searching part is the third step of this process. This stage is where the data retrieved (for example years of experience) will be used to find an available car for the user to be able to rent. The program would search through the pre-existing cars database and find cars that the user would be allowed to rent. This depends on whether the cars status is available and matches the experience level of the user.

**New screen:**

When the search is finished the next part is for a new window to open automatically for the results to be outputted. The reason why a new window opens rather than just outputting the data on the same original window is so that the user is “altered” in a way. They would see a new window and realise that the next step is about to occur

**Output:**

The fifth stage is when all the cars available from the search are outputted on the new window. They are then followed by an entry box where the user must enter how long they would like to rent the car they chose.

**Update:**

The sixth stage is simply updating the cars database with the new information like which car has been chosen and for how long etc. This is so that when the program is used next the new user can’t choose to rent a car already out on rent.

**Final screen:**

This final step is there only to thank and inform the user the process was successful. Also, this will act as a confirmation screen where all they choices the user made are outputted so the user can confirm that is what they want. Once checked the user will be presented with a button that takes them back to the start or they can just close the program.

Diagram showing how the subroutines link

Chart, diagram

Description automatically generatedShown below in the diagram are the multiple subroutines after the program starts and the main screen opens. Each separate routine is colour coded.

Subroutines

I can now plan the subroutines I can use in this program. The program will be written as one python file and when ran will automatically create a new csv file for the personal details. In the same folder, there will be a pre-existing csv file which contains a list of all the cars the company have. Having a folder designated to just these three files will keep the project organised and will reduce chance of error in the program as, when ran, all the external database files will be located easily. This program will use tkinter (a built in GUI toolkit).

**Confirm button**

On the start screen there will be a confirm button for the user to click when they have finished the first step and would like to continue. This button will have multiple uses all written as separate functions and will be called upon using the “command” function when creating the button.

Pseudocode/Example algorithm (Written in IDLE as colour coding makes the code easier to visualise)

**A picture containing diagram

Description automatically generatedValidation function**

First before any data is saved it needs to be checked. The data entered by the user needs to be in the correct format and datatype otherwise there will be errors later in the process. The different types of data entered are name, postcode, email, DOB, and experience.

Text

Description automatically generatedPseudocode/Example algorithm

**Saving to database**

Next all correct data needs to be saved to the database but only if the data is correct which is why the” save\_to\_databse” function is only called if validation is correct. This functions task is simply to save to the external csv file.

Pseudocode/Example algorithm

Text, letter

Description automatically generated

**Clearing text boxes when saved**

The user needs a form of confirmation that the data has been saved so clearing the boxes is a good method to do so.

Pseudocode/Example algorithm

Text

Description automatically generated

**Opening a new window**

After the button is pressed a new window will open for the next step of the procedure. This is the window that will output all the available cars and allow the user to choose which one they would like.

Pseudocode/Example algorithm

Text

Description automatically generated

**Creating a drop-down menu for the available cars**

On this new window there will be a label followed by a drop-down menu instead of an entry box this time where all the cars will be listed, and the user would have to pick one. Below would be another entry box where the user can enter how many weeks, they want to rent that car for. Below in the example algorithm where “Renault Clio” is there will be multiple cars called from the database depending on the experience entered

Pseudocode/Example algorithm

Graphical user interface, text

Description automatically generated

**Main Loop**

The main loop is the first window that will open and will be running in the back the whole time. It is the “master” window.

Graphical user interface, text, application

Description automatically generatedPseudocode/Example algorithm

Testing these algorithms and trace tables

These functions are written using tkinter, which is a built in GUI tool within python). As there are many different datatypes involved and multiple databases the testing will occur alongside development. This is more time consuming and has a higher workload however, this is necessary as each step and subroutine relies heavily on the last so each step must be tested thoroughly before moving onwards.

Inputs and outputs

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Details entered by user | Will be saved to database and used to compare | Retrieved and outputted when necessary for a more user-friendly system (for example constantly referring to the user through their name) |
| Submit button | Will save details and will check the data as a form of validation | Output messages depending on whether data was entered in the correct format. It will also output a new window if everything is correct. |
| Drop down menu | Will contain all available cars for the user to choose | User will click on a car and that car will be outputted next to the box as confirmation for the user |
| Return to start button | When clicked the program restarts | Main start screen outputted, and program is started from beginning |

Key variables

These are the main variables that will be constantly used throughout the program

|  |  |  |
| --- | --- | --- |
| **Name** | **Datatype** | **How it is used** |
| name | String | User’s name will be assigned to this variable when retrieved so the user can be referred to throughout the program |
| Postcode | String | User’s postcode will be assigned to this variable |
| Email | String | User’s email will be assigned to this variable |
| DOB | String (as brackets will be included in the format DD/MM/YYY) | Date of birth. User’s DOB will be assigned to this variable |
| Experience | Integer (as whole numbers will only be relevant when representing number of years of experience) | User’s experience will be assigned to this value and called upon to compare and find relevant cars |
| Renting\_time | Integer (number of weeks) | Number of weeks the user wants to rent the car will be assigned to this value |

Validation

One of the main qualities needed with this program is robustness. This will be accomplished through validation. Validating the data entered by the user and making sure it is in the correct format and data type. These are the multiple ways the user can interact with the program and how they will be validated:

Buttons

As I will be using tkinter the buttons are already built well and robust. When created the programmer can choose to add a function to the button but if no function is added the button will do nothing. This is useful as I (the programmer) can choose what each button does exactly and when the user is able to press it (as there is an option to make it unclickable).

Entry boxes

The method to validate these boxes would be to check what data is entered and output a message informing the user whether the data is entered correctly. If the data is entered incorrectly the outline could change to a red colour as tkinter allows you to change features of the box after it is created. And if the data is entered correctly the box could either stay the same or the outline can change to green.

Drop-down menus

With drop down menus the user is very limited on what they can do so there is almost no chance of error from the user’s side. However, adding a label that outputs when the user selects an option just to make sure they chose the correct one will reduce chance of human error to almost 0.

Testing method

As mentioned before testing will occur during each stage of development and documented. I will test each stage by entering different types of data and inputs and seeing what occurs. This will help grasp the process from a user perspective and see if any improvements can be made.

After development is complete a final test will be carried out by me and multiple others to test if someone who does not know the code understands what is occurring.

I also will be carrying out destructive testing where I enter incorrect data on purpose to see how the program will handle it ensuring no error messages pop up.

All data entered and all results will be recorded and shown to the stakeholders to see if requirements are met.

Iterative development

I will be using iterative development for this program. This is where the program is broken down into smaller stages and subroutines and solved using an iterative process. Each stage will be tested and improved constantly until the whole program is completed.

Testing checklist

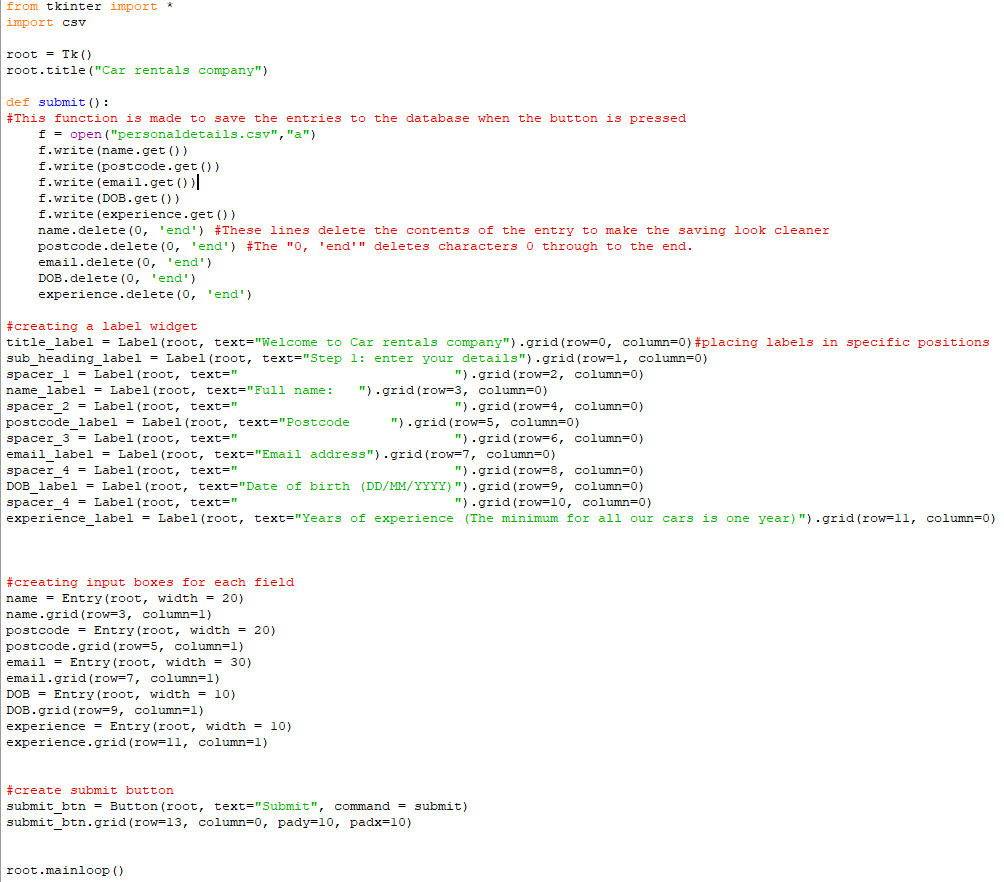
This is the checklist I will follow when testing to make sure that I have tested all aspects

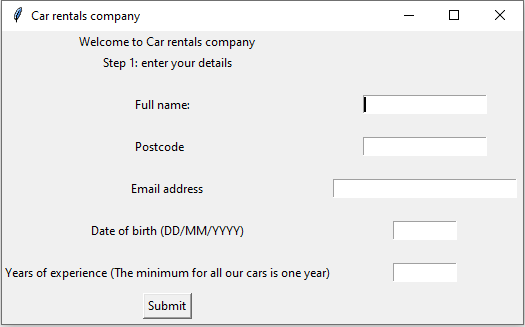
|  |  |
| --- | --- |
| **Test** | **Functional?** |
| Text entered in entry boxes |  |
| If submit button checks the data entered |  |
| If submit button saves to database |  |
| If submit button clears boxes |  |
| If submit button opens new window |  |
| If window outputs correct cars (test with multiple different years of experience) |  |
| If car database is updated on stock availability |  |
| If car database is updated with how long car is out to rent |  |
| If “back to start” button restarts program |  |

When this list is complete all the functionality of the program should be met with no errors. This will be passed to the stakeholders so that I can check if requirements are met too.

**Iterative development**

**Stage 1 – GUI**

*prototype:*



When this code is executed, the GUI below opens, this includes the text boxes, buttons, and text next to the boxes. Also included in this screenshot is the “submit” function which was made for when the button is pressed. This function saves to an external database and clears the boxes to make the whole GUI look cleaner.

*Validation:*

Text

Description automatically generated

Graphical user interface, application

Description automatically generatedThis function validates whether or not numbers and strings are entered into the entry boxes. If for example, a letter was entered into the date of birth box the program would not save it to the database and would ask the user to only enter numbers. At first I noticed that any correct data was saved even if the other boxes where filled incorrectly by the user so they wouldn’t know what was right and wrong. Therefore, I added error messages that displayed in red and a variable (save) that would constantly change depending on each if statements check and if the variable has a value of anything greater than 0 then the error messages shownup telling the user to fix any mistakes

*Failed tests and errors:*

A picture containing text

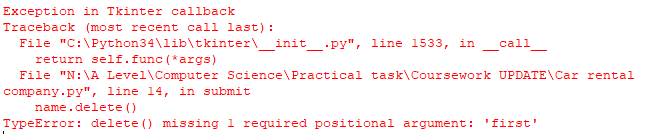
Description automatically generated

Graphical user interface, application

Description automatically generatedThe problem with this version of the function was that the save variable that I used to check whther the inoput was correct or not only depended on the last boxes input as it was replaced with 1 or 0 in the last statement which meant that anything before that didn’t matter.

So as shown above no matter what was entered in each box as long as the last one was entered correctly then the boxes would clear and save to the database. This was changed and instead of replacing what the variable “save” was each if statement either added or didn’t change it and the final statement just depended on whether the variable was greater than 0.

In addition to the validation error, another eror with clearing the boxes was made:



This was fixed simply by seperating the boxes position and its creation lines





*Review:*

In this stage a GUI which contained buttons and boxes that allowed the user to interact with and enter their data so it can be saved in a database and checked against was created. Alongside the GUI a validation process was completed which ensured that any data entered by the user was correct and followed what was asked for. This was trialed and tested with multiple different names and dates of birth etc.

**Stage 2 – Creating a database**

*Prototype:*

Text

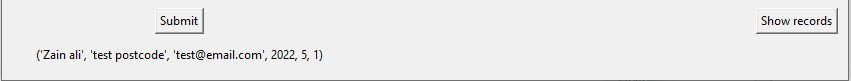
Description automatically generated

Here I have used sqlite3 to create the database “customer details” however the file created is just a databse file so it is unable to be opened by any program and viewed. Data can be saved, edited and fetched from this file using SQL.

In addition I created an extra button called show records next to the submit button purely to output the data saved back to the user. This was created for myself as a way to test that all the data was saved seperatly and not in one long string. I called this function “query”.

Graphical user interface, text

Description automatically generated

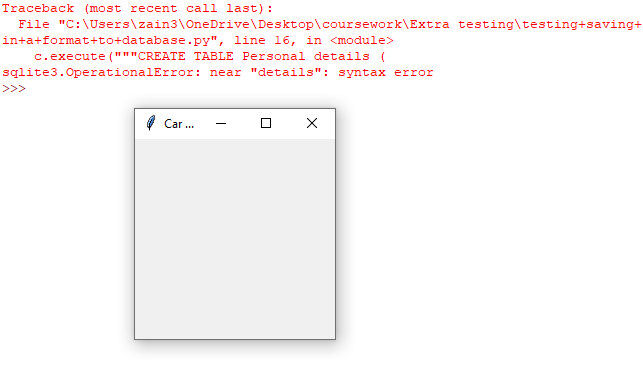


*Validation:*

The same validation is used here as the values entered by the user (that are checked before submitting in the GUI stage) are placed into each corresponding collumn which also have data types assigned to them witing the SQL code.



*Failed tests and errors:*



Here I encounted a syntax error here as the table name cant have any spaces so I added an underscore.

Text

Description automatically generated

A picture containing text

Description automatically generatedIn addition every time the code was ran for testing this error would show up as the code was trying to create a new table every time instead of using the existing one. I fixed this by changing “CREATE TABLE” too “CREATE TABLE IF NOT EXISTS” as shown below

Saving to database/review

To save the entry boxes to the databse I have just created I used this function which asigns the different data types to their respective columns:

Graphical user interface, text, application, email

Description automatically generated

This saved the details in the background and cleared the entry boxes.

**Stage 3 – Opening a new window**

*Prototype:*

In this stage I am aiming to write a function that opens a new window after the data is saved by the user. The code below should open a new window when the submit button is clicked but only after the validation funtion is ran through first.

Company name

Description automatically generated with medium confidence

Here is the funtion in its base state. All it will do is open a black window separate to the oringinal but also titled car rentals company. The funtion “open” is called within the validation funtion which is called when the “submit” button is clicked. This is to ensure the contents of the entry boxes are checked and saved to database before continuing on with the process.

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generatedWhen ran the new window opens up with no error. However it has no contents so the next step is to add the next step and allow for user input. I will use the labels and buttons as I had for the first window alongside drop down menus which will be created in the next stage.

*Validation:*

Text

Description automatically generatedHere I used the same validation function to ensure that the window is only opened for the next step when the details are entered correctly. There isnt any specific validation needed just making sure this function is only ran after as show below.

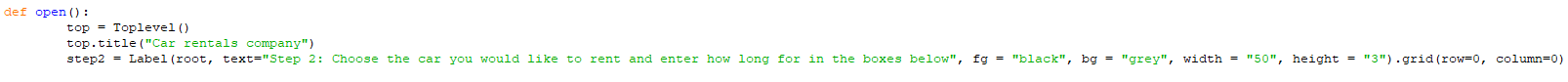
*Failed tests and errors:*

Editing the second window

To edit what is outputted on the second window I must add all the labels, button, menus etc under the line “top = Toplevel()” similar to the first window where everything was written under the “root = Tk()”. So below is a screenshot of all the code that changes the look of the second window.

Graphical user interface, application, Word

Description automatically generated

However as shown in the image above, the label I wanted to add saying “Step 2: Choose the car you would like to rent and enter how long for in the boxes below" did not output even though I wrote the line of code within the funtion.

This is because the label is being called to the “root” window which is the original window the opens up when the prgram starts. To solve this problem I simply have to change “root” to “top” as that is the name of the second window.

Graphical user interface, text, application

Description automatically generatedHowever after doing so and the label outputted I realise that the text was too long so it was cut off.

So to solve this I simply changed the width value within the label properties from 50 to 75 as shown below.



Graphical user interface, text, application

Description automatically generated

*Review:*

Graphical user interface, application

Description automatically generatedIn this stage I was able to open a second window for the second step and add details to it like information on the next step and what to add. Also it was crucial that this next window opens and the next step starts only after data has been saved and validated which is what I tested and made sure of by entering incorrect data and seeing if the second window opened regardless, which it didn’t as shown below:

So far I have fufilled mutiple requirements. I have created a window that shows up when the program starts, included a form of validation, included intructions, button, entry boxes etc. So from now the next step is to add more to the second window and building it up with a consistent, lightweight design.

Links to success criteria:

* Main window that pops up when ran
* Buttons and text boxes that have functions
* Lightweight consistent design on each window
* Validation

The program when ran so far:

Graphical user interface, application

Description automatically generated

**Stage 4 – Editing the new window and adding drop down boxes**

*Prototype:*

The intial objective this in stage is to add more to the second window and creat drop down menus. The user’s experience will be retrived from the database and be used in an if statement to output the model of car they are able to rent. Then the user will be able to enter how long (in weeks) they want to rent for.

Firstly I added all the labels needed before the entry boxes buttons and drop down menus so I can visualise where everything will be placed.

A picture containing text

Description automatically generated

When ran there was a syntax error as I had forgot to add a number after row on the label for spacer\_6



After fixing this error I also realised that the values for the other two labels were incorrect so I fixed them too.

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generatedWhen ran this was the look of the second window

Graphical user interface, text, application, email

Description automatically generatedI relaised here the text was too close together which wouldn’t be the best to look at for the user so I decided to add spacers again between the two rows. Next to the text “here are the available cars for your level of experience” will be a drop down menu for the user to select a car.

Text

Description automatically generated with medium confidenceNext I wrote the code to fetch the value the user entered for their experience. Here it was important to fetch the latest value not the first. Below is what I had coded so far.

The code above should order the table by descending date so that the latest record is first. Then using the “fetchone” command I am able to fetch the first record which is now the latest.

However when ran nothing happened, no error was displayed nor was the latest experience outputted.

After running the program again and replacing “index” with “rowid” I came across this error which was preventing me from outputting the experience to check if the retrival worked

Graphical user interface, text, application, email

Description automatically generated

A picture containing text

Description automatically generatedTo solve this instead of using two separate execute lines, I used one line and used the “rowid” funtion to flip the table and the “fetchone” funtion to fetch the first line.

So the latest value entered in the experience box was returned and set equal to the “records” variable so it can be compared later on for the specific cars.

Text

Description automatically generated

The code under “records = c.fetchone()” was copied from the “query” function to so that I can ensure that the correct data was fetched by having it outputted to me. This will be removed in the final code.

Output on second screen. Same as input on first (text to ensure correct data was returne)

Finally for this stage I needed to add an entry box for the user to add how long they want to rent the car for. In addition I added a new collumn into the personal details table so that it can be saved to when the user clickes continue.



A picture containing text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generatedHere was the output:

I decided I needed another grey spacer for the second column to make the program look cleaner so I added that in using a label.

*Review:*

Graphical user interface, text, application

Description automatically generatedAt this stage I have worked more on the look of the second window and followed the consistent design mentioned in the success criteria. I also have added a way to return the latest entry for the experience column of the table so that the user that is currently using the program has their relevent data at hand. This was all done under the “open” funtion. Also the section where the experience was printed was removed as its only purpose was to output the data so I can confirm it was the correct integers being used.

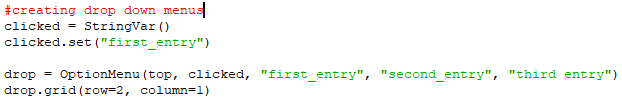
Graphical user interface, text, application, email

Description automatically generated

The next step is to add a drop down menu above the entry boxes which will contain available cars depending on the users experience. This will be done through a series of if statements. In addtion I must add a new column in the table for rented cars where the car the user picked is recorded in that column. This is so that staff can check what cars users are renting later on.

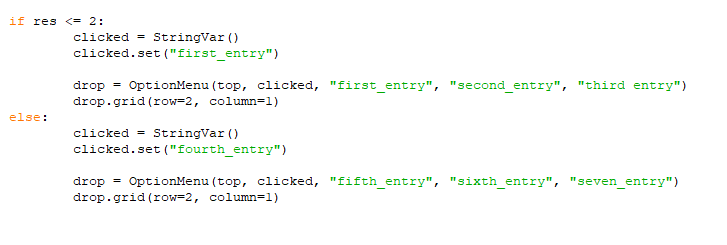
Stage 5 – Creating drop down menus

In this satge the objective is to create a drop down menu that has a set of values (I this case list of cars depending on what the user’s years of experience is. This relies heavily on the stage 4 so it would help to perform further testing on the last stage. The experience of the user will be returned and used in an if statement to be comapred to so that the correct versions of the drop down menu can be outputted.

Here is the set of code to create simple text boxes with no entries (the shell).

In these lines I use “first\_entry”, “second\_entry” as place holders to test if the drop down works.

After confirming the box apears in the correct place I added an IF statement where if the experience is less than or equal to 2 the first 4 terms output otherwise if the experience is anything else the second sate of data outputs in the menu.

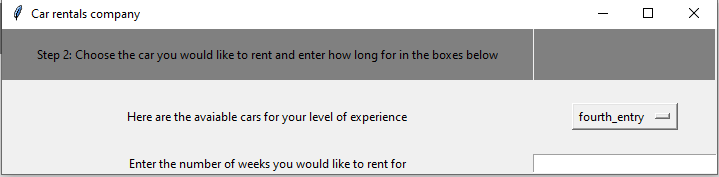


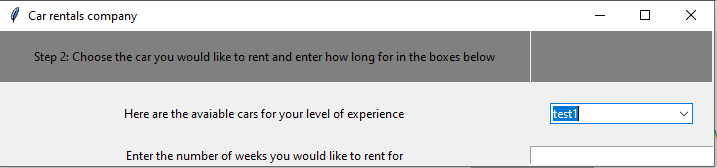
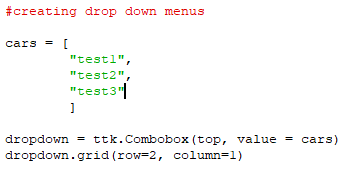
However I ran into this error where the variable “records” retrieved the experience value as an integer. So I replaced “records” with res to convert it to an integer.



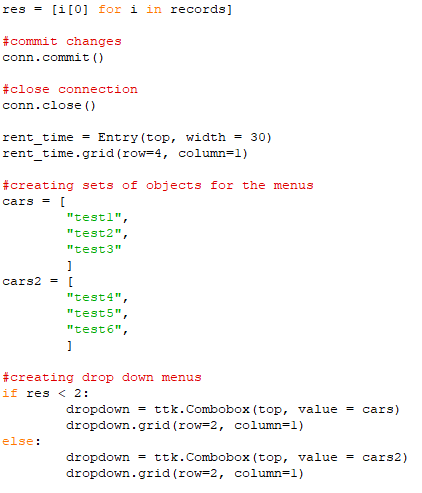


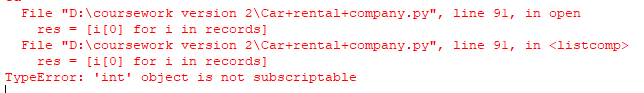
This was the final output:

However, the no matter what experience was entered the second set of values outputted. To solve this, I imported ttk from tikinter and created a different type of drop-down menu as shown below.



To solve the problem of outputting the correct drop-down menu I tried a different method of comparing the tuple to an integer shown in the code below

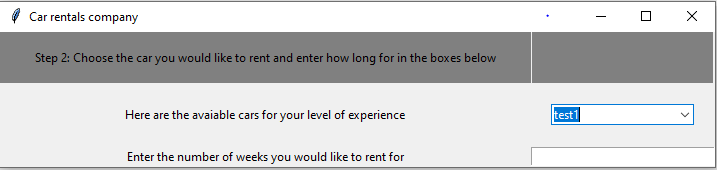


However, this method did not work as shown in the error message below.

Instead, I used a third method which worked. Rather than setting another variable equal to the first item in the “records” tuple I used “records [0]” to call the first integer to be compared as shown below



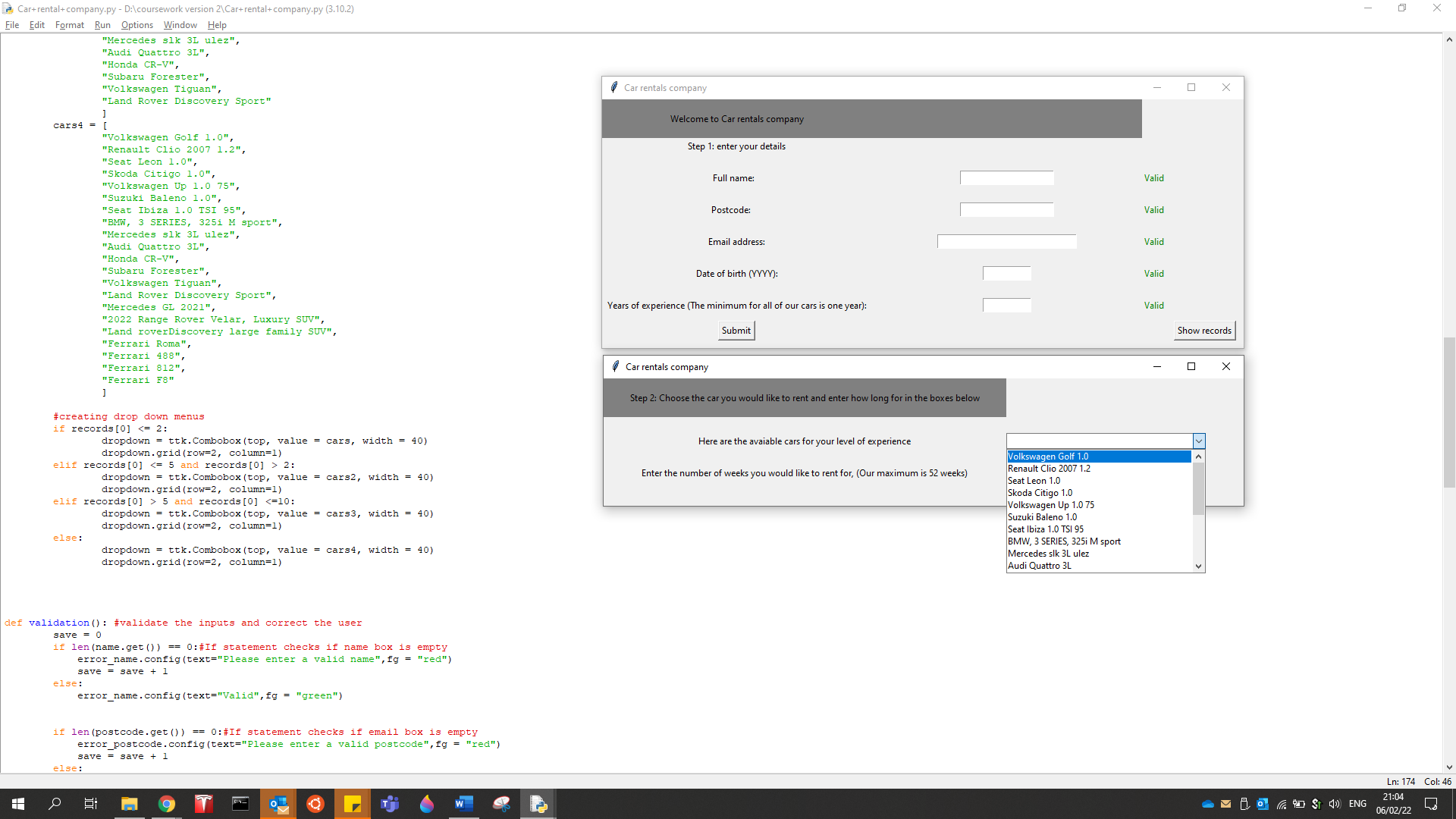
As this is just a test run, I used values “test1”, “test2” etc rather than actual cars so that I could check if the right entry boxes would be created, and they did as shown below when 1 was entered into the experience box.



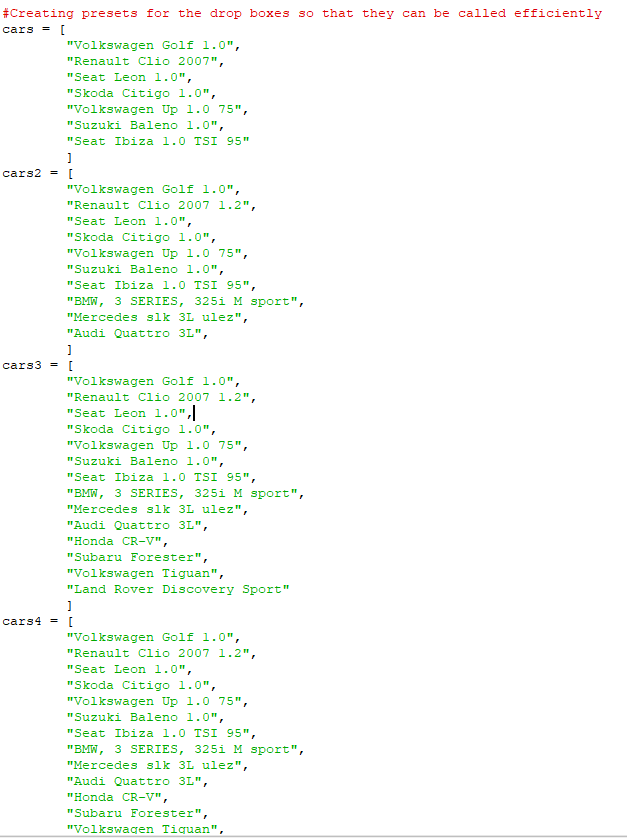
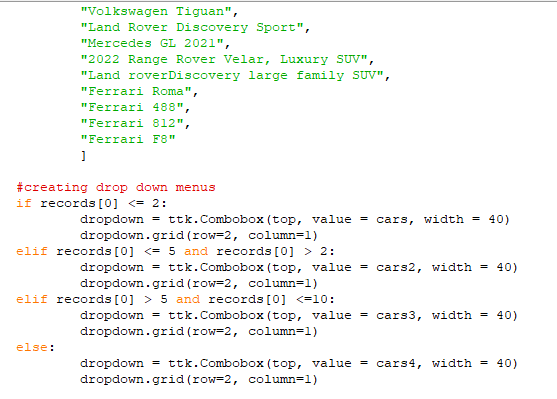
*Validation:*

This sections validation will be created in the next stage when a button is placed on the second window which would open the third confirming everything but will only open that third window if the entry box is filled.

*Review:*

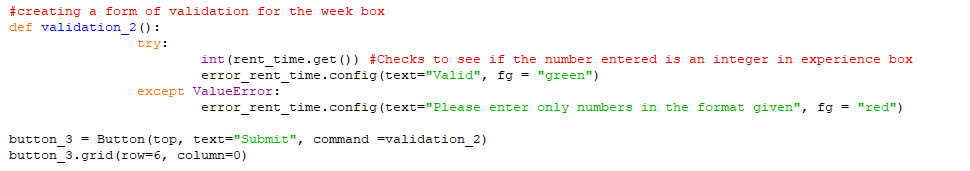
After many attempts and multiple fails I got the drop-down menus to work with different entries for each bracket of years of experience. So far, the GUI looks like this:

The code for this stage is below:

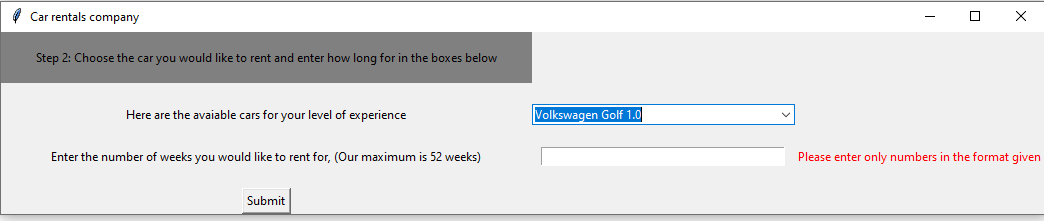


Stage 6 – Opening the third window and adding validation

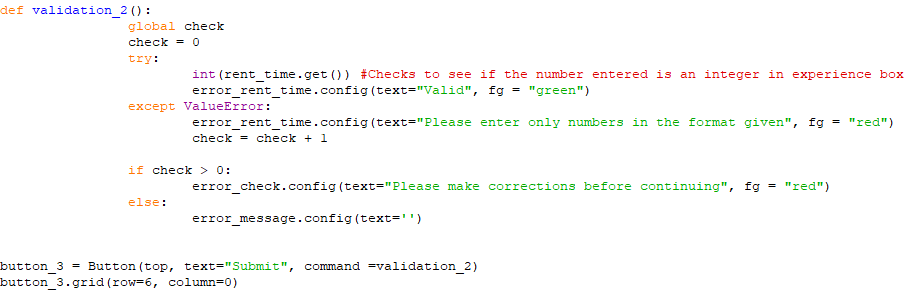
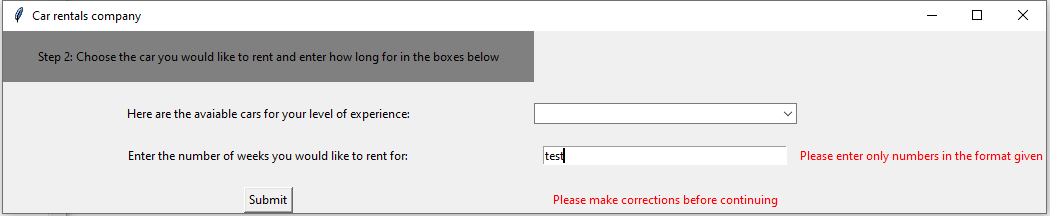
*Validation*

For this stage I plan to add a new button on the second window that saves the choices selected by the user to their respective database. Before doing so however I will create a form of validation ensuring the user entered their value for the number of weeks.

This was similar to the validation used before and when ran an error popped up as I had forgot the “.get” when checking if the entry was an integer.

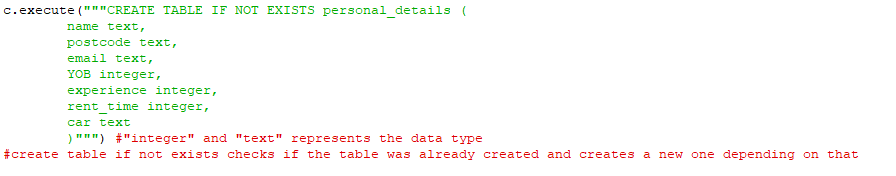
Working output so far:

In addition to this I wanted to add another message next to the button the same as on the first window to keep the consistent design.

At the end of the statements other functions will be ran. The next two to write are the saving to database function and opening the third window.

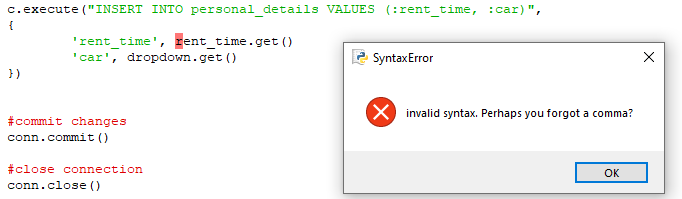
*Saving to database*

The main objective here is to create a function (that is called if the numbers entered are correct) that saves the number of weeks and the car chosen. First, I need to go back and add in two more columns for these two new variables and use a similar save function as before.

Code so far:

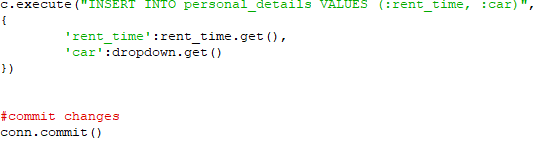


First test:

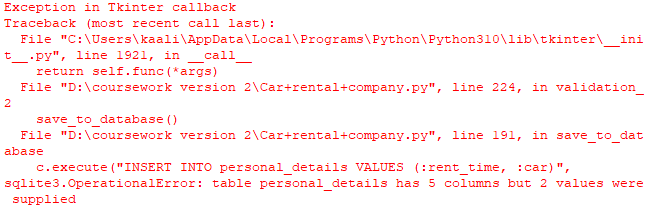


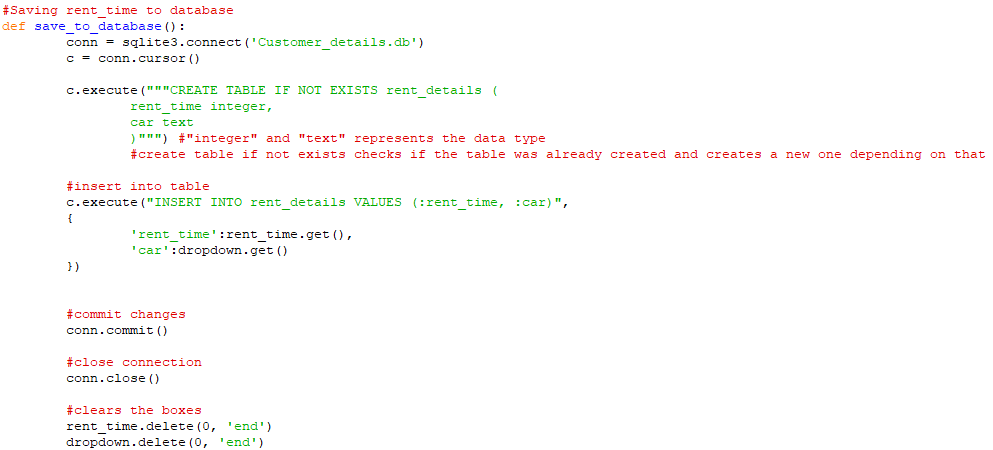
Here I realised I should be using colons not commas.

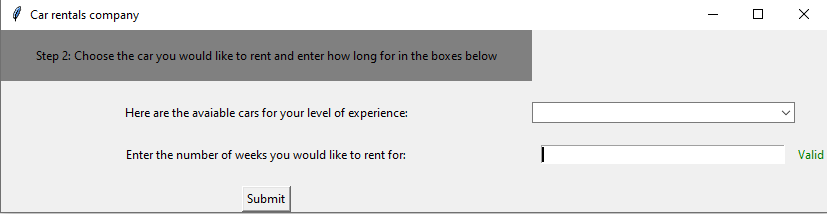
Fix:



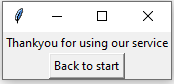
When ran this error occurred:

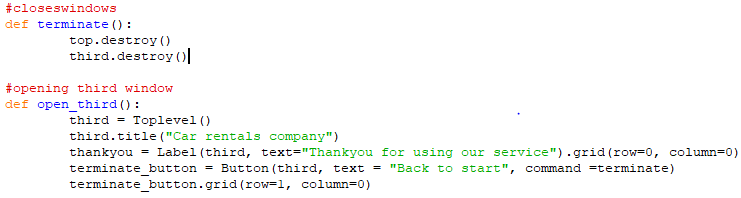


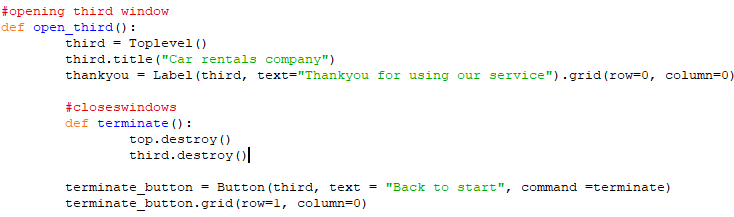
This is because all the columns must be filled as the set of lines I used is for a new row in the database. Instead, I decided to create a separate table alongside the first so hold this data.

I know for sure this time the function worked as the boxes cleared with no errors which was the final task of the function.

*Opening the next window*

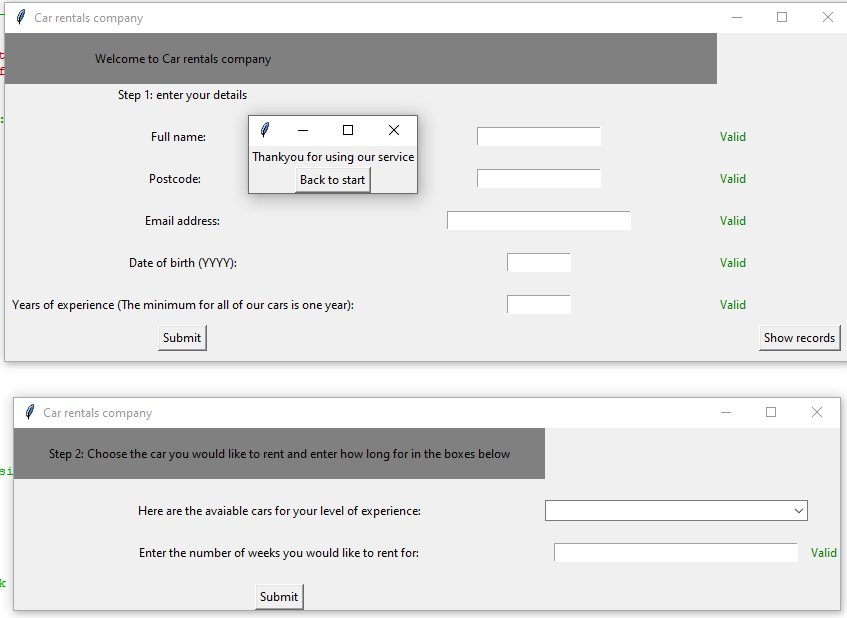
Next, I need to open the final “thankyou” window which will just ask the user to end the program which will terminate all the windows.

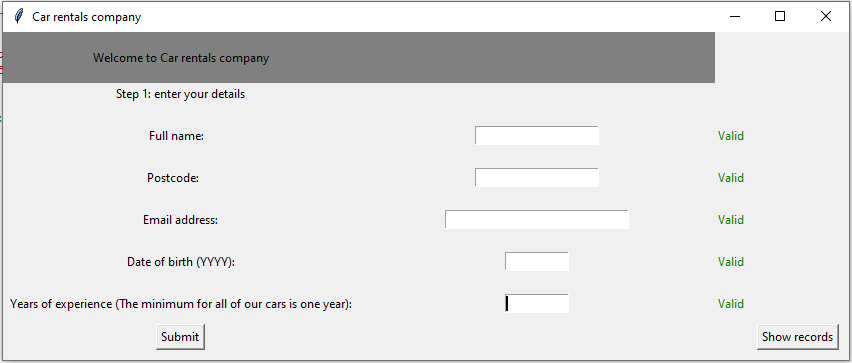


The first function’s goal is to close both new windows that opened in the process, however only the second window was closed so to solve this I simply placed the terminate function within the open function as “third” was defined after.

*Review*

In this stage I completed multiple tasks. I added in a form of validation for the number of weeks and saved the details entered to a separate table in the database. The program’s initial use is now complete however I would like to add a final stage where I create a staff option at the start where the “show records” button sits. Most of the success criteria is complete.

Before:

After:

Stage 7 – Staff access

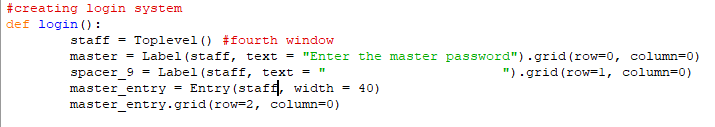
In this section I will be aiming to replace the “show records” button with a “staff access” button where staff can use the same program to search the two tables in the database.

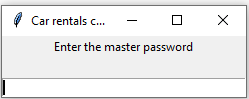
First, I renamed the button and changed the text within it:



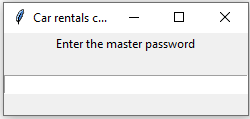
*Login system*

Next, I created a new function where a new window will open with a box to enter the master password.

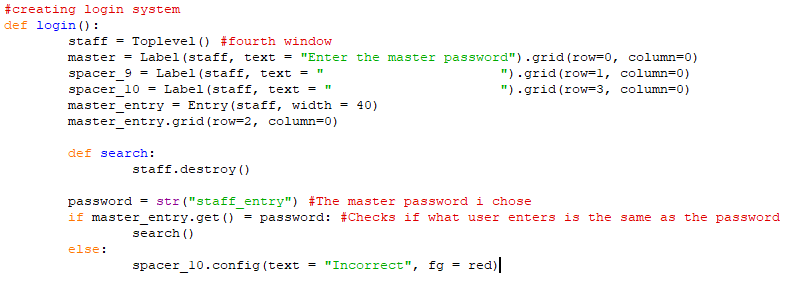
Before creating an actual login system, I first had to open another window which is shown below, once again using spacers to keep the consistent clean look.



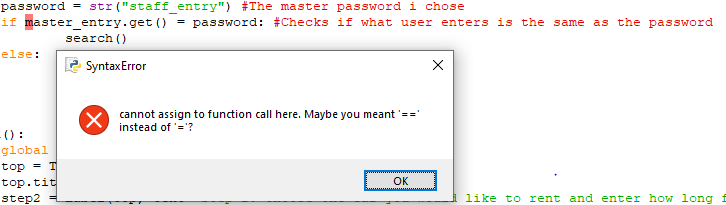
When ran this window opened with no errors. However, I didn’t like the look of the entry box, so I added another spacer below it to make it seem less cut off.

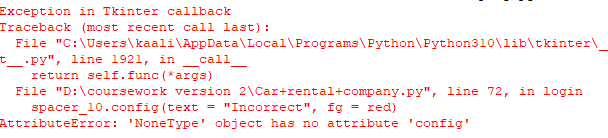
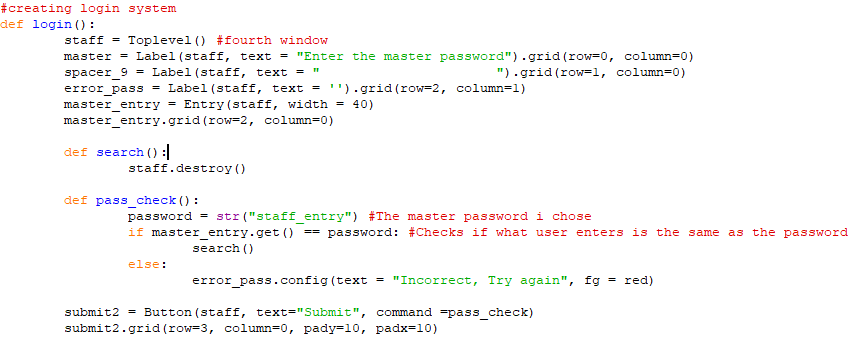


The next part is to create an if statement to check whatever is entered into the entry box against the passcode I choose.

In this screenshot I created an if statement that che4cks if the password was correct. If so, it runs another function that just for this test will terminate the window. If the password was not correct a message outputs to the user informing them it was incorrect.

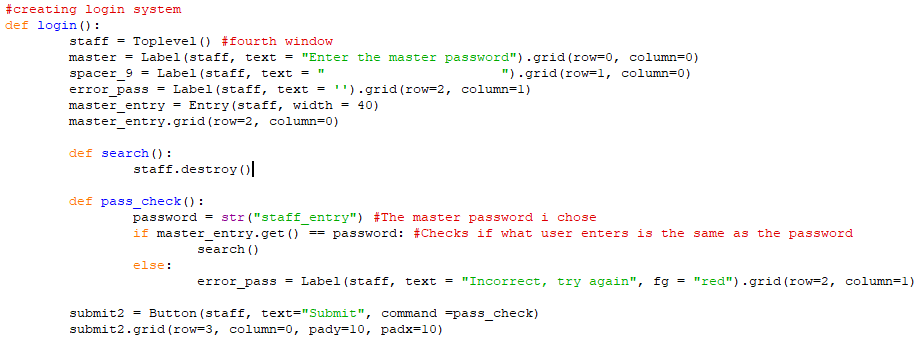
When ran a logic error showed up informing me to change the signs used to compare.

I ran into another problem for the error message.

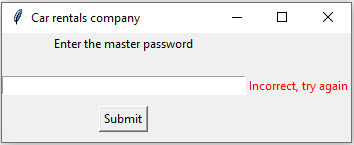
This was a simple case of selecting the incorrect label. I also realised that I did not include a submit button which I added in the function below.

Here I added a separate function for the comparison and continuing to the next part. However, when ran another error showed up related to the “.config”.

So, to solve this I decided to remove the configuration of the pre-set label and just create a new one at that moment:

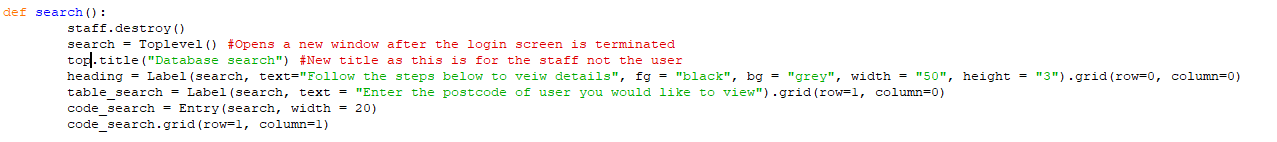


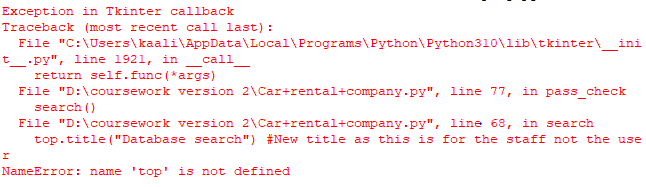
This returned an output with no errors and the window looked similar to the others keeping that consistent design.

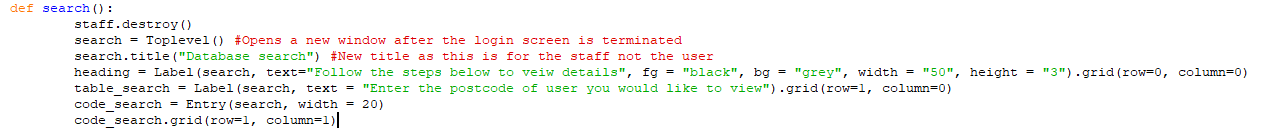


*Creating the search window:*

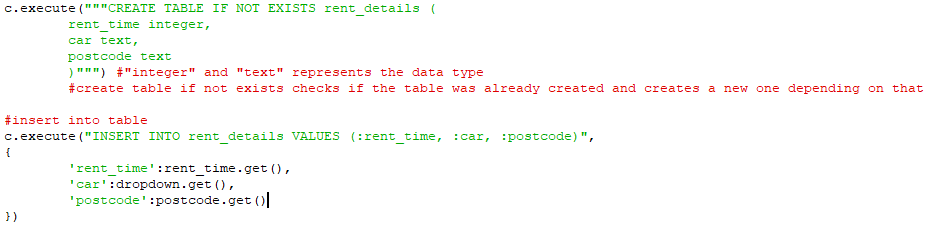
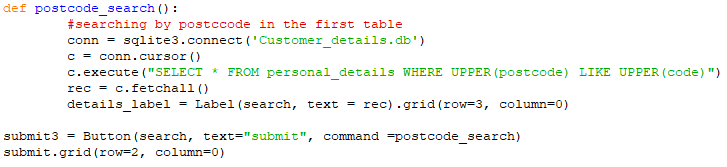
In this section I will expand on the “search” function and the look of the staff window. First the objective is to create a new window within the function after the login page is closed.

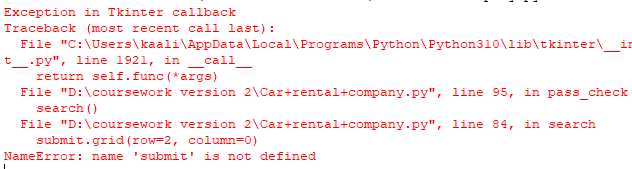


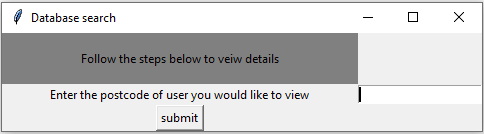
When this code was running this error was outputted when the correct password was entered, and a blank window opened. This was because I used “top.title” from the window earlier rather than “search.title”

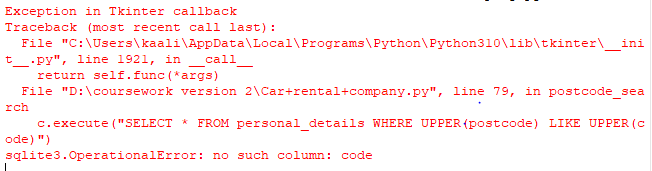
This was solved through the code below:

As I will be searched two tables with the given postcode I need to go back and make a column in the second table called postcode and make sure the variable is store there too so that the correct information is shown.

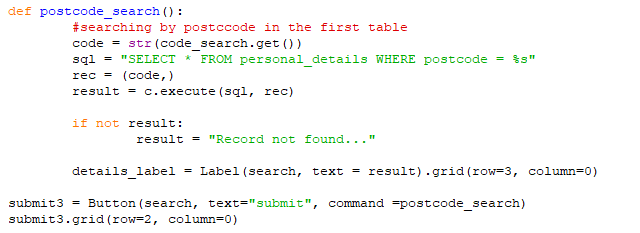
When ran there was no errors so I can assume the saving was successful. Next, I need to use sql to search and output all relevant details to the person with that postcode. However, this will be written in a function that will be called from another submit button on screen.

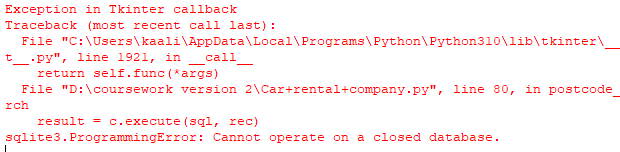
This is what I had codded so far before running the program. When ran an error appeared as I had not specified the correct “submit” button when placing it in its position.



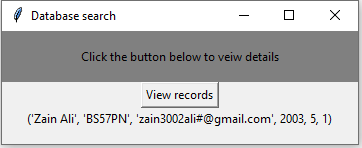
The button was outputted as expected with no errors however when a postcode was entered to search the table this error occurred

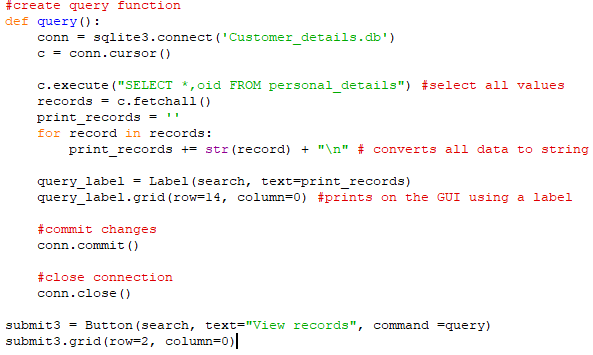
I tried a different approach to this as shown below where I set specific values for what the postcode show be so that the correct details are outputted.

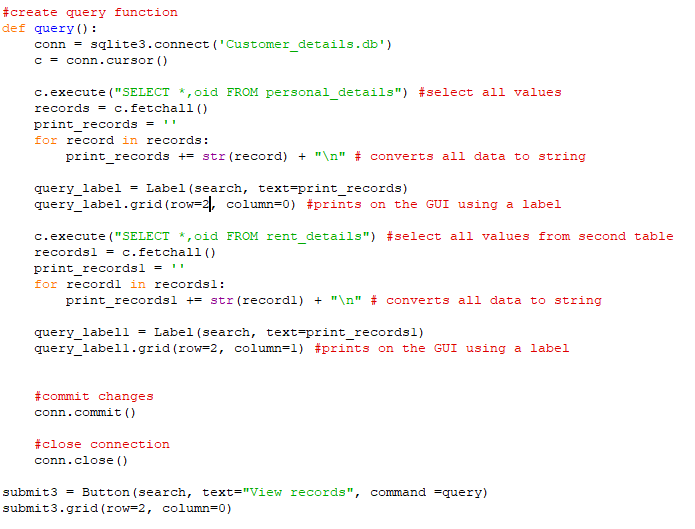


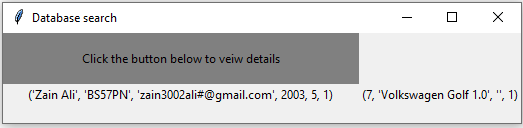
I ran into an error however as I did not open the connection to the database.

This whole method proved to not work as I wanted to so I decided to switch the objective and make it so that when the user clicks a button all the data shows on screen.



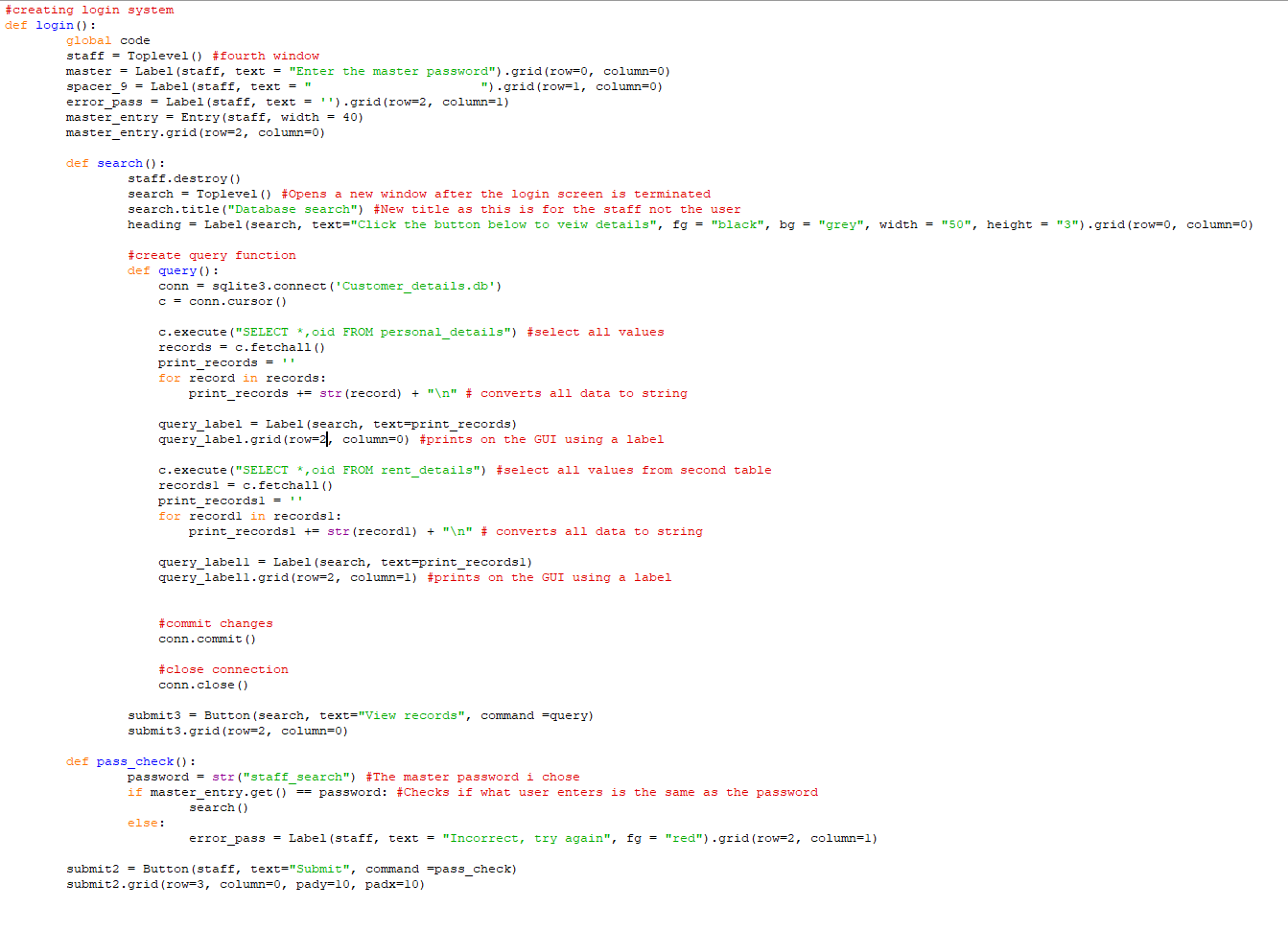
Next is to output the second tables data which can be matched by index number which is built into sql. I used the fetch all function to fetch all the relevant data not just the first entry.

This returned no errors and outputted the correct data:



*Review*

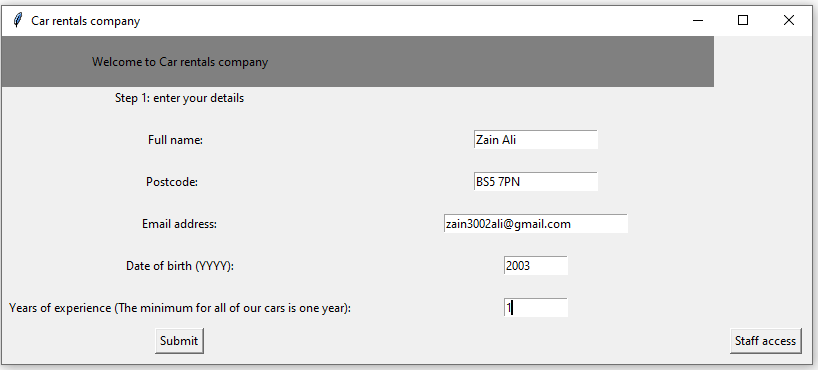
Though this final stage was not planned I still thought it was crucial to do so to have a method to read the database and ensure all changes occurred.

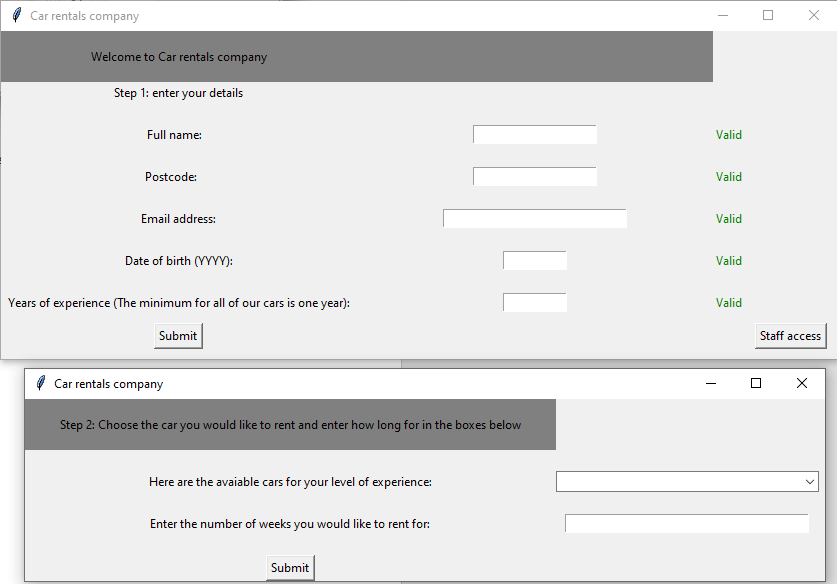
Here is the final code of this section:

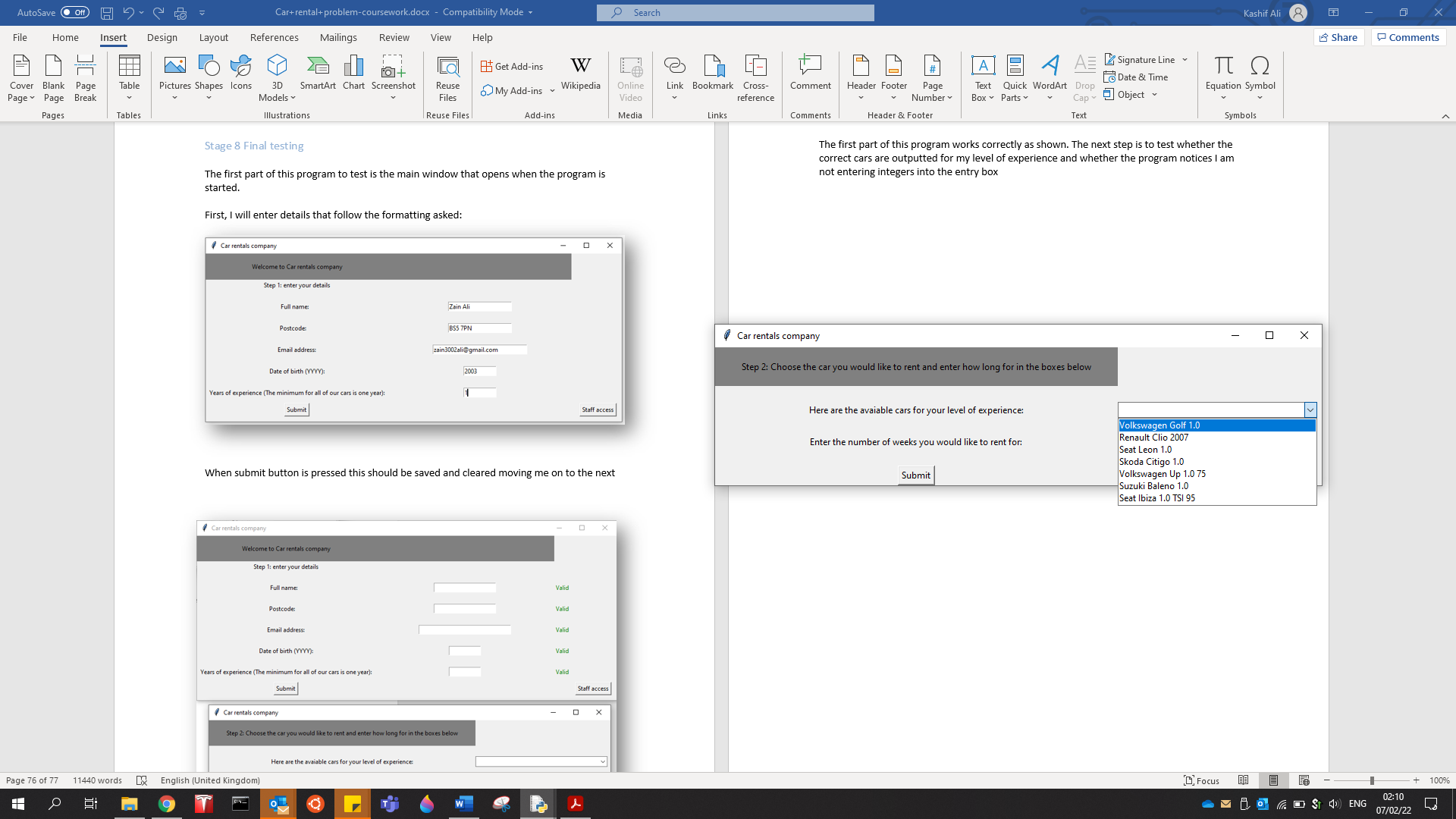
Stage 8 Final testing

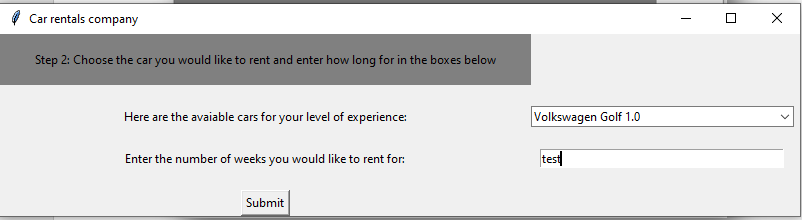
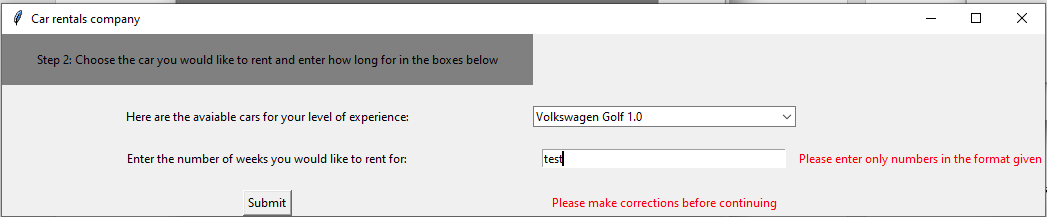
The first part of this program to test is the main window that opens when the program is started.

First, I will enter details that follow the formatting asked:

When submit button is pressed this should be saved and cleared moving me on to the next step.

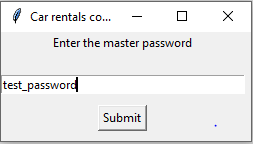


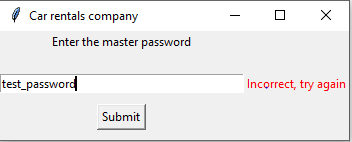
The first part of this program works correctly as shown. The next step is to test whether the correct cars are outputted for my level of experience and whether the program notices I am not entering integers into the entry box

As shown above the correct list does appear now the entry box’s validation must be tested:

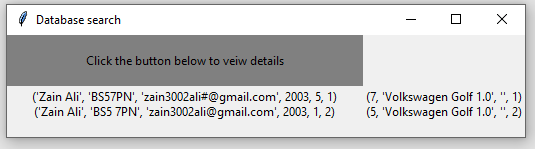
The validation also worked when text was entered rather than integers. So the next step is to test and see if all this data is saved to the external database separately. This can be done by the staff access button so as I perform this test I will also be testing the login screen

The correct master password is “staff\_search” so this entry below show be in correct and the program should not proceed onwards.





When the correct password is entered another window is opened and this login window is closed which was also intentional. The button has already been clicked and replaced in the screenshot below and this also proves the saving of data entries was successful.



Checklist

This is a finished version of the checklist I created in the development stage as I have now finished the testing and the coding of this program.

|  |  |
| --- | --- |
| **Test** | **Functional?** |
| Text entered in entry boxes | Yes |
| If submit button checks the data entered | Yes |
| If submit button saves to database | Yes |
| If submit button clears boxes | Yes |
| If submit button opens new window | Yes |
| If window outputs correct cars (test with multiple different years of experience) | Yes |
| If car database is updated on stock availability | No |
| If car database is updated with how long car is out to rent | No |
| If “back to start” button restarts program | Yes |

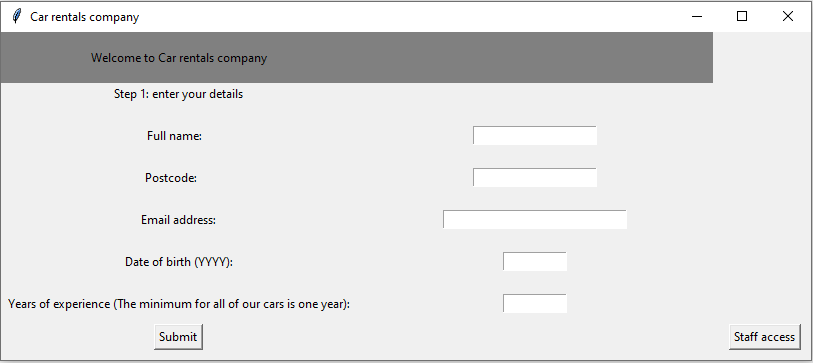
**Evaluation**

**Criteria met**

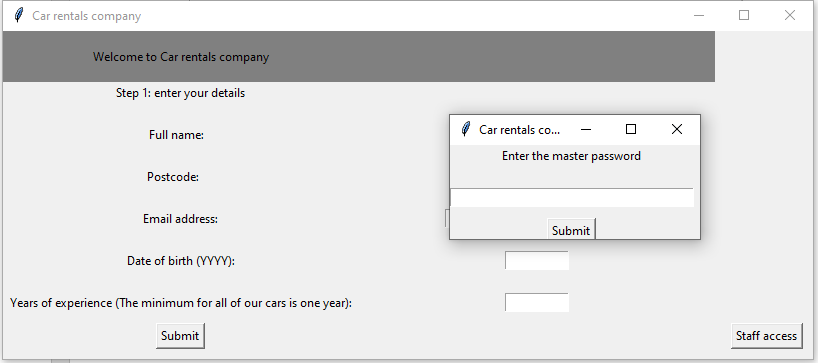
Success Criteria

|  |  |
| --- | --- |
| **Criteria** | **Complete?** |
| Main window that pops up when ran | Yes |
| Buttons and text boxes that have funtions | Yes |
| Lightweight consistent design on each window | Yes |
| Drop down menus | Yes |
| Built-in calender | No |
| A databse for personal details that can be read from and updated | Yes |
| Instructions at each stage | Yes |
| Output relevent cars after checking against data | Yes |
| Validation | Yes |

Evidence

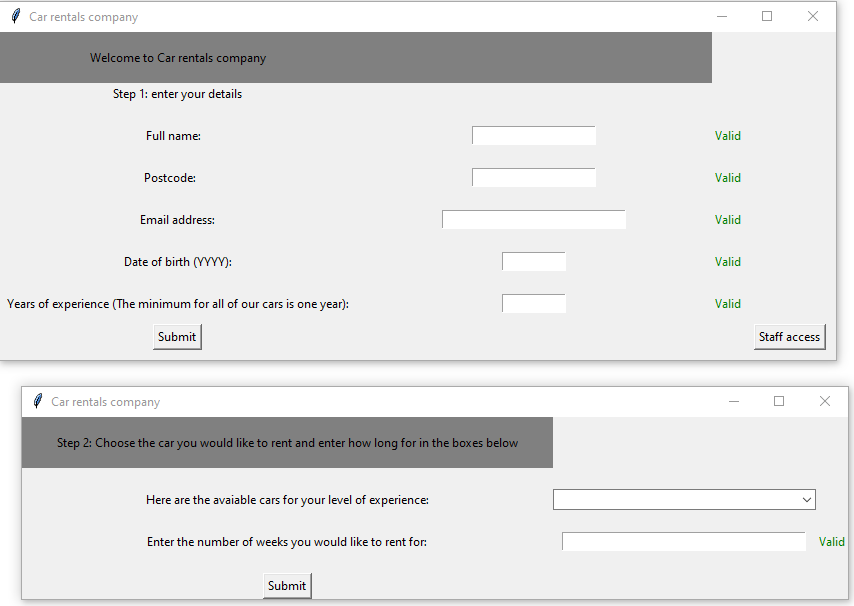
**Main window that opens when ran:**

Above is what is outputted when the program is ran. This is the main root window that every other sub window runs off and links to. If this window closes all others would but if another window closes this window stays open as it is the root.

**Buttons and text buttons that have functions**

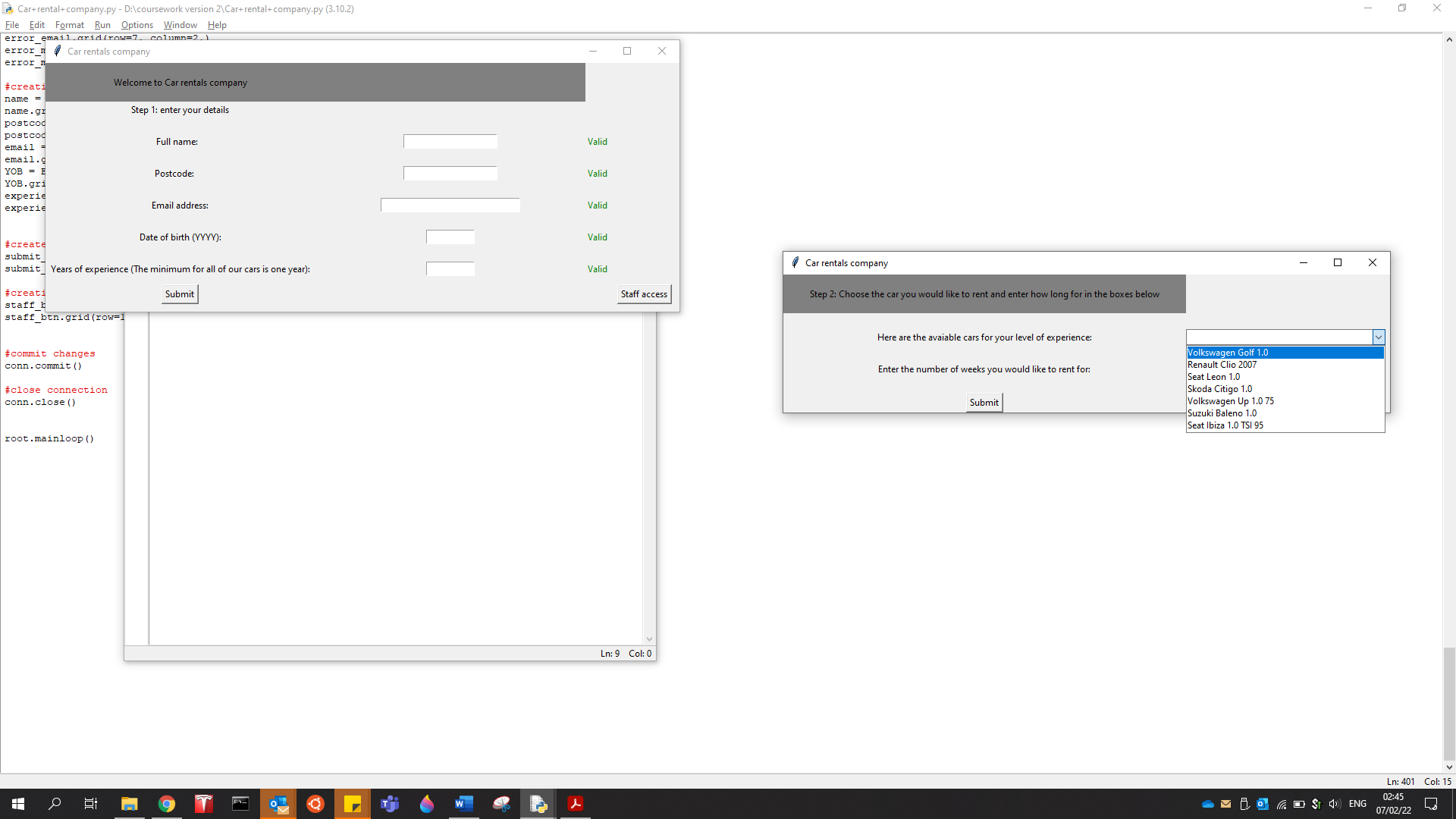
All the windows in this program use various buttons and entry boxes but this is the best to show when proving they have functions and are useful. The button “staff access” opens this separate login screen for a user to view the database (if they know the password). Also the entry boxes have set datatypes assigned to them so only specific data can go through the system.

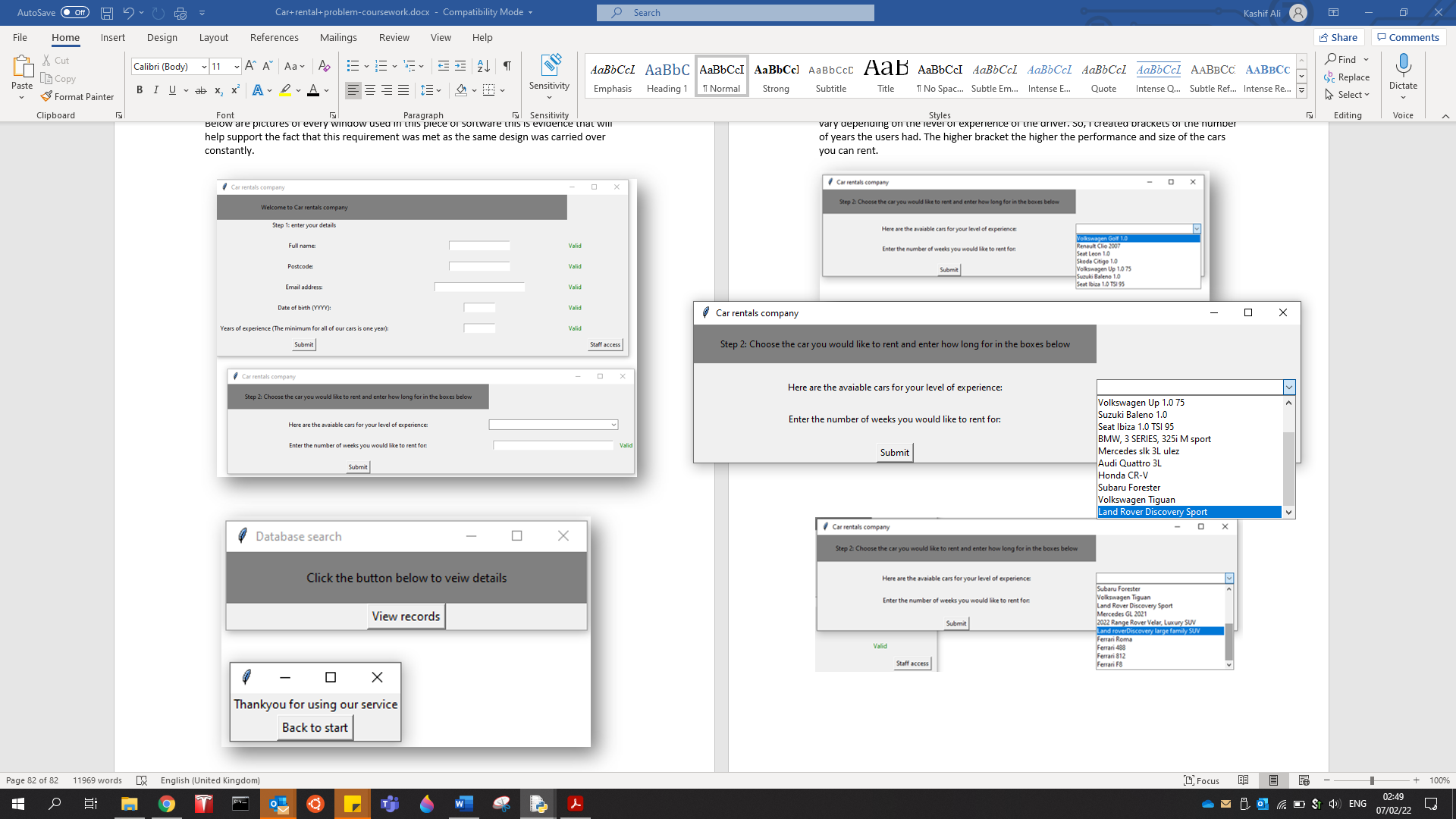
**Lightweight and consistent design:**

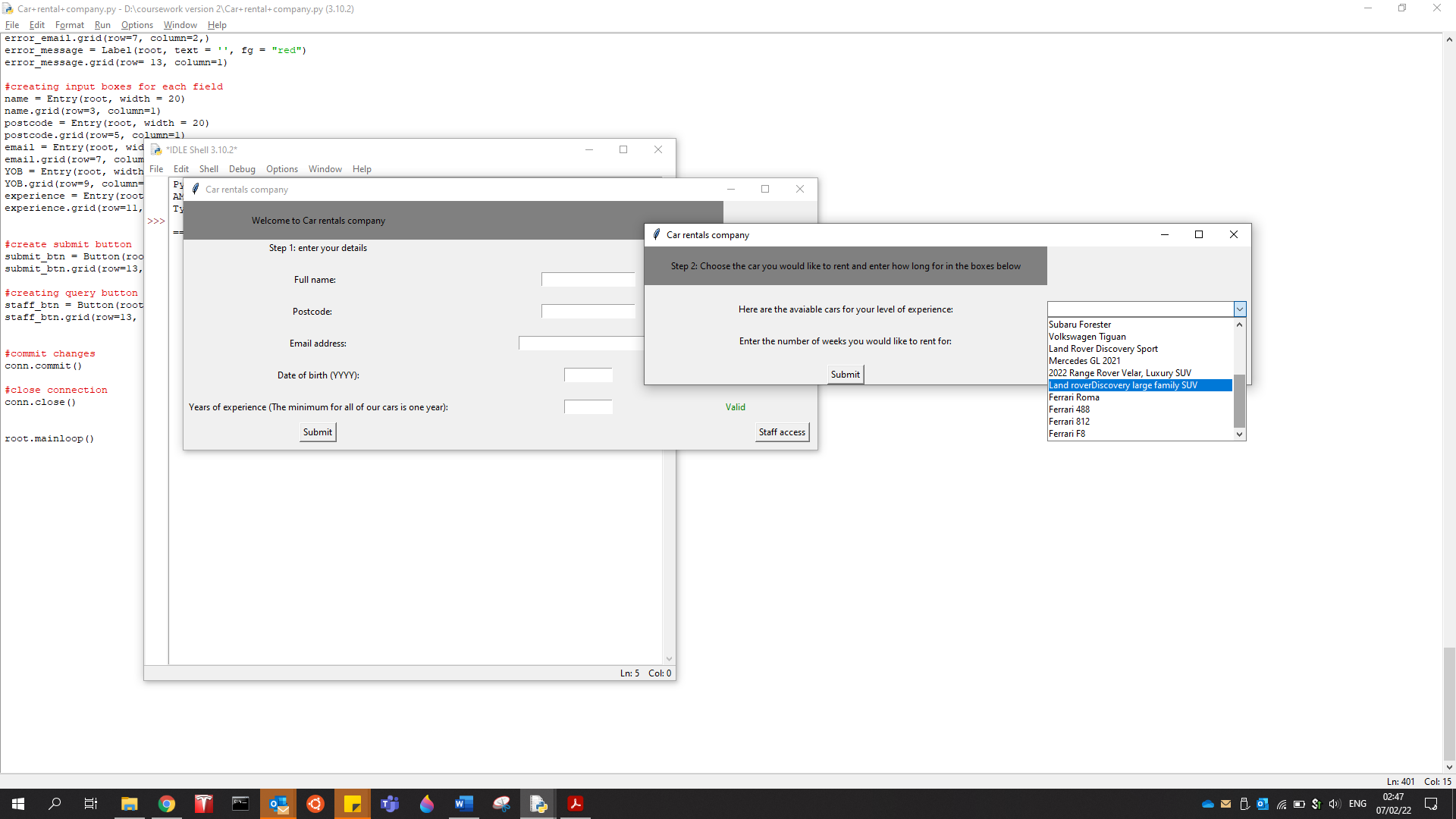
Below are pictures of every window used in this piece of software this is evidence that will help support the fact that this requirement was met as the same design was carried over constantly.



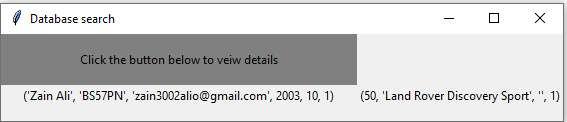
**Drown down menus/Output relevant cars after checking against data:**

I used drop-down menus when listing available cars. The list of contents in the box would vary depending on the level of experience of the driver. So, I created brackets of the number of years the users had. The higher bracket the higher the performance and size of the cars you can rent.





**A databse for personal details that can be read from and updated:**

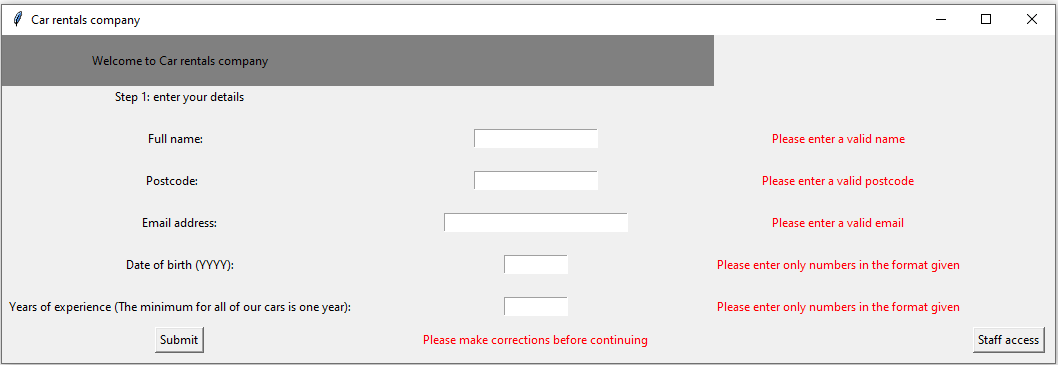
This can be proofed from the staff access section of the code:

When the button is clicked all the data in the tables would be outputted orderly. In this case it seems like te databse was whiped for testing reason.

**Instructions at each stage:**

As shown in the images of each window above there are clear readable instructions on what you should do at each step. Also when you enter details incorrectly there are messages informing you that you have made a mistake and that the prgram is not continuing because of that.

**Validation:**

Any situation where the user has to enter any kind of inputs to interact with and continue the process I used validation method. For example:

These are the same messages that have been repeated throught the prgram in any situation where validation was necesesary

Limitations:

The two maion limitaitions and drawbacks of this piece of software would be the renting time input the car database.

The problem with the renting time input is that it is sclaed in weeks on the program which would not be the most specific. To solve this, in the futur I could add a calender using a python mod. This would allow the user to pick a specif date and time they wan to return the car.

The second limitation is the car database. Because I have not used an external database purely for the cars available, it would mean that there is no way of telling whether a car is out for rent or not as there isnt a system constantly being updating about that. Instead I opted for a table which constainted all the detail fro the second step. But even this isnt enough as it does ot link to the personal\_details table so it can only be used for outputs which greatly limits what you can do with a database.

**Main code**

