Contents

[**Version 1 implementation** 3](#_Toc99960237)

[Overall design 3](#_Toc99960238)

[Version 1.1 implementation 7](#_Toc99960239)

[Testing 11](#_Toc99960240)

[Errors 11](#_Toc99960241)

[Analysis 11](#_Toc99960242)

[Version 1.2 implementation 11](#_Toc99960243)

[Testing 16](#_Toc99960244)

[Errors 16](#_Toc99960245)

[Analysis 16](#_Toc99960246)

[Version 1 conclusion 16](#_Toc99960247)

[Version 2 implementation 16](#_Toc99960248)

[Overall design 16](#_Toc99960249)

[Version 2.1 implementation 19](#_Toc99960250)

[Testing 22](#_Toc99960251)

[Errors 22](#_Toc99960252)

[Analysis 22](#_Toc99960253)

[Version 2.2 implementation 23](#_Toc99960254)

[Testing 25](#_Toc99960255)

[Errors 25](#_Toc99960256)

[Analysis 25](#_Toc99960257)

[Version 2.3 implementation 25](#_Toc99960258)

[Testing 28](#_Toc99960259)

[Errors 28](#_Toc99960260)

[Analysis 29](#_Toc99960261)

[Version 2 Conclusion 29](#_Toc99960262)

[Version 3 implementation 29](#_Toc99960263)

[Overall design 29](#_Toc99960264)

[Version 3.1 implementation 31](#_Toc99960265)

[Testing 35](#_Toc99960266)

[Errors 35](#_Toc99960267)

[Analysis 35](#_Toc99960268)

[Version 3.2 implementation 36](#_Toc99960269)

[Testing 39](#_Toc99960270)

[Errors 39](#_Toc99960271)

[Analysis 40](#_Toc99960272)

[Version 3 conclusion 40](#_Toc99960273)

[Version 4 implementation 40](#_Toc99960274)

[Overall design 40](#_Toc99960275)

[Version 4.1 implementation 42](#_Toc99960276)

[Errors 44](#_Toc99960277)

[Testing 45](#_Toc99960278)

[Analysis 45](#_Toc99960279)

[Version 4.2 implementation 45](#_Toc99960280)

[Errors 50](#_Toc99960281)

[Testing 50](#_Toc99960282)

[Analysis 51](#_Toc99960283)

[Version 4.3 implementation 51](#_Toc99960284)

[Testing 55](#_Toc99960285)

[Errors 55](#_Toc99960286)

[Analysis 56](#_Toc99960287)

[Version 4 conclusion 56](#_Toc99960288)

[Version 5 implementation 56](#_Toc99960289)

[Overall Design 56](#_Toc99960290)

[Version 5.1 implementation 59](#_Toc99960291)

[Errors 61](#_Toc99960292)

[Testing 61](#_Toc99960293)

[Analysis 61](#_Toc99960294)

[Version 5.2 implementation 62](#_Toc99960295)

[Errors 65](#_Toc99960296)

[Testing 66](#_Toc99960297)

[Analysis 66](#_Toc99960298)

[Version 5.3 implementation 66](#_Toc99960299)

[Errors 80](#_Toc99960300)

[Testing 80](#_Toc99960301)

[Analysis 81](#_Toc99960302)

[Version 5 conclusion 81](#_Toc99960303)

[Full code print out: 82](#_Toc99960304)

**Version 1 implementation**

## Overall design

Window will have a light blue background with light pink labels

Each window will have the same title with a hyphen saying what the window is used for e.g., admin page

The name of each window will follow the title closely e.g., admin page or appointment booking

Each window will have a title label which has a large black font and a light pink background unless otherwise indicted

Each label will have a light pink background with a smaller black font unless otherwise stated

Each button will be a yellow colour with a black font matching the other labels

Entry boxes will have a grey background

Spaces will be created using the same colour as the background

Characters are using the font Calibri, with a font size of 20. This is to be kept consistent unless otherwise stated. Keeping this consistent will allow the program to keep consistent spacing and formatting between labels (titles and buttons etc).

Overall aims

To allow a user to book, view and amend an appointment, ensuring that users cannot book two appointments at the same time using validation

The user should be able to change the data, time, and treatment if there was an error upon entry

The program would allow the viewing of appointments for a day or week or display specific appointments

A window for graphs where various analytics can be viewed about the business

Stock levels will also be kept within a database so warnings can pop up when stock levels are insufficient

Version 1 aims

The overall aims for this version would be to create 3 windows which each follow the design stated previously. The “Main Menu” window would be produced which would allow navigation to either “book an appointment” or “Admin page”. The program should be able to navigate between each window back and forth through buttons.

The user should be able to make an appointment where this would be stored within a database but at this stage of implementation the date and time would not be validated or cross referenced with other appointments.

To allow the user to view their appointment through the user of their surname, later in the program’s development, the aim is to give each user and their appointment a specific id to be used as a primary key when identifying them and their details.

**Interface design**

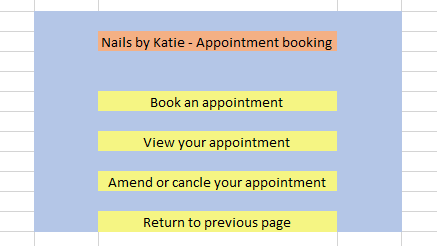
This is an interface design produced based off the client’s requirements:



“MainMenu\_Title\_Lbl”

“MainMenu\_Admin\_bttn”

“MainMenu\_App\_bttn”



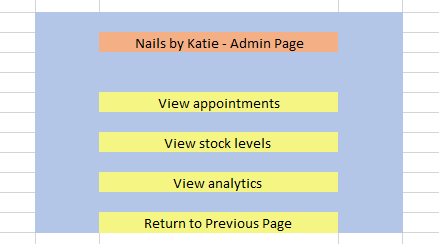
“AppMenu\_Title\_lbl”

“AppMenu\_book\_bttn”

“AppMenu\_view\_bttn”

“AppMenu\_ammend\_bttn”

“AppMenu\_return\_bttn”



“Admin\_Title\_lbl”

“Admin\_app\_bttn”

“Admin\_stock\_bttn”

“Admin\_analytics\_bttn”

“Admin\_return\_bttn”

**Algorithm Design**

For main menu:

Create frames to place the labels in

Create window geometry for the main menu

Create the title label and use frames to position it

Create the other buttons and use frames

The button will withdraw the main menu and update the new window upon being pressed

For admin menu:

Create a frame to place labels within

Create the window geometry for the admin page

Create each label and title label and position with frames and ‘.pack’

This page will have no functionality during this version.

For appointment menu:

Create a number of frames to place labels in

Create the window geometry for the appointment page

Create title and widget labels and use ‘.pack’ to position it

Book appointment and view appointments will have functionality in this version and will be designed as followed:

Booking an appointment will ask for a surname, name, date, a time and what type of appointment they would like. then this is saved to a database

Viewing an appointment will read the database, ask surname to identify the user’s appointment and then display the times/dates with the appointment type.

The ‘.pack’ method is used here for 3 reasons. One of the reasons would be the need for labels to appear and disappear in certain spaces which ‘.pack’ allows using ‘.forget’ as well. Another reason is how the appearance of the window requires consistent spacing within frames created within the window. A final reason is that I have more experience with using the ‘.pack()’ within tkinter and python which allows high capabilities using this method.

**File design**:

Csv will be within the same file location as which the program is run from and will be named ‘Appointments.csv’

**Structure design**

A picture containing chart

Description automatically generated

## Version 1.1 implementation

The first part of this version was creating a main menu window, creating a title and buttons with their design and names corresponding to that of the design.

Window geometry for the Main menu

Graphical user interface, text, application

Description automatically generated

Creating frames for the window

These create the labels for the window. Each label has a corresponding name along with various formatting such as position, text, colour and size. The font size here is important because it needs to be reused when creating spaces in order to format the page and its labels

These are spaces. They are labels within the frames which are used to separate each label with equal spacing. Without them the widgets would be squashed together

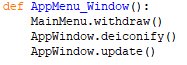
Graphical user interface, text, application

Description automatically generatedAfter creating the first window with 2 buttons a title the output was as such:

After each button was created in the code, it was tested to confirm that it displayed it correctly and followed the design.

Next the implementation of the appointment menu was done. This involved created the geometry for the window and then created a corresponding function which takes away the main menu and updated the Appointment window to display.

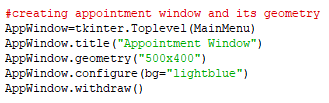
The code to set up the function and geometry is displayed below:



Closed the Main Menu

Loaded App Window

Opens the new window



Creates window name

Creates window dimension

Configures background to light blue

Graphical user interface, text, application

Description automatically generatedOnce the window and the corresponding function was implemented; the code and the button was tested to check if the window could open itself and close the Main Menu. Initially this created a syntax error:

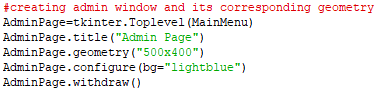
Graphical user interface, text, application

Description automatically generatedThis created an error because the syntax should have been ‘deiconify’ instead of ‘deinconify’. Fixing this error allowed the program to work as intended. Clicking the ‘booking an appointment’ button opens the window shown below:

Graphical user interface

Description automatically generated

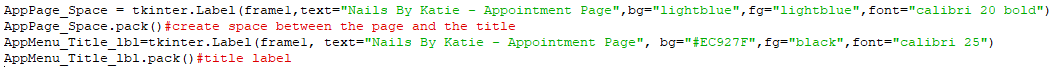
Testing this allowed me to repeat the process for the admin log in window.



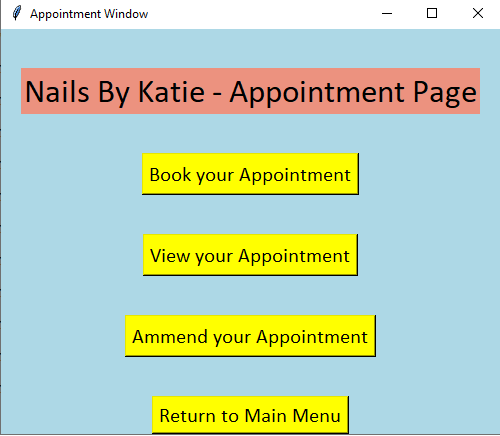
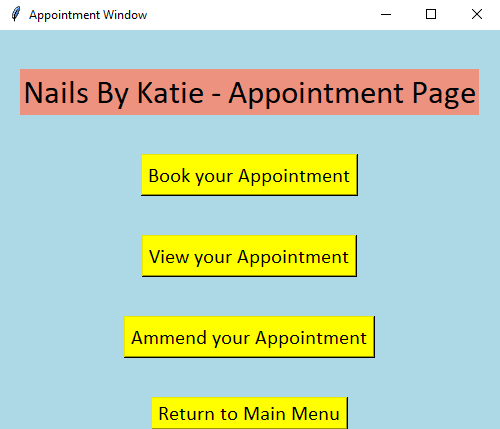
This completed the same process and uses a similar function to the one shown above.

Graphical user interface, application, Word

Description automatically generatedOnce both windows were working and displayed corrected, titles and buttons were added to the Appointment page.

This implementation was tested as it was coded to ensure that it was appearing as intended within the window.

Each of the buttons were the coded and tested as each button was implemented. Whilst these buttons have no functionality as of yet, they will within upcoming versions.



Text, letter

Description automatically generatedWhilst implementing the rest of the buttons within the appointment page, the size of the page was too small and not long enough to facilitate the last button as shown within the design. To fix this the dimensions of the appointment window was increased to allow the button to fit better.

 This was the code used to associate the command which opens the main menu and closed the appointment window. This allows the user to traverse through the menus. This was tested by clicking both buttons and ensuring that the program can repeatedly move between both windows.

This is a function which allows the user to go backwards to the main menu. This function can be used for both the admin page and the appointment page, rather than creating 2 separate functions

As the final part of this version, I implemented the title and the buttons onto the admin page. Since the admin page had already been implemented and the button which takes the user from the main menu to the admin page had been included and tested, allowed me to work on implementing the buttons.

Initially all 5 frames were made for the window:

Create a frame and place it within the window called “AdminPage”.

Text

Description automatically generated

Places the frame within the window

Configure the colour of the frame to light blue which matches the initial design.

Graphical user interface, text, application

Description automatically generatedThis was then tested and run after the first frame was implemented to ensure that the program would run without an error within the python shell. After this was tested, the rest of the frames where codes and tested.

Following the frames, the title and buttons were added. The title and then the buttons were coded one at a time and tested each time they were implemented. The final output is as follows:

Text

Description automatically generatedThe code is similar to that of the appointment page with a few adjustments to text and comments.

### Testing

The testing for this version largely involved ensuring that the labels and buttons coded were placed within the window as per the design element. This included implementing a label and ensuring it matched its corresponding design. Throughout this process minor adjustments were made to the size of labels to ensure sufficient spacing between them.

### Errors

This version had a few small errors. This was limited to a syntax error where ‘deinconify’ was used where ‘deiconify’ should have been used. This was used when creating a command which upon the press of a button, opened the main menu. This error was found when testing that the program could open the appointment window from the main menu.

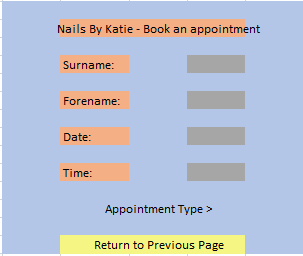
### Analysis

The implementation of this version was well done in the sense that there was little errors and I was able to produce multiple windows which matched the design element. This version accomplished its aims of creating all 3 windows with the ability to move between all 3 windows. The aim for the next version would be to allow the user to book an appointment, and then save the data to a csv. The csv file will consist of a surname, forename, date, time and appointment type.

## Version 1.2 implementation

This version included the implementation of a new window which allowed the user to book an appoint and save it to a csv. This included a new window where the user would input various data to book an appointment. This version will not have any validation and will allow users to book the same appointment on top of another appointment. In another version each user will be given a unique identifier which would allow a user to book, view and amend appointments using their identifier.

The design for the book appointment page is as followed:



“BookApp\_forename \_Lbl”

“BookApp\_Title\_Lbl”

“BookApp\_surname \_Lbl”

“BookApp\_surname \_Ent”

“BookApp\_forename\_Ent”

“BookApp\_Data\_Lbl”

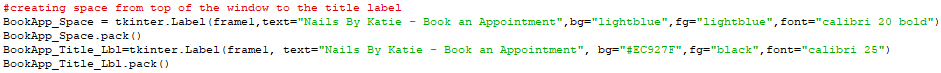
“BookApp\_Data\_Ent”

“BookApp\_Time\_Lbl”

“BookApp\_Time\_Lbl”

“BookApp\_Type\_DropDown”

“BookApp\_Return\_Bttn”

Text, letter

Description automatically generatedShape

Description automatically generated with low confidenceinitially the frames for the window were created. All 5 were created and then a space and the title label were implemented and tested. The code for the frames, space and labels are similar to the structure of the code of other similar elements. Parts of the code is shown below:

A picture containing timeline

Description automatically generatedWhilst implementing the labels within the page, some of the spacing was un-alligned with where it should be. This was due to the fact that the space before hand (between the title and surname label) was placed within the first frame, frame 1. To fix this the space was changed from frame 1 to frame 2. This change produced an output as:

Graphical user interface, text, application

Description automatically generated



This is a part of the code which moves the label surname and forename to the left. When the entry boxes are implemented, the opposite will be used and moves the entry box to the right-hand side.

The rest of the labels were implemented after. Whilst testing the implementation of the return button, when pressed the main menu would open and the booking appointment window would remain open. This was an error because I had used the wrong command associated with the return button.

After this correction was made, the program was tested to show that the program could move through the appointment window and booking appointment

Graphical user interface, text, application

Description automatically generatedA picture containing graphical user interface

Description automatically generated

The entry boxes were added after the labels, buttons and drop-down menu had been incorporated within the window. This allowed testing of the return button before the entry boxes.



Graphical user interface

Description automatically generated with medium confidenceShape, rectangle

Description automatically generated

This is the code used to create an entry box, first the frame is stated, then a width of the box and then a background colour. This element is then pack to the right-hand side of frame 2, parallel to the surname label:

The code for this was then copied and adjusted for the 2 remaining entry boxes and tested to ensure it was displayed correctly.

When implementing the time label with its entry box, it appeared to be confusing as users will not know available and valid. This led to a change in the design to incorporate radio buttons which allows valid hours that the user could pick from. These buttons allow the user to pick only valid time frames, removing errors from the window as users cannot choose un-valid times. The radio buttons will represent all the times within a day, 5 1 ½ hour slots from 9 am till 4:30 pm. The time slots will be 9:00, 10:30, 12:00, 1:30, 3:00 ending at 4:30.

The original idea for the time would have been identical to the rest of the entry boxes on the screen, but this was changed to radio buttons. The image shown was an initial test to ensure that the radio button was implemented correctly, and the process could be repeated for the other 4 available times.

Shape, rectangle

Description automatically generated

When the other radio buttons were implemented, the output showed the times to be in reverse order. This would add confusion to the users and reduce its ease of use. To fix this, the order in which the buttons were coded were reversed. This means that “3:00 PM” is packed first and “9:00 AM” is packed last.

A picture containing text

Description automatically generated

Text

Description automatically generated

Graphical user interface, application

Description automatically generated

Text

Description automatically generatedGraphical user interface, text, application

Description automatically generatedAn alternative to radio buttons would be to create a drop-down menu like the appointment type. This is a better option over radio buttons because it fits within the screen easier. Using a drop-down menu also provides clarity to the user and allows for only the correct time to be selected.

Text

Description automatically generated This is the code used for the clear button. This button will remove the data from the entry boxes and will reset the drop-down menus back to the first value. the parameters 0 and 100 are used to select the positions of what to delete. 0 to 100th position.

A picture containing text

Description automatically generatedthis is the code used to save the data to a save. Initially the command is called when the save button is pressed. It then gets and stores the data from the 3 entry boxes and stores the data under 5 variables. The csv is then opened, the variables are added together and unified under one variable and the data is separated by commas (separating the data within a csv). The new variable is then written to the csv.

Whilst testing this and entering new information, an error occurred where data would not move onto the next line within the csv. This was due to there not been a carriage return (“\n”) onto the end of the variable. This meant that data would continue to write onto one line.

This is where the variable containing the data for the information is joined onto the next one. This is fixed by adding “\n” onto the variable when its being created.



Adding the carriage return allowed the information to separate to separate lines and be viewed individually. For this version, users and their information will not have unique identifiers, but in later versions the plan is to create a unique identifier for an appointment to display the appointment times if the user wishes to view it. An identifier may also be used to link a surname and forename to an identifier. This can be used to reduce data in the appointments but can lead to errors if a new user wishes to book an appointment and they don’t have an identifier. The identifiers may also be forgotten, and a recovery method must be included to the program.

Graphical user interface

Description automatically generatedThis version also does not have validation of the information being entered into the boxes. This validation would be aimed to be included into another version and display an error message depending on which entry was not valid. The validation is important to prevent discrepancies within the data and the errors are important to add to the user friendliness of the program and allows users to fix their invalid entries.

The saving feature was tested by entering data, clicking save and reviewing the csv to ensure the data had been saved correctly in the correct positions. When implementing the save button, an initial test was to code the method to get the data and print the data within the python shell. This process was then duplicated for the rest and the combined variable was printed to the python shell. After this was working, it was then written to the csv.

### Testing

Testing for this version was consistent throughout. This included testing the entry of information and checking that this was moved to the csv correctly. The clear button was also tested to ensure information was deleted and the drop-down menus were reset. The final part of testing was the implementation of the labels and entry boxes. These were implemented one by one to check that the label and entry boxes fitted within the frames as per the design mix.

### Errors

The main error for this version was that the data was not saving correctly to the csv. This was due to the variable not having the carriage return which would allow the next part of data to move onto the next line when being viewed. This error also caused 2 variables from 2 sets of data being connected which would lead to errors when being read. This was able to be spotted during tested and fixed easily. Another error would be a few syntax errors when creating the save button, this was when the variable names for entry boxes were being incorrectly replicated when getting the information from them. This caused the program to display a red error message within python idle. This was found during the initial test of printing the data within the python idle.

### Analysis

To analyse this version, we must consider how well the program was able to develop to implement the booking appointment window and how the information is saved to the csv. The program was able to book an appointment smoothly and allows the user to enter all their information. The errors for this version were able to be found and fix easily throughout testing. This version was also able to meet its aims as well as stick to the design. The original design did not incorporate two buttons which were included into the final version, the clear button and the save button. This was because the clear button was not intended for this version but was able to be implemented alongside the save button. The save button is essential to save the data to the csv, without a save button either the data would never save, or the data would be saved per character input which would create repeat data.

## Version 1 conclusion

To conclude this version, the implementation was well done in the sense that the program was able to follow the design and the aims of the versions were met. This version did not allow the user to view their appointment. This was because I made the decision to implement appointment specific id’s, which allow them to be uniquely identified, before allowing the user to view their appointment. This sets up the aims for the next version to implement specific ids for the appointments which will be saved alongside the appointment data. The next version will also allow the user to view their appointment within the view appointment window either using their identifier or surname. As well as viewer their appointment, validation should also be implemented to the book appointment page in order to allow only valid responses to the entry boxes.

# Version 2 implementation

## Overall design

The design for the windows will continue in the same format as of the rest of the program

This version will see the introduction of errors which will be a red label with black font to make it stand out as an error

The view appointment page will have a window called “View Appointment” and will hold an entry box for users to enter their unique id and another button to open another window which will allow a user to enter their forename and surname in order to identify all appointments that the user has booked. This button is used to identify their appointment if they don’t remember their unique id

The window will have multiple buttons will yellow backgrounds and black writing, entry boxes will a grey background and then orange (#EC927F).

Overall aims for the version

This version has two aims:

Aim 1: To validate the information that the user will enter into the entry boxes to book an appointment

Aim 2: To create a view appointment window where their information about an appointment can be displayed

The validation will validate the surname box to ensure hyphens cannot be repeated, and only letters can be inputted, this is the same for forename

The validation will ensure the date has 2 numbers followed by a slash, then 2 more numbers and another slash then 4 numbers but would also allow 2 numbers e.g. 22 would be 2022 but wouldn’t allow 3 numbers e.g. 022

Interface design

“ViewApp\_Title\_Lbl”

Table, timeline

Description automatically generated

“ViewApp\_Srch\_Bttn”

“ViewApp\_Forgot\_Bttn”

“ViewApp\_UniID\_Ent”

“ViewApp\_UniID\_Lbl”

The information collected from the search will be displayed between the entry box and the forgot id button

“ViewApp\_Return\_Bttn”

Information collected by the search would then be displayed in the middle between the return button and then entry box

“ViewAppSur\_Title\_Lbl”

**Table, timeline

Description automatically generated**

“ViewAppSur \_Surname\_Ent”

“ViewAppSur \_Surname\_Lbl”

“ViewAppSur\_search\_Bttn”

“ViewAppSur \_Return\_Bttn”

Algorithm design

The algorithm design for the validation is as followed:

When the save button is clicked:

Check surname box for numbers and double hyphens

If true then error

Else pass as false (no error)

Repeat for the rest of the information and then display the subsequent errors

Write to csv is all the information has passed

When the clear button is clicked:

Clear information and error messages

The algorithm design for the view appointment page is as follows:

Create the window and its geometry

Create frames for the page

Create the title label

Create the entry box and its corresponding label

Create return button to previous page

Create search button which would search all the data read from the csv and print the corresponding data if a link is found. The search button will also validate the id to ensure it is in the right format.

The forgotten id button will open a new window which will have an entry box for the user to search up their surname and then will display all appointments within the csv under that surname.

File design

The design will include the csv within the same folder. The name will remain the same and will be read and written to throughout. This version will include error detection which won’t allow the user to use the program if the csv has not been found within the folder.

Structure design

Table

Description automatically generated with low confidence

## Version 2.1 implementation

To start with validation was implemented to the surname entry box. Initially, the validation was added and would output “error” within the python shell.

Text, letter

Description automatically generatedThis was the initial algorithm I used when making the validation for the surname. This would first check for if the entry box was empty (without spaces) and would then print accordingly in the python shell. If it wasn’t empty then it would check for a digit, and then would error accordingly. It would then check for a repeated hyphen. Since surnames can have two parts to them separated by a hyphen, the algorithm must allow for a hyphen but not one that is repeated. This line used the in command rather than looping the check using for I in range and checking each value of I.

if of the checks pass then the algorithm would print that its valid. Each check was tested and checked to ensure it worked as it was implemented.

Text, letter

Description automatically generatedThe figure here shows and updated version which would only allow characters from the alphabet and wouldn’t allow ‘.’ or ‘/’. This in theory would work but since a hyphen isn’t part of the ‘isalpha()’ method, a hyphen would cause the algorithm to fail at that if statement and wouldn’t pass as valid even though it should. This meant I would add another line which would make the entry valid if a hyphen was in there. This caused a further error would allow the user to enter a number and a hyphen which would return as valid because anything with a hyphen would return valid.

Text

Description automatically generatedThis meant that I would have to loop part of the algorithm to check each letter of the entry for either part of the alphabet or a single hyphen (a double hyphen will still error).

This part of code also checks if the entry has spaces within the entry, this will error.

The 2nd half of the function will loop for the length of the entry and will check each character for a digit and will error if it finds one and will end the loop if it does find one. It will then check if each character is part of the alphabet or is a hyphen. 1 is then added to the counter which started at 0 and will return as true when counter becomes the length of the entry and will print that it is valid.

A no entry should print no entry and an entry with a number should return ‘contains a number’:





During testing of this version, there was an error with the double hyphen. This is because whilst it would error and print the error, it would also say it is true and valid.



Text

Description automatically generatedThis occurs because the loop to check if it each character is in the alphabet or if it’s a hyphen would still run if the rest of the function fails or passes. To fix this, the program needs to stop the loop and not run the loop and just return the error. This was fixed by:

The loop is now part of an else to the if’s and elif’s that do the checks prior. If the first checks pass then the else will be run, otherwise it would.



The function that was created now had to be adjusted to allow it to be applied to the forename entry box.

Text

Description automatically generatedthis function now has two parameters, the name which is either surname or forename and then the entry which is gotten from then entry box when the function is called. A new function was created to call the validation function when the save button is clicked. This function is called “Save” and will have all 3 validation functions (surname, forename and date) and will then save the information if all 3 functions return true.

Text

Description automatically generated with medium confidence

The last part of the version was to create a form a validation for the date. This consisted of making sure the characters were in the right place and the numbers validated were within the correct ranges. The first check of the function was to check that the entry was the correct length that the algorithm expected and the positions of characters would line up correctly.

A picture containing diagram

Description automatically generatedThe second check if the is to check if the year is bigger than 2020 and if the characters were numbers. If they were bigger and they were numbers then the if statement would pass and it would move onto the next part of the function. Whilst testing this, I noticed that the program would allow the user to enter any year even if it shouldn’t be valid. This meant there was an issue with the check within the function.

This meant I had the change the check which should only allow a year above 2020. To change this, I checked each digit from 6 to 9 for each digit for the year 2021, e.g., digit 6 would be a 2, digit 7 would be a 0, digit 8 would be 2 and then digit 9 would be a 1.

This was then tested and worked as a result. When tested the year 2021 would be valid and anything else would be invalid.



Text

Description automatically generatedThe last part of the function was to set a counter which increased by one after each if statement passes. So, when the counter became 7 then ‘date\_valid’ = True and the value is returned.

The function to check time has a few flaws in its current version. Currently the validation does not account for leap years as well it does also not account for months or days that have passed. Months with 31 days or only 28 days are also not accounted for and would allow the user to input day 31 without showing that the data is invalid and would save accordingly. I have a plan to use a calendar module within python which would bring in the date from the current computer and wouldn’t allow a user to input a day, month, or year behind the current one. This is planned for another version.

This line of code was then implemented to ensure that the data is valid before it is saved otherwise invalid data would have been saved and the list would become un-usable.

### Testing

this version required me to test each possibility for the data to be invalid and see how my program responded to the entries. As shown in the screenshots throughout the description it is shown that I would enter the data and the program would error accordingly. All 3 of the data that needs to be validated was tested in the same way to ensure that all possibilities were accounted for. The surname and forename were also tested after the implementation of the date validation to ensure that there was nothing to interfere with the validation of the surname and forename

### Errors

This version saw a few errors. One error was that the program would return true if a user inputted a hyphen for either the surname or forename because the program was written that if one was inputted then it would return true regardless of any other factors e.g., a number. This was fixed to only allow the alphabet or a hyphen but not one that is repeated. The function would then check for no entry, a double hyphen, and spaces. If they passed it would then check for a digit in each position and finally if it’s in the alphabet or if it’s a hyphen. Another error is that the date would not validate the year correctly and would allow the user to input any year. This was checked to check each digit individually for 2021 and would only allow this year.

### Analysis

To review this version the implementation was good. 2 working algorithms were implemented which checks the entries and only allows valid entries. The algorithms are also written in a way which would allow me to reuse them at a later part in the program e.g., validation other surname or date entries. This version also completed Aim 1 in the sense that I have validated the information and only allowed valid inputs to be saved to the csv. The next part of the version would be to implement a way of showing the errors to the user in the same way as the rest of the program is, within the tkinter interface and not the python shell.

## Version 2.2 implementation

This version is where I will create error messages to be displayed to the user when they click save and their entries are not valid. This will be through red labels displayed within the page, following the design, and will be cleared when the user presses clear data. To start, I first created each error message as a label.

The labels were created for each of the 3 areas that were validated. They are all to be place within the same frame, adjacent to the drop-down menu for the appointment type.

Text

Description automatically generatedAll the labels have accurate variable names which correspond to their specific error, and they all have the same formatting and label colour, following the design for the error messages.

Chart, bar chart

Description automatically generatedOne of labels was tested to ensure it was in the correct position and the correct colour and size.

Once all the labels had been created, then I had to assign the correct labels to the function and the corresponding checks.

With the function being used for both the surname and forename, the program cannot print surname has no entry when it’s the forename with no entry. This meant I had to distinguish when it was the surname or the forename. So, if the name parameter was passed as surname then the ‘surname\_error’ variable would become true and if not then the ‘forename\_error” would become true. Then if an error is detected within the entry then it checks if surname\_error is true and prints a surname error and visa versa.

Text

Description automatically generated with medium confidenceText

Description automatically generated with medium confidence

When testing this the program provided the following error:

This was caused by when the forename parameter was passed the original if statement, if name == “surname” would fail and forename\_error would become true. But because the first statement failed, Surname\_error was never created and cannot be used in an if.

Text

Description automatically generated with medium confidenceTo fix this, Surname\_error had to be declared as false when the first if statement failed. Text

Description automatically generated

Text

Description automatically generatedText

Description automatically generatedWhen the forename and surname both had errors their error messages would appear side by side and would fall out of position. To fix this, a different method of positioning is used. Instead of side=tkinter.RIGHT, anchor = ‘e’ is used instead which gives the result:

A red sign with black text

Description automatically generated with medium confidence



This process was then repeated for the remained of the error messages.

The error messages were implemented to the date validation and would print them accordingly. This was then tested for each possibility and was checked if the validation would print the validation in the correct place. Examples of the date validation is shown below:

A picture containing text

Description automatically generatedThe entry 21/31/2101 would pack the labels ‘Date\_Year’ and ‘Date\_Month’ since both the year and month entered an invalid.

Shape

Description automatically generated with low confidenceThe validation worked as intended.

Another part of this version was to add a method to remove the error messages. When an error message is packed, the text in the clear button changes to ‘Click to remove error and data’. Clicking this button runs the same function before the text was changed and would unpack all the error labels and remove all the data and reset the drop-down menus.



These were packed when the save button was clicked and nothing was in the box. Then the user would input their valid data and click save

This is the result of the new valid data being saved. But these errors are still there and would confuse the user.

While testing the program after the implementation, if the user didn’t clear the error messages after getting one and decided to edit one of their entries without using the clear button, the error messages would still stay within the program, but the new data would be valid and would save.

To fix this, when the user would click on save appointment for the second time, all the error messages would be unpacked and then repacked accordingly. Instead of writing out all the labels and unpacking them all again, I wrote a function which unpacks all the labels and another function to remove the data. So, when the clear button is clicked, all the data and the errors are removed and when the save button is clicked, all existing error messages are removed.

### Testing

To test this version the validation was tested to ensure that it would pack the right messages based on the error it would have encountered. Errors were tested in conjunction with other errors to test if the program would continue to validate other areas if one area already errored. Since this version didn’t have any changes to the validation, the validation was only tested to see if it printed errors correctly and would remove them when the correct buttons were pushed.

### Errors

This version had a few errors when implementing the changes. One of the errors was when I was choosing which error message to pack when the algorithm was run. When doing this I used 2 values which I would set either one to true or the other to true. In the first iteration only one of the variables was ever declared but both were referenced. This caused the program to error. To fix this, both variables were declared and then were referenced accordingly. Another error was that the errors messages would stay packed if the users didn’t clear all their data but edited it instead. To fix this, when the save button is clicked, a function which clears all the errors messages is run and then through the other validation functions which are run, the error messages are packed again when needed.

### Analysis

To conclude this version the implementation was good as it followed the design plan and met the aims of the version. The error messages follow the format with a red label and black font to make them stand out and they are shown to the user and deleted again after. This versions error was able to be fixed. The aims for the next version would be to create a window which has the necessary buttons and input boxes and another menu which stems from that window with the necessary boxes and buttons.

## Version 2.3 implementation

This version started by created a new window which opened when the view appointment button is clicked. Following the window and creating a command which opens it, the frames for the window were created.

The title label, the entry box and the label for unique id was implemented after and when run was shown as:

A screenshot of a computer

Description automatically generated with medium confidence

Multiple spaces were then made between the entry boxes and the rest of the buttons. The spaces are place holders for where the information found by the search would be displayed. The 3 buttons, following the design were then displayed across two frames, frame 4 and 5.

Text

Description automatically generatedAfter implementing the buttons, functionality was added to the search button which would validate the entry and display an error message accordingly. The function and the displaying of the error message work in the same way that the create appointment validation works.

The code for the validation is shown below:

This will get the entry and set it to the variable ‘ID\_ent’

Text

Description automatically generated

Returns the value to the function in order to perform the search

If the entry is all digits, then it will set the value to true

Checks the entry for spaces

If the entry has no entry, then it will pack the error message that says there is no entry

If the entry has something from the alphabet, then it packs the corresponding error message

Created a variable which will return true if the entry is valid

Used to show the value is true whilst testing

Shape

Description automatically generated with low confidenceShape, rectangle

Description automatically generated with medium confidenceThis function was then tested to ensure that the validation worked for each possibility. When the user doesn’t enter anything but clicks the search button the program would display the error ‘ID has no entry’.

And when a valid input is inputted and the save button is clicked, ‘valid ID’ will be printed into the python shell and a True Boolean value would be returned.

Shape

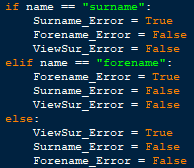
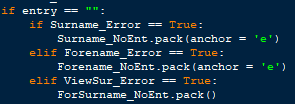
Description automatically generated with low confidence

After all the tested for the entry box was completed, functionality was added to the forgot id button which will open a new window which a user would input their surname into if they couldn’t remember their unique id.

The process for creating this window follows the same path as the other windows, creating the window and its geometry, creating its frames, creating and packing the spaces, titles and entry boxes. Once this had been implemented; functionality was first added to the return button to allow the user to move between the two new windows. The window that can be accessed through the view appointment window can only be accessed through this window, highlighted in the structure diagram for this version.

As a part of this window, a surname is entered and needs to be validated. My initial idea would be to reuse the function that is used to validate the surname when it’s entered when booking an appointment. This was tried and when tested, the error messages were packed into the book appointment window. to resolve this, I initially tried to alter the original function to pack different errors if it was dealing with the surname entry rather than the surname entry when saving an appointment.

To do this I had to create a new value which would be set to true if the ‘name’ value passed through the parameter was not either surname or forename. A new set of error messages were also created which were the same as the original surname error messages but just with different variable names. Then based on this value, which label to be packed would be decided.



When this was initially tested this produced an error. A label would appear within the window where the view appointment surname entry box would be checked. This was strange because the code which packed this label was in a command which was only run when a button was pressed, and the label would disappear if the variable name was the same as the others. To change this

The new command points to a command like the ones used before, the function clears the error messages and then runs the check algorithm.

This then fixed the issue, and the surname would be validated correctly and would create the right error message. When testing this, I went back to check if the function would validate both the surname and forename correctly when the user was to book an appointment. Then I found that the validation did not pack the error for the forename. This was because one of the parameters passed into the function was spelt incorrectly and did not fit one of the if statements and then wouldn’t set the right values to true and false.

Graphical user interface, application

Description automatically generatedGraphical user interface, application

Description automatically generatedOutput for View Appointment window: Output for View Appointment through forgotten ID:

### Testing

Testing this version firstly consisted of ensuring the formatting of the window was as expected and that the buttons which allowed the user to traverse between the windows worked correctly. The buttons which return the user to the previous menu will gain a function in a later version which will reset the window, remove the errors and data and when the user returns to that window it will appear as a brand-new window. Another part of testing was ensuring that the validation worked for the ID and the surname. A part that was especially tested was if the function used to validate the surname interfered with the other window and validation when the users would book an appointment.

### Errors

The version did include errors which mostly involved errors surrounding the validation of the surname and interfering with the other window. A large part of this error was the misspelt variable which caused the if statement to fail and error. Another part of the error was how the validation was called when the button was pressed. The way to fix this was when the button was pressed, another function was called which then calls the validation function. This also allows the function to return a value which is either true or false. And then this returned value with determine whether the program searches or doesn’t.

### Analysis

To conclude this version, it was implemented well, and the output follows the design. The window has some functionality but not to the standard that was aimed. The aim for this version was to display the appointment to the user when they search their appointment, but this version only included the creation of the two windows and the validation of their entries. But the validation of the information works well and displays the error to the user. The labels, title, buttons and entry box all follow the same colour schemes and positioning as the other window and fits the design of variable names and layout well.

## Version 2 Conclusion

The overall aims for version 2 were met to some extent. Aim 1 was met fully and aim 2 was only met to a certain extent. The validation of information when the user is booking an appointment is validated successfully and displays the user to the errors the validation may encounter. The program will only save the data if all the validation functions pass as true and displays to the user that the data has been saved. Aim 2 was partially achieved as the window was created and the entry was validated but there is no searching occurring and as of yet the database has not been read in. the aims for next version would be to implement and display the unique ids for the user and to allow the user to search their unique id or surname to find their appointment(s).

# Version 3 implementation

## Overall design

The design for the windows remains the same as the other windows, orange labels, yellow buttons, grey entry boxes and red error messages.

The output of the unique id and the appointment results will be orange labels with white text which details the nature of the information e.g., ‘Date:’ and then the information would be black as to stand out. This will remain consistent between the 2 data outputs

The outputs of the unique ID will be where the rest of the error messages are displayed on that page.

The output of the information searched by unique ID will be displayed in the centre of the screen and displayed under each other. If the appointment is searched by a surname, it is likely that that surname will have more than one appointment and therefore each appointment will be displayed on each line.

**Aims for the version**

Aim 1: To implement and display unique IDs to the user when they book an appointment

Aim 2: To allow the user to search for their appointment using their unique ID and display the information to the user

Aim 3: To display a user’s appointment(s) through the search of their surname

**Interface design**

Table

Description automatically generatedThe variable names of pre-existing Labels, entry boxes and buttons won’t change. Error messages will also no change, but the Unique ID label will be called ‘BookApp\_Unique\_ID’.

Chart

Description automatically generatedThe pre-existing labels, entry boxes and buttons wont experience variable name changes.

The data labels will have names corresponding to what they are displaying. E.g., the label displaying the date of their appointment to the user will have the variable name ‘ViewApp\_Data\_Lbl’

The surname would be ‘ViewApp\_Surname\_Lbl’

The pre-existing labels, entry boxes and buttons wont experience variable name changes.

Chart

Description automatically generatedHere the data will be shown in rows and the labels will be corresponding to the appointment. The labels will be displayed in order of date.

The labels will be names in order e.g., ‘ViewAppSur\_App1’ will be the top row and ‘ViewAppSur\_App2’ will be the second

The labels will have varying text which changes on the appointment data and will be named as variable which will be detailed when implemented.

**Algorithm design**

For finding csv

Try:

Open csv

Except:

Csv cannot be found = true

Pack error message

If csv found = true:

Write csv contents to a list

For Unique ID calculation

Take last part of list of data and add 1 to the unique ID

For printing Unique ID

Pack unique ID label

Configure Unique ID label with new value

Searching unique iD

Search entire list for the id

If found then set variable to position and read all data within that position

Else pack error message

Packing data

Configure labels with new data and pack to window

**File Design**

In this version there will be a check to make sure that the required csv is located within the user’s computer and in the correct place. If the file isn’t found, then the program won’t run correctly. In later version when multiple other csv files will be used, only parts of the program which can access the corresponding csv will be able to be used e.g. the stock levels will be functional if the stock csv can be found but maybe anything to do with appointments won’t be functional because the appointments csv cannot be found.

**Structure Design**

Chart

Description automatically generatedThis version doesn’t have any aims which would change the structure from the previous version.

## Version 3.1 implementation

The first aspect of the version was the checking of the csv and If the file can be located. If the file couldn’t be accessed, then the program won’t display any buttons but will pack an error message label. In another version this label might become more complex and give more detail e.g., the file is open and cannot be edited or the fact that it is not there.

A picture containing text

Description automatically generatedThe code is used to try and open the csv and if it can’t then a Boolean value is set to True which prevents the packing of other buttons and packs the error message.

Graphical user interface, text, application

Description automatically generatedText

Description automatically generatedIf the csv cannot be read then the value ‘csv\_notfound’ is set to true and then the error message is packed and if it isn’t true then the program continues as usual.

If the csv cannot be found, I would expect the program to pack an error message and not the buttons which would normally be displayed.

This was confirmed when tested.

Another part of checking if the csv can be read was the reading of the data and holding it within the python program. Graphical user interface, text, application

Description automatically generated

This section of code will create a data structure for the data of the csv to be entered. A loop is run for the number of lines of data within the csv and then separates the data on each line into separate variables which is then appended to the list. The list is then printed to show that the process has worked.



Text, letter

Description automatically generatedThe next part of the version was to implement unique ID’s.

This is the code used to get and create a unique ID for the next appointment. This happens after the csv has been read in and then will find the last entry into the csv, finds the unique ID of that appointments, converts the ID to a number and adds 1 to it. This is then displayed to the user when it is written to the csv.

With this implementation there was 2 errors.

One of the errors is that when writing the unique ID to the csv is didn’t allow it to be an integer so I would have to change the value to a str when its being written.

A picture containing text

Description automatically generated



Another error would be that, in the current version, only one appointment can be saved when the program is run, and would have to be re-run to read the csv and get a new unique ID.

This will also have the same effect on the searching methods. Only appointments booked prior to the program being run will be able to be searched for.

To fix this, there are 2 options. One option would be to re-read the csv and append the list on top of the current list within python. Another option would be to append the information to the current list without re-reading the csv. The second option is more efficient and therefore the method I would implement.

To write the new information to the csv I used the same method as when writing the list from the csv. This was done by appending each entry from the entry box when save was pressed. This initially produced an error within python because the append method cannot use ‘.get()’ within it and has to be defined outside of it.

A screenshot of a computer

Description automatically generated with medium confidenceText

Description automatically generated These are the variables used to establish the entry from the entry boxes.

Counter is passed from the original function of where the csv contents were written. This allows me to use the same position rather than calculating a new length of the list.

Another error was caused by using counter. When another appointment is booked without re-running the program, the data is pushed into the same space as the other data previously saved. This was because counter was not incremented and remained constant. When changing this and adding one to counter, this did not fix the issue because counter was only incremented within the function and goes back to its original value after the function finishes. To fix this, I made the counter a global variable so it can be increased outside of the function. This then fixed the issue and allowed the data to be repeated saved without restarting the program.

A final error with this would be how I created a function to calculate the new unique ID. This function would be run each time the save button is pressed. When this was implemented first, ‘()’ was missing from when the function was called and as a result would not run. This caused a knock-on effect shown below.



 This is then what is saved to the csv as the unique ID because it has not been calculated. When the program is re-run or another appointment Is booked the program will error.

Text

Description automatically generatedThis error’s because the program expects a number but is given that string.

Logo, company name

Description automatically generatedAfter this I wanted to change the Unique ID label to update. Initially the label would not update when the user would re-save an appointment. This was because the re-calculation occurred within the save function and the variable which contained the unique ID was outside of the function and did not update. To fix this I re-ran the calculation to create the unique ID within the function and configure the label to update the unique id with a new one.



The variable Unique\_Value\_Text is used here to combine a string of text and the unique ID value. To make it clearer to the user as to what the random value on their screen is, text is used to show what the value is. And since a tkinter label cannot have a string and a value as its text as it thinks it is two different attributes as they are separated by a comma. To work around this the value is converted to a string and is added to a line of text to become a single variable.

Graphical user interface

Description automatically generated with medium confidence

Whist the implementation was tested throughout, the final out of this version should output the unique ID and that the data was saved successfully if valid data is entered.

### Testing

This version included a lot of testing. This mainly included booking appointments repeatedly and testing if the information was correctly represented and was put into the correct places. The unique value was heavily tested to ensure that it would increase by 1 consistently and wouldn’t create repeat values and cause a lot of problems further on. Another part of testing was printing the list and the unique Id when they were calculated. This allowed me to spot that the data was not being put in correctly due to the counter error. Lists and values were printed into python so I could see what was happening within the program as it was being run. This was important as I could see that the unique value had be calculated correctly and would be recalculated when the save button was clicked. The list which contains all the data was important to print as I can see that all the data was taken correctly and placed within the list in the correct way.

### Errors

This version had a lot of errors. One noticeable error would be trying to add the unique value to the string that was being written, which errored because it could not write an integer and would have to be converted to a string to be written. Another error would be that if the user wanted to save another appointment after just booking one, they would not be able to book a valid appointment because the list within python had not been updated and therefore would not give an accurate unique ID and would also allow the user to rebook an appointment on top of the same time. The next error would be when trying to fix the error detailed above. This was where you could not append using ‘.get()’ and would have to create it as a variable beforehand. A fourth error would be the error associated with the counter and how this value would not update outside of the function and would go back to its original value when the function was recalled. This was fixed by making counter a global variable so it would update outside of the function and data would not be saved on top of other data. The final error was how the unique ID calculation function was not called correctly which adversely affected the csv and the list. The unique id was stored as the location of the function of the calculator which, when the program was run again or the user tried to save another appointment, would error as a new unique value was not created and errored.

### Analysis

To conclude this version there was a lot of errors but they were all able to be fixed and aim 1 was fully met within this version. The use of the calculations of unique ID’s and the adding to the list when a user books an appointment sets up the next part of the program which is to search and display the results for their appointment. The next version will display the data associated with the user’s unique ID when they enter it and press search. The display of the data will follow the design elements detailed at the start of the document.

## Version 3.2 implementation

The aim for this version is to display information to the user based on the Unique ID they have entered.

To do this I would take their entry for their unique ID, validate it and then search the list within python for a corresponding value. When this value is found then all the data will be set to different variables which will be called within different labels and then packed into the window.

Text

Description automatically generatedText

Description automatically generatedThis was the first version of the search. This was to search the entire list and would set a variable as the position of the appointment. This version produced an error where it would always pack the error message because it would fail the first if and therefore pass to the else, where the error message was packed. To fix this, a position variable was introduced which would increment by 1 every time the search failed. When this reached the length of the list – 1 then the error message would be packed.

This version will also stop the loop if the unique ID is found.

This function was then tested to ensure it would pack the error message that the appointment count not be found. To test this a unique id that had not been used was typed into the entry box and the output should be the error message.

Graphical user interface, text

Description automatically generated with medium confidenceThen this was also tested to see if it could find an appointment which the program would pack nothing, and this worked correctly.

Text

Description automatically generatedFollowing this was to add the ability to display the information to the user. To do this, Boolean values were introduced to indicate whether the appointment had been found or not. And if it was found then the following would be run:

A screenshot of a computer

Description automatically generated with low confidence

Text

Description automatically generatedAfter these variables were declared, more variables which contained different strings were to be made. These are used to combine the variables and other strings to display the information to the user accurately.

A screenshot of a computer

Description automatically generated with medium confidenceThen after these strings were created the labels were created to display the information.

Text, letter

Description automatically generatedThe final part of this function was to pack the labels and unpack the spaces within the window, specifically the spaces in frame 3.

The last part of the function was to pack the error message if the appointment couldn’t be found.

This was then tested by entering a unique ID and cross referencing the accuracy of the data displayed and the information displayed within the csv.

A picture containing text

Description automatically generatedThis proved that the program was showing appropriate data to the user when their unique ID was entered. These labels are correctly displayed and follow the designs for the window and its corresponding labels.

After this, I focussed on allowing another appointment to be searched for when the user tried to search again. This involved unpacking the labels and repacking the spaces previously unpacked.

Text

Description automatically generatedInitially I tried to unpack the labels within the existing clear function used to clear the error messages but since the labels were declared within the search function, they produced an error message saying they were undefined.

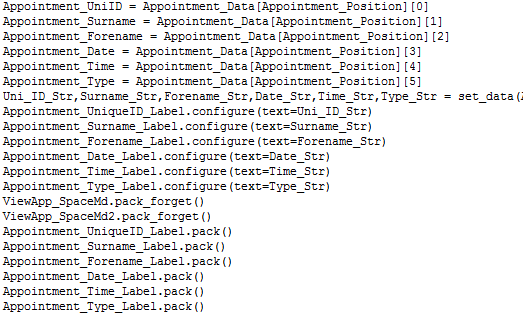
To fix this, the declaration within the function had to be moved outside of the function to be used and unpacked.

The declaration of the data variables, the strings with the variables and the labels were all moved out of the function.

To start, the data variables are declared as the first position of the list of appointments and then the function which creates all the strings is called after and returns the new variables. Using these variables, the labels are declared using those strings.

Text

Description automatically generatedThis function is then called within the function which searches and displays search results to update the labels. The Labels are then configured with the new data and then packed.



Configure the labels with the new information

New variables which contained the searched data

Function to create new strings

A picture containing timeline

Description automatically generatedChart

Description automatically generatedThis was then tested to ensure that a user can search for one appointment and then search for another.

### Testing

This version involved moderate testing in the sense that the program was tested when each label was Implemented, and the function was tested to see if it could find the appointment based on the user’s input. The testing of this was successful and I was confident in the ability of the program to search and display results to the user. Another part of the testing was to ensure that the user could book an appointment and straight away search for it without restarting the program. This worked because the list which the program operates from is updated when a user books a new appointment. The final part of testing was searching one appointment after another and ensuring that the data would update correctly and would collide with the existing labels

### Errors

The version contained a few errors which were mostly caused through variables and their declaration and the inappropriate use of if and else statements. The error associated with variables was caused by some variables trying to be altered outside of the function it was declared in. In most cases it would be better practise to declare the variable such as a label outside of the function, but I was short sighted and tried to work around this. The other error was the error associated with the if statement. This was where if the first position was not what the program was looking for then it would error regardless of if the appointment existed. This was fixed by incremented a variable instead of straight away packing the error message. The error message would be packed if the variable hit a certain value.

### Analysis

To conclude this version of development, the implementation was well done because I was able to accomplish aim 2 effectively whilst sticking to the design of the window. The implementation was also well done as there is no reoccurring errors, I was not able to fix but there were errors which did take additional time and understanding to find and fix. The program also allows for the user to search for another appointment without restarting the program which increases user-friendliness and how easy the program would be to use for the user. One area where the program didn’t follow the design is the output of the data. The design states that the output of the data will be white, which the data is, but what the date is e.g., ‘Date: ‘ was to be black to stand out. This wasn’t possible as the output per label is one string, and the text can only be one colour within one label. The aim for the next version would be to allow the user to search for appointments corresponding to their surname and display their results accordingly.

## Version 3 conclusion

Version 3 of the implementation focused on the introduction of unique ids and displaying the appointment data to the user. There were 3 aims, two of which were accomplished and the third places to pending. This was due to my scope of the program. It is unrealistic to display all appointments based on one surname as this could be potentially years/months of appointments which would overload the page and make it difficult to understand. To combat this, a date/time library can be Imported to get the date from the user’s computer and only display bookings after the current date. With the introduction of this library, allows me to better validate the booking of appointments and allow them to book any date past the current. Due to this, the aims for the next version would be to validate dates better when booking an appointment as well as displaying all current appointments under their name.

# Version 4 implementation

## Overall design

The design of the windows and labels in this version will not change and will remain consistent with the rest of the program.

The output of the data when a user searched their surname will output all appointments with a date greater than the current. It will output the appointments in order of appointment date but there may be an option further on with development to order the output by unique id instead. The output will be orange labels with white text.

The output of the data is still planned to be one appointment per line and have them repeat under each other. This is why the date is important as to not show potentially 100 appointments which have already happened.

**Aims for the version**

Aim 1: validate date entry when booking appointment using the date time library within python.

Aim 2: display data to the user based on their surname, only showing data passed the current date.

Aim 3: order the information, displayed based on a search, in order of dates.

Chart

Description automatically generated**Interface Design**

**T**his the design for the window to enter their surname. There will be more than 2 appointments shown but the design has been simplified in order to create a visual representation. There are two methods on how to present the information:

One method would be to create a separate frame for each line and have it as separate labels which each contain a piece of information. The other method would be to create one big label which contained all the information and then place the labels under each other.

I have chosen to put all the information into a single frame as this will be easier to pack all information. When packing an unknown amount of information e.g. 3 appointments or 1 appointment, it will be easier to pack them one after each other into a single frame rather than changing the frame each time when trying to display each appointment.

**Algorithm design**

Comparing dates

Get computer date and entry

Compare which is bigger

If bigger then accept

Else pack error label

Finding appointment

Validate surname

Search list for surname

Set labels with corresponding data

Pack labels

Ordering labels

Compare dates

Biggest date is the last

Smallest date is the first

Pack accordingly

**File design**

This version will continue to use the same csv; ‘Appointment ID’. The version will also keep the check to check if the csv can be round and edited.

**Structure Design**

The structure of the menu will not change from the last version as there is no introduction of a new window or ability to move between them. Below is the current structure design.

## Version 4.1 implementation

The first part the implementation was to research the use of the date and time library within python. After researching ways to use the library through articles on the internet, I was able to create a function which compares todays date and the date entered. To test this initially I created the function in a separate python file to test its functionality.

A picture containing text

Description automatically generatedFirstly, the library is imported and then the program takes an input and converts it to a string. It needs to be a string in order to split the entry into day, month and year. This is then followed by creating variables which contain the day, month or year of the input. These values are then converted to an integer form and put into the method which converts the given date into a specific format in order to compare it. Then these dates are compared and prints accordingly.

This required a large amount of testing to understand how the library works and the limitations to the use of it. The first test was to compare two dates, one being the current date on the computer and the second being the user’s input.

Text

Description automatically generatedThis is the output of the test discussed above. The date the 15th of December 2021 was inputted and compared to today’s date which was the 13th of December 2021 and printed that date 1 (13th) was smaller than date 2 (15th) which is the correct outcome.

Another test would be to test if the it would be smaller if we compared something from 2022 e.g. 5th of January 2022 compared to the 13th December 2021.

Text

Description automatically generatedThis was the output after running the test. The process of comparing them was the same as before and outputted that the date entered was bigger than the current. This was the expected outcome.

A third test was to compare a date before the current and review the output of the program. The date 10th December 2021 was entered and compared to 13/12/2021 and printed the following:

Text

Description automatically generatedThis was the correct output. And this output will be used to tell the program that the date inputted in invalid.

The final test was to check if the function would prevent entries which didn’t exist e.g. 29th of February. This was tested by inputted the 29th of February 2022 and ensuring that it would error. But this was tested again to accept 29th of February 2024 which does exist.

Text

Description automatically generatedThe two photos below show this test and how the program reacted. The outputs were correct and expected.

Text

Description automatically generated with low confidence

This function will be used to compare the date of the entry and that of the computers date and will only allow dates bigger than the current. This comparing of dates will also be used again to display the information of a user’s surname, in order to display the appointment in order of dates.

Text

Description automatically generatedThe next step was to implement this function into the main program and adjusting the function to get the information from the entry. The end of the function would make the date\_valid variable true and would return as true and save the data.

With this implementation gave the program errors as this collided with the existing validation. The main error being the validation of the day, and whether it should accept up to 30, 31 or 28. There is 2 methods to fixing this issue. One method would be to find every possible max day per month per year or allow the function newly implemented to validate the date. There are issues with each method. The problem with using the first method is that this is very length, time consuming and will take larger amounts of space and processing time. Whereas issues with the second method are that the error messages won’t be as specific as before and may lead to slight confusion with the user. The ultimate method would be to print a calendar in which a user can click a valid day on any month and any year but I am uneducated on how to do this. The method I have chosen is to allow the function to validate the entry and deal with vague error messages although some error messages such as length or wrong order can still be detailed.

One error message which can stay the same would be the if statements to check the length of the date entered and print a message if the length isn’t correct. Another series of checks which can stay would be checking for integers in the correct place and checking ‘/’ are in the correct places.

Text, letter

Description automatically generatedThe if statement to run the date comparing the current date had to be adjusted to ensure it wouldn’t error and would run when all the rest of the data had been validated. A try and except loop was used. This tries to put the data into a date form and if it can’t because the date doesn’t exist e.g. 29th of February then it will error.

I would then test the error messages to ensure the error messages showed when they should because the entry wasn’t valid. The first test I did was to check that the entry wouldn’t allow letters in any position:

Text

Description automatically generatedthis was the expected output and this happened consistently regardless if the letter or invalid character was in the month, day or year.

Text

Description automatically generated

Another test was to ensure that the new function didn’t allow invalid dates e.g. 40th:

The page was then tested by an external user to remove bias and test its real work application. This allowed me to see and understand where confusion could be found and allows me to spot more bugs when someone else is testing and they don’t know what they are testing for.

### Errors

The biggest error this version would be the collision between the function implemented and the existing validation function that had been made. The new function allowed for the user to input any valid date as the function from the library would only allow valid dates. This meant that I did not have to think of every possible date that could be entered and attempt to validate each possible combination of month, day and year. Another error was when the function was called but the try and except part had gone down the except part meaning that d2 hasn’t been declared. This caused an error because d2 was referenced after the try and except but wasn’t declared unless the try path was successfully taken. To fix this the comparison was moved to within the try path and would only happen if the entry was a valid date.

### Testing

The testing for the version largely consisted of testing each possibility of what the function would accept and what it wouldn’t. through testing it was derived that the function would only allow valid dates, which is useful as this was an ongoing part of my validation to include leap years and months with 30 or 31 days. Another part of the testing was to check that error messages were correctly displayed when they should have been. The 2nd part of testing was done by an external user which allowed for unbiased testing and an example of real-world use without someone who knows what they are looking for testing it. External testing is useful as I can see other errors that I wasn’t testing for myself but errors which shouldn’t exist and can be fixed.

### Analysis

To conclude this version, the implementation was good and well done. With effective testing and understand of the date and time function developed; I was able to meet aim 1 of the version and effectively validate a date entry. This is also important ground work for the part of the program where data is to be ordered in order of date. The aim for the next version would be to display all future appointments based on someone’s surname. Another part of this problem is if the user has a lot of future appointments booked and would exceed the given space within the window. A way to fix this would be to display all appointments within 3 or 6 months of the current date. This is another application of the date and time library and something to consider within the scope of the program.

## Version 4.2 implementation

To start this version, the first part was to find all appointments linked to a user’s surname and set up the variables associated with them.

This was done by creating a function which looks through the list of data containing the appointments and appending the position of each appointment which contains the entered surname. These positions are then taken as variables and all the information is taken from each position. String variables are then declared based on the information taken from the data list.

Text

Description automatically generatedThis is the function that finds the position of the appointments. First it gets the entry (providing its valid) and then searched the entire list of the appointments for the surname entered and if it finds it, it will append the position to a list. If there are no matching appointments then ‘appointment\_found’ = false and would pack an error message.

This is tested by printing the list after running a surname that had several appointments booked. When ‘Fowler’ is search, the list is searched and the following positions are appended to the list:

This is all correct and cross referenced with the csv file except one appointment. This appointment would be appointment ID 1033

The reason this did not show up because this is booked under the surname ‘fowler’ and not ‘Fowler’ which are deemed different within python (because of the value of the capital). To fix this, when a user’s information is entered it needed to be converted to lower case letters consistently. This means that when they search ‘Fowler’ all appointments under ‘fowler’ will show up because they will all be lower case and the entry of a surname will also be converted to lower case. So when ‘Fowler’ is entered, it will convert to ‘fowler’ and that will be searched.

Once this was accomplished, the next task was to compare the dates of the appointments saved and compare that to the current date and remove any appointments from the lists with dates that have already occurred. To do this, the current date was found and then compared to that of the entry date. This is done using the same method of version 4.1.

Text

Description automatically generated

This was the first iteration of the comparison, which would take each date for each corresponding appointment and would then print this to python. To test this, I typed in ‘page’ which has a total of 6 appointments. The program should identify all 6 and print their positions and their dates.

Text, letter

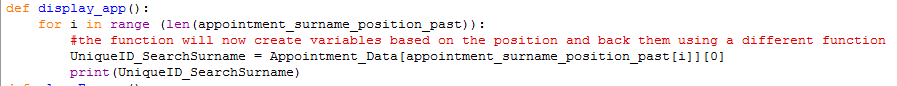
Description automatically generatedThe test worked as expected and produced the correct results. The results were checked with the csv and the corresponding positions.

Text, letter

Description automatically generated

A picture containing text

Description automatically generatedAbove is the code used to compare the two dates that are in question. Firstly it takes specific characters from the date in the position in the appointment data list. Then it changes it to an integer and finally into a date formate. This is then compared with the current date and if the date for the appointment is after today’s date, then the position of this appointment will be saved and then displayed to the user.

The next part was to take the positions saved and extract the information about the appointment from those positions. Below is the start of a function which will set each variable up based on each position on the list.The function was tested after creating it to see if would correctly identify and set the right values. When I typed in page to search a surname I expected it to find one valid appointment which would be appointment iD 1039 which is an appointment on the 12th of January.

Text, letter

Description automatically generatedText

Description automatically generated with medium confidenceThis test shows the function worked as intended and then the function was expanded to obtain the rest of the variables from the appointment information.

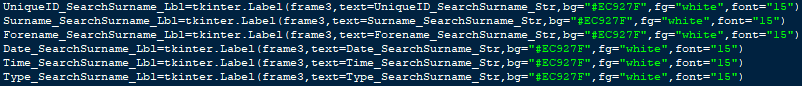




A screenshot of a computer

Description automatically generated with medium confidenceThese variables are then translated into a string under a new variable which will be the text for each label which contains the information of an appointment. These variables are then returned.

Following the create of the variables and the labels, a separate function was created which when called would pack all of the labels. When this was tested an error was produced due to when the labels were created, the string it tried to use as text was not yet declared. This was a similar error to an error in version 3.2 where strings had to be declared to create the labels and the strings were altered when needed. A similar fix can be implemented here by creating the strings before the labels.

Label creation:

Text

Description automatically generatedError message:

To fix this, I took the labels and the string creation outside of any functions so they were declared outside of the function. Each label now needed to be configured to update the text within them to display the new information that the program has just found.

The next step was to configure each label with the new information that had been obtained. This is done by using the .configure method and then changing the text to the new string variables which contain data corresponding to the appointment. This was then tested by inputting ‘page’ and expecting it to display information corresponding to this appointment here:

Text

Description automatically generated with low confidenceThis shows that the function worked to an extent by displaying correct and date-valid information to the user. This originally didn’t follow the design however as the data is printed underneath each other instead of being on one line for each appointment.

To change this, instead of creating separate labels for each bit of information, all the information was merged into one string which was then put into one label. This was the result:

Graphical user interface, text, application

Description automatically generated with medium confidence

This method has an issue with the sizing, initially it was too big so the writing has to be scaled down to fit within the screen without the user having to take any action.

The next part of this was to test how the program dealt with two or more appointments. Initially the loop was called as many times as there was corresponding appointments and the idea was for the loop to pack another label whilst keeping the original, but this didn’t work because they are the same label being configured. So, when this was run, it would display appointment 39 but then would be configured again to display appointment 40 and removing appointment 39.

A way to fix this would be to pre-determine how many appointments to show, e.g. the next 3 (max) appointments. Using this, I could create 3 labels which would get packed depending on how many values there was within the list of valid appointments and pack each of the labels.

Firstly, I created all 3 of the new labels with different variable names and then introduced a pointer the function which is called when the button to search is pressed. The idea of the pointer is to differentiate which label needs configuring and packing. Then a condition is introduced:

Text, letter

Description automatically generatedThe first If is to check if the list is 3 or less and if it is, do the loop for this length, and if it isn’t 3 or less, then do the loop 3 times. This is so that the program doesn’t error when trying to pack 4 sets of appointments when the limit is only 3.

Text, letter

Description automatically generatedThe packing function was also altered to incorporate this change.

This change was to use the pointer to differentiate which label needed to be packed. The pointer would change between 1,2 and 3 and would pack a different label depending on this value. This was then tested when only 3 appointments were valid and booked under the surname ‘page’. The result was:

Text

Description automatically generatedGraphical user interface, text

Description automatically generated

the print in python was to show that there was only 3 valid appointments and they were all found correctly.

The final test was to test how the program dealt with more than 3 appointments and what the program printed.

Text

Description automatically generatedThe output in the tkinter window was the same as when there were 3 appointments (as expected) but the output in python included the new appointment and showed that there were 4 valid appointments.

The next step would be to compare all the date-valid appointments and order them in date order and then display them in date order.

### Errors

This version had a lot of errors and problems with the program. This is largely due to how the data was found and displayed to the user for example the labels and displaying the correct amount. Another large part of this version and its error was the searching for a surname and the capitals and lowercase and how that affected the search. This was an error because within python ‘fowler’ doesn’t equal ‘Fowler’ numerically because F and f have different numerical values. This meant that I had to make a few changes to the csv, specifically making all the names lower case, and implementing the .lower() method to the .get() methods in order to obtain an all-lower-case version of the entry that the user was trying to search for. Another large error was how I was trying to pack the same label 3 times but to appear as a different label, instead when I ran the program it would update the existing label and would only show 1 out of the 3 appointments. To fix this, I created 3 separate labels with unique names which would be packed depending on how many appointments there was to show to the user. As well, to simplify the problem I capped the appointment display to 3 to make it easier to implement whilst providing good functionality.

### Testing

Version 4.2 had a large amount of testing, to ensure dates were compared properly as well as if the appointments could be identified properly. This required a lot of printing into the python shell in order for me to cross reference with the csv to ensure that the data was being picked out as intended. The labels and displaying the information required the most testing as the functions being used were tested whilst they were being built up in order to spot and fix potential errors as well as maintaining the scope of the program and being flexible in the design. This method allowed me to change how the labels were put together, by combining all of the strings rather than displaying them as separate labels and duplicating these into 18 different labels rather than just 3. This change also allowed me to better follow the design of the window and the method of displaying it to the user.

### Analysis

To conclude 4.2, I would say that aim 2 has been fully achieved. The program will display 3 date-valid appointments to the user based off of a surname and effectively compares the appointment date to today’s date in order to remove past appointments. Whilst this version did have a large number of errors and testing, the program has been fixed and is now working as intended. The next step for this version would be to work on aim 3, ordering the appointments in a closest to furthest date order. This will require me to extract the dates of all the valid appointments and compare the distance to the current. Following this, I will re-order the appointments in the list and display the appointments using the same method as version 4.2.

## Version 4.3 implementation

The start of this version would be to create a new list which takes all of the dates out of the validate appointments in order to compare the distance between today’s date and the appointment date.

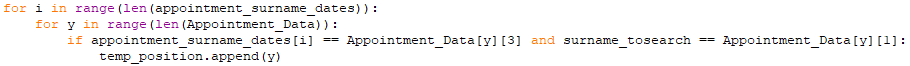
Text

Description automatically generated This was done by appending the positions of valid appointments to a list. The next step was to work out how to compare dates. To do this I researched the date-time library further and discovered you can minus dates from each other and it will give you a day value as the difference. I initially tested this in a separate file:

After testing this method, I thought I might be able to use the ‘sorted’ function in python in order to order the dates. To test this, I did:

This shows the sorted list on the top and the un-ordered list on the bottom. This shows that this method worked and it was no longer necessary for me to minus the dates from each other and work it out that way.

The original method for the next step of the ordering would be to find the dates in the list of appointments again and record the positions, although this would work if there was only one appointment per day, there is a max of 5 and therefore multiple different people on the same day and therefore cannot just identify the appointment by using the date of their appointment. This meant that I had to check that the surname and the date matched and then record the position from there.

this snapshot of code contains two loops and an if statement. The two loops are to initially order all elements of the ‘appointment\_surname\_dates’ and the second loop is to check all elements within the list, which contains all the information about all the appointments, for the corresponding date and the surname. If both these are found corresponding to the same appointment then the position of the appointment is appended to a temporary list. This was initially tested by printing various lists and positions into python.

The first list to be printed was all valid appointments which corresponded to that surname:



The second list to be printed was all the dates which corresponded to those appointments from the list of appointments:



The 3rd part of the test prints was to print the list just printed but in an order version to verify that the dates had been properly sorted:



And the last part of the testing was to see the new order list of positions which should be in date order:



These prints show to me that the program is currently working as intended.

The last part of the test was to ensure that the sorted method would work for dates in February, which would be further in time than January but the day would be lower than the 13th (the current closest appointment to the time of writing). To do this an appointment was saved in February and then search for an appointment by a surname.

Graphical user interface

Description automatically generated with medium confidence This photo showed that the sorted method did not correctly sort the list of dates and placed the 1st of February at the start of the list when it should have been at the end of the list. From this test, I cannot use the sorted method effectively for comparing across multiple months and must revert back to the previous method in order to order the dates. Whilst I can no longer use the sorted method, the part of the function which compares the date and surname can still be used.

To attempt to create a new function, I created a small sub version of the program which contains a few dates and I would attempt to order them in an external environment from the main program. This allows me to focus on the problem at hand by extracting the problem from the program in order to decompose the problem.

Text

Description automatically generatedAbove is the entire sub version. This was the method I used to fix the issue with ordering the dates. Firstly, I will show the program working as intended and then will explain the code.

This shows that this program was able to effectively re-order the dates into an accending order.

The first part of the program is where the distances from todays date and the date within the list and recording the distance. In addition the date is appended to a list within a the correct format in order to be used by the library. This is important in order to reuse the values again to indentify which dates gave the same distance as recorded. This helps reduce the amount of repeat code within my program. The sorted method is then used to order the list of distances into an accending order. Then a loop is initied which loops for as many dates there are, 2 new values are then introduced which are order and pointer. The order value Is used to start and end a while loop which will end when the distance between a date in the list and todays date is verified with a distance in the list of ordered distances.

The first line is to calculate the distance again, and is then compared with position i within the list of ordered distance. If this isnt found then pointer is increased and then a new distance is created and compared to the same position i. when they are found to be equal, the date corresponding to the pointer position will be appended to a new list which contains the ordered list of dates. The lists are then printing in order to assure that the sort worked correctly.

The next step of the development was to implement the new method into the main program.

Following the intergration and attempting to run the program, this was the error that was presented.

Text

Description automatically generated

This error was due to d1 being in a different format to the main program. The fix to this was to change the variable name to the current date equivelent (‘date\_current’). When run this worked correctly and would successfully alter the list to reflect an accending date order.

Text

Description automatically generated with medium confidence

Old position order of appointments

Old order of dates

New order of dates

New position order of appointments

The last part of the integration was to change the variables that was being used to fetch the appointment information to the new variable which contained the ordered list of appointments.

A screenshot of a computer

Description automatically generated with medium confidenceA screenshot of a computer

Description automatically generated with medium confidence

Graphical user interface, text

Description automatically generatedGraphical user interface, text

Description automatically generatedThe photos above show the test to show that the function now works and correctly displays to the user their appointments. Within the photo, it shows two different people which have the same surname. This was considered, and a fix would be to include a first name search in addition to a surname in order to make the search more specific to the user. Another issue would potentially be how the appointments won’t be show in time order, so if the appointments were on the same day, the order may be wrong due to the dates, but the chance of having two appointments under the same day with the same surname is small but would be a factor to consider.

A point to consider for this, is the potential security risks and concerns associated with a real-world use of this program. A username and password could be used to make the process more secure, instead of making everyone’s appointment available to anyone that can search that surname.

### Testing

Version 4.3 had a large amount of testing to ensure that the dates would be sorted correctly when the function was run. The list of positions was printed to check before and after of the sorting to ensure that the positions were re-ordered to an ascending order of dates. A lot of testing was carried out in and out of the main program. One of the main important parts of testing was testing the sorted method which found out to be dysfunctional for what I needed it for. More through testing should have been carried out before discovering that the sorted function didn’t work and building a large amount of the version of the basis, although this was recoverable. Another part of testing was testing how the dates misused from each other and found the distance between the two dates.

### Errors

Whilst developing this version, there was a large amount of errors. A large part of these errors came from inadequate testing of the sorted method, as well as creating a new function which would order the dates. The sorted function would not correctly sort dates outside of the same month, so when February and January appointments where compared, February would appear before January whereas this wasn’t what was desired for the function. There were a few variable associated issues when implementing the new function which was created in an external environment. This came from changing of variable names and inconsistencies between the transfer, notably was the names of lists and the position of them.

### Analysis

To conclude this version, the implementation was okay and achieved the main purpose of the version, aim 3 which was to order the dates of displayed appointments. There were many errors throughout the version, but they were overcome through research, testing and recreating the problem in an isolated environment. The testing in this version was able to spot most errors and spot that the sorted method wouldn’t work in the way that was needed for the project. I was able to follow the design by keeping each appointment to one line and keeping them all within the same frame.

## Version 4 conclusion

Version 4 was an important version in order to display users their appointments without their unique ID for their appointment. This version also allowed for better validation of the date entry when booking an appointment, using the date and time library within python. Both these developments allowed me to achieve all 3 of my aims, whilst sticking to the initial design and following closely with the feedback and request of the customer. Although in the current state you can still book appointments on the same time and date as someone else. To fix this will be aim 1 of version 5. The next area of development would be to allow the user to edit their appointment, changing the time, date, type or cancelling the appointment. This will expand into a new window where the user will be able to select a field using a drop-down menu and then enter the information, they wish to replace this field with. This function will need a unique ID in order to identify their appointment.

# Version 5 implementation

## Overall Design

The design for the edit appointment window will have the same colour scheme as the other windows. This window will also have drop down menus and radio buttons to help the user edit their appointment.

The window will have a title, followed by a label and an entry box to enter their unique ID. After the appointment is found, a drop-down menu will appear for the user to choose what they want to change about their appointment. The type, time and date are the 3 options a user can change, as well as to cancel the appointment.

There will be error messages and confirmation messages to appear on the window to assist the user and make it easier to understand.

There will also be a pop up, in a separate window which will ask for confirmation before cancelling your appointment.

**Aims for the version**

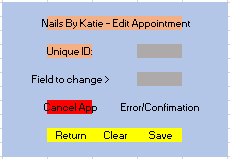
Aim 1: Ensure that appointments cannot be booked on top of another appointment. Validate the time and date and compare it to the list and return the search.

Aim 2: Create a window which has an entry box for an ID and a drop-down menu which the user can change and a corresponding button- or drop-down menu is shown.

Aim 3: Change data of an appointment, ensuring that data cannot be saved on top of another appointment. Ensure that the csv updates correctly.

Aim 4: Allow the user to cancel their appointment

**Interface Design**



EditApp\_Title\_Lbl

EditApp\_UniID\_Ent

Subject to change between drop down menus and an entry box

EditApp\_Error/Confirmation\_Lbl

EditApp\_Save\_Bttn

EditApp\_Clear\_Bttn

EditApp\_UniID\_Lbl

EditApp\_DropDown

EditApp\_Cancel\_Bttn

EditApp\_Return\_Bttn



ConfirmCancel\_Lbl

Cancel\_N\_Bttn

Cancel\_Y\_Bttn

**Algorithm Design**

Blocking same date and time entry:

Take the date entry from the entry box

Compare the date with all dates existing within the database, recording positions where the dates match (if any)

If no dates are found to be the same, the appointment is saved on the date entry

If dates are found, the corresponding appointment’s time of appointment would be compared to see if they were on the same time

If true, appointment is not saved, error message packed to user to say date and time filled

If false, appointment save procedure called, appointment saved to database, green label displayed to user to show it has been saved.

Creating edit appointment window:

Create a window name, geometry and frame.

Create a function which is called when the button ‘edit appointment’ is clicked in order to open the window to edit the appointment

The title label is created, followed by a space, then an entry box and a label which the user will input their unique ID into

Then a list which contain the different fields to be used for the drop-down menu

The drop-down menu is created into the next frame

Opposite the drop-down menu will be either another drop-down menu or an entry box depending on the entry to the first drop-down menu

A red button will be under the first drop-down menu and will take the user to a separate simple window which will ask the user to confirm yes or no to cancel the appointment

Opposite the red button is the space to display the error or confirmation that the change has gone through, and different labels will be packed depending on the status of the change.

Below this, there will be 3 buttons:

Return to menu window will take down the edit appointment window and open the appointment window

Clear date button will reset all the entry boxes and drop-down menus

Save button will attempt to save the date entered to the corresponding to the entered Unique ID. Part of this function will also re-write the csv with the new edited data.

Confirmation window:

Create window, geometry and 3 frames

Create function which calls the window of the confirmation

Display to the user a line of text asking for confirmation and two buttons below it

Clicking the button yes will take you back to editing appointment window with a confirmation label saying appointment deleted

Clicking the button, no will take you back to editing appointment window to carry on like normal

To save the edited data:

When save button clicked, validation information and display error messages corresponding to the errors

Can use previous date validation function to validation date entry

Stop function to save if error occurs

Edit Appointment data list with the information from the edits

Save this new data list on top of the current csv data

Display confirmation message to user if the appointment updates successfully

**File design**

The file will contain a csv which holds the information for each appointment and a python file to run the program. Whilst the data will be edited in this version, there will be no changed to the file contents.

**A picture containing chart

Description automatically generatedStructure Design**

This version largely develops the edit appointment window which will branch into a confirmation window if necessary.

## Version 5.1 implementation

The start of this version was to compare if the entered date was the same as any other dates within the list. If there was a repeat date, this position would be appended to a list and repeat would be set to true

If repeat is true, the time of the new appointment is compared to the appointments with the same date, and if the time is also the same, taken is set to true and an error label is packed. The function then returns both values and then the program continue.

Text

Description automatically generatedThe program will check if taken is true and will only save the data/appointment If taken is false. Taken will be false if repeat is false so the program doesn’t need to check repeat.

This was then tested by trying to book an appointment on the same time as another booked appointment:

Graphical user interface

Description automatically generated with low confidence

This is the appointment that stops the save from going through.



Graphical user interface, text

Description automatically generatedChanging the time of the appointment made it that the time and date wanted was no longer contested meaning an appointment could be saved on to that time, without contradicting another appointment.

Another test would be to try and book an appointment on a day that hasn’t been book on at all, meaning the variable repeat should remain false and so should ‘taken’.

Graphical user interface, text

Description automatically generatedThis entry contains a date which doesn’t have a single appointment booked on It, meaning both variable repeat and taken should remain false, which they do, tested by printing them both to python shell after the check\_date\_repeat function is run:



### Errors

This version’s errors were limited to syntax errors when implementing the returns and calling the function. The function was able to identify dates that matched the entry and was able to compare appointment times in order to identify if the appointment

### Testing

The testing for this version was carried out by attempting to book an appointment in an existing date and time and expecting the program to error and say that the space is taken (shown on page 4). A second test was the book an appointment on the same day as another appointment but at a different time, expecting the program to save the appointment as normal. The third test was to attempt to book an appointment on a day which has nothing on it at the time of booking. This test should keep ‘taken’ false as well as ‘repeat’ which means the appointment will save.

### Analysis

To conclude this version, the implementation was well done due to a low level of errors as well as achieving aim 1 of the version. This version means that the user cannot book an appointment onto another appointment time, which is an essential factor of a booking system. This helps keep the database reliable as well as give the program a better real-world application. The aim for version 5.2 would be to create a window following the design and algorithm plan detailed above. Whilst this window won’t have editing or cancelling functionality, the window will have all the features of the finished version.

## Version 5.2 implementation

Text, letter

Description automatically generatedThe first part of the window was to create the window, geometry and colours.

Following the window creation, 4 frames were created all with ‘EditApp’ prefixes for each frame. This is a change from the other windows as they have the same frame names. Changing the names makes it easier to reuse other parts of the program by reducing the confusion and likely errors due to crossovers.

Text

Description automatically generatedFollowing the frames, a function was created in order to open the window upon clicking the ‘Edit Your Appointment’ button.

This function was then added as a command to the button.

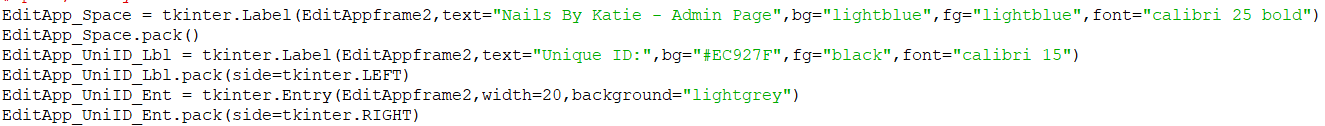
A picture containing graphical user interface

Description automatically generatedThis was then tested by clicking the button and ensuring that the window would open:

Shape, rectangle

Description automatically generated

Following this test, a space and a title was introduced to the window. Once this was run to ensure the label was passed in properly, the unique ID label and entry box were introduced

 Graphical user interface

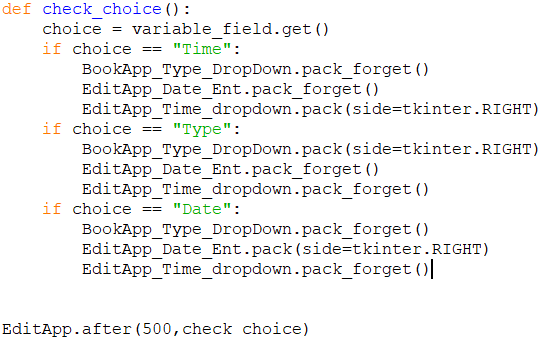
Description automatically generated with medium confidence

The next step was to introduce a drop-down menu which would determine the field that the user would want to change about their appointment.

The idea is that the value selected in the drop-down menu will constantly be looked at, so that the entry box or drop-down menu beside it will be reflecting the right values.

The first option was to use a while loop to constantly check the entry box, but when placed in a while loop the rest of the program doesn’t run and the tkinter window/interface doesn’t open.

Then the second option was to use the tkinter method ‘after’. The use of this method was found through researching the internet and reading into tkinter and its methods. This method takes two arguments and will run a function after a given time delay.



Time delay in milliseconds

This if the function called by the ‘.after’ method. This function checks the entry and packs either a drop-down menu or an entry box depending on the answer.

This method was tested and run but the function was only called once. This was proved by adding a print command to the function and seeing that it only printed once. After researching this, the problem is because the function within the .after is only called once and .after needs to be used again within the function in order to repeatedly check.

Text

Description automatically generated

Shape, rectangle

Description automatically generatedShape, rectangle

Description automatically generatedShape, rectangle

Description automatically generatedThese are the 3 options available to the user. The entry box or drop-down menu changes depending on what the user has selected. This test shows that the function and .after method works correctly and as expected.

After this iteration, the rest of the window was implemented.

A picture containing shape

Description automatically generatedA picture containing text

Description automatically generatedThis is the space and each individual code for each label for the 3 buttons at the bottom of the menu. Output:

The last part of the implementation was to create the external window to ask for confirmation when a user wants to cancel their appointment. This window will pop up when the red button is clicked and will close when either button is clicked. When the button is clicked, the appointment ID will be validated and verified to ensure that the appointment exists. If the appointment id cannot be found, then the window will not open, and an error message will be packed.

The window was set up followed by two frames. A function that opens the cancel appointment and closes the edit app window was created and is called when the cancel appointment button is run. At the end of development, another function will be created which would also validate the entry and call the window.

Graphical user interface, text, application

Description automatically generatedLabels and 2 buttons were introduced to the window, following the design elements closely. The output is shown below when the window is opened.

Graphical user interface, text, application

Description automatically generatedTo make the window more user friendly and clearer to use for the user, the buttons were expanded, and colour coded to represent yes and no. The title and window was the expanded in addition.

Text

Description automatically generatedThis is the entire section of code which creates the cancel application window.

### Errors

Version 5.2 had 1 major error and a few minute errors. The biggest error was with the .after function. The first implementation of the method didn’t work as the function that should have been called was only called once. This was then fixed through researching the method and understanding that the .after method needs to be within the function that the original .after method called. Another error was when setting the width for the buttons in the cancel app window, the width was originally set to ‘50px’ but this errored as the function expected an integer rather than a string. This was fixed by changing ‘50px’ to ‘50’.

### Testing

To test version 5.2, I ensured that the user would be able to traverse between the main menu and the edit appointment page. The user can go between both windows using the two buttons on each window, both using a function to traverse the windows. Another test of traversal was opening the cancel appointment window. Whilst the user cannot close this window, the user is able to open the window and in the next version will either cancel their appointment or keep their appointment and return the edit appointment window. Another aspect of testing was moving between the 3 option the user must pick from to edit their appointment. This involved changing the three options and ensuring that the correct drop down or entry box was displayed.

### Analysis

Concluding version 5.2, aim 2 was implemented into the program. This meant creating a window which will allow the user to enter a unique ID and edit a field from their appointment or cancel their appointment. Whilst the editing part has no functionality the window does allow for traversal, the changing of fields and a separate window which asks for confirmation to cancel their appointment. The next version will allow the user to cancel their appointment or edit the fields of their appointment. Version 5.3 will also validate the entry of the data and/or the unique information using the functions already implemented to validate other fields e.g. view appointment using the unique ID

## Version 5.3 implementation

Version 5.3 started by implementing validation to the unique ID and the field they wish to change. Another part of this was only validating the date field both time and type fields are drop down menus and don’t need validating. If the data field was validating even though that wasn’t the field to be changed, the program would display an error saying that there is no entry in the entry box.

To implement the validation, I used a function previously used to validate the appointment search using a unique ID. To use this function, I must create a value that would change based on off what it was validating either searching or editing appointment. This is Important as the function needs to pack different errors based on what it is validating.

A picture containing table

Description automatically generatedThe first implementation was to create a value, called ‘Type’, which is either given ‘Edit’ or ‘Search’ based on whether which ID entry is being validated.



The value type is passed into the ‘Validate\_ID’ function.

A screenshot of a computer

Description automatically generated with low confidenceThe second part of the implementation was to check the value and follow the function down different paths depends on the value.

The screenshot on the left was the validation before the implementation. The screenshot on the right includes the if and elif statement which checks the value to see if it is search or edit. The validation process is the same following both paths but each path packs different sets of labels. A new set of labels were declared which display the same error message but are packed inside of frame ‘EditAppframe4’ instead of ‘ViewAppframe3’.

Graphical user interface

Description automatically generated with medium confidenceA picture containing chart

Description automatically generatedDuring the implementation, a test was carried out by clicking the edit appointment button, which would get and validate the entry to the Unique ID field and display an error. The intended output would be that the error message is placed to the right of the cancel appointment button. The initial output was packed in the middle of the screen. This was an error associated with the creation of the error messages which were being packed into ‘EditAppframe3’ instead of ‘EditAppframe4’. This was then changed and retested, with the correct expected outcome (shown in the photo on the right.

Graphical user interface

Description automatically generatedOther tests were carried out in order to check the other parts of the validation. One test was to check that it picked up on spaces within the entry and should pack ‘ID contains a space’ in the same position as the previous error message.

Typing in the entry ‘2 2’ which contains a space in between the two numbers. The function will see the space and will pack the correct error message associated with the error.

Timeline

Description automatically generatedThe 3rd test was to ensure that the function packs an error message corresponding to an invalid character within the entry. The entry ‘2e’ contains an invalid character which is ‘e’ which fails under the .alpha method and fails the validation and packs an error message as shown to the left.

Text

Description automatically generatedThe 2nd part of implementing validation to the window was validating the date entry. The first part would need to look at which field on the drop-down menu was chosen by the user and would only attempt to validate the entry box if the date field was selected. The variable ‘choice’ has already been declared within the function used to check which field is being changed. This variable is used within an if statement and prints the value of choice.

This if statement was then tested by running the function and changing each field to change. The first test was to run the function when date was selected. The expected output would be for the program to identify that the field is ‘date’, print the value of choice and that the validation would go ahead if it was implemented.



Shape, rectangle

Description automatically generated

The screenshots about show the output of the program which shows the intended response.

The next test was to change date to either type or time and review the output from the program. The expected output would be for the program to print “choice != date” and the validate (if implemented) would not run and cause the program to crash.

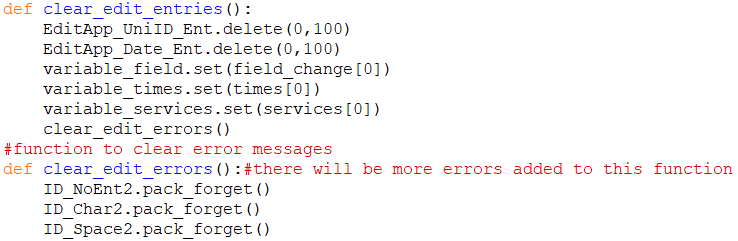
Shape, rectangle

Description automatically generatedThe screenshot to the left shows the output of the program which matches the expected output of the function.

The 3rd part of version 5.3’s implementation was to add functionality to the clear entry button as well as remove the errors when the save button is clicked.

Two functions were created. The first function clears each entry box and resets each drop-down menu, whether it’s being displayed or not, back to its position 0. The other function is used to un-pack the errors within the window. The 2nd function is called within the first function, as well as the save button

It is important that these two functions are separate so that the function to remove the errors can be used in two different functions without having to list all the labels which would be un-packed more than once.



The ‘clear\_edit\_entries’ function removed all the entry box contents and resets the drop-down menus and the ‘clear\_edit\_errors’ function removed the error messages.

The next step was to test both functions by entering some data and changing the drop-down menus.

A picture containing graphical user interface

Description automatically generatedA picture containing graphical user interface

Description automatically generatedThe expected result of clicking the clear entries button was that the entry box for unique ID be cleared, the field to edit drop down be reset and the drop-down associated with the type field to be reset as well. The screenshot below the test shows the intended outcome.

Graphical user interface

Description automatically generated with medium confidence

Graphical user interface

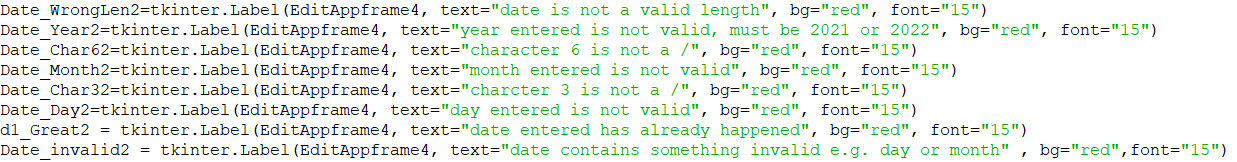
Description automatically generated with medium confidenceA second test to be carried out was to input an invalid unique ID, attempt to edit the appointment (this would run the validate function) and pack an error message. The test would then be to click the clear entries button, which should remove the error message as well as the entry to the unique ID field.

The two screenshots illustrate the test carried out. The output is the correct and expected output.

Another step during the implementation was to validate the date entry if that is the field to edit. This can be done using the previous validation used to validate the date entry when booking an appointment. This validation also includes validating dates aren’t in the past.

With the re-use of this function, slight iterations need to be made, like how other functions were re-used where they need to pack error messaged in different parts of the programs. The relies on a variable being passed through which determines where the function is being used and which path of error messages need to be packed. Before this is carried out, new error messages need to be declared with different positions.

8 new labels were introduced which displays specific error messages to the user depending on which part of the validation function fails. These new labels are differentiated by new variable names and a different frame which they are packed into.



New frame. Placed in the same position as the error messages for the unique ID

New variable name in order to pack these labels specifically.

The next step was to change the function in order to pack the new labels depending on what is being validated.

Text

Description automatically generatedThe screenshots to the left show the addition of 2 if statements when the function finds an error and needs to pack a corresponding error message.

Text, letter

Description automatically generatedThe variable Type is passed into the function as either Save or Edit which suggests where the function is being used. This then packs either Date\_Month error message or the Date\_Month2 error message which are in different places.





Once each condition had been altered, the function was tested to ensure it would validate and display error messages as intended. The first test to check that the function picked up on the correct or incorrect length of the entry. This check is to stop the whole function because the entry is already invalid

Text

Description automatically generatedThis is the code to check the length. The intended outcome would be for the wrong length label to be packed adjacent to the cancel appointment window.

Diagram

Description automatically generated with medium confidenceThe screen shot shows that the function picked up on an incorrect length and packed the correct label in the correct position.

Graphical user interface, text

Description automatically generated with medium confidenceThe next test would be to test the year, where the function should not pass a year unless it is ‘2022’. Typing in a year such as ‘2021’ or ‘2023’ should fail the validation and should pack the error message saying that the year is invalid.

The screenshot above shows the output of the previously described test. The output, whilst the correct output as intended, is in the wrong format and doesn’t fit into the window. To fix this, where the labels are packed, instead of side = tkinter.RIGHT, anchor = ‘e’ is used instead to fix the issue.

Graphical user interface, text

Description automatically generated

The screenshot to the right shows the same test as before but now the labels are placed under each other which sits better on the screen. Error messages are also shorted as to not push the screen or other buttons out of place.

The next part to implement was aim 4, cancelling an appointment

To accomplish this aim there are 2 methods I had come up with.

Method 1: this is where the appointment would be deleted from the csv and the list within python. This is a good method as it removes redundant data and frees up space within the csv. A potential error with this method is the free space within the csv. The program would not fill this gap within the csv with another save and the next time the database is searched, the program will error as it will expect an entry, but it will be empty.

Method 2: another method would be to add a true or false value to each appointment to suggest whether the appointment is cancelled or still valid. Whilst this does avoid the issues of gaps within the database and reduce the numbers of errors, this method does have other issues. One issue being it keeps data that is pointless within the database, using storage space which isn’t strictly necessary. Another issue is when it comes to booking appointments on top of other appointments. Method 2 would need the program to check if the date that a user wants to book on top of is only taken by valid appointments rather than appointments that have been cancelled.

The final part of version 5.3’s implementation would be to edit the appointment that’s been selected and update this appointment within python and the csv.

Text

Description automatically generatedThe first step of this is to find the appointment that the user wants to change by searching for the appointment using the inputted unique ID. This can be done by using the same function as when the user would search for their appointment using their unique ID.

The code above is the code used to call the “find\_app” function and located the appointment based off the unique ID. Appointment Found is a Boolean value which is either set true or false depending on whether the appointment can be found or not. If the appointment cannot be found the error message label is packed.

This iteration of the function is tested in two ways, by typing a valid Unique ID and expecting it to find the corresponding appointment and outputting true. The other test is to type a valid ID which doesn’t have a corresponding appointment and expecting the program to output false and pack the error message.







The test shows that the program found the corresponding appointment to the unique ID 1020, as shows as a screenshot of the appointment information. The program then outputted true, showing that the appointment was found.

The second test would be to type in a validate ID which didn’t corresponding to any valid appointment. This should output an error message and ‘false’ to python idle



A picture containing shape

Description automatically generated

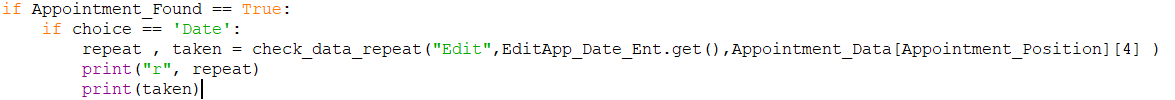


The screenshots here suggests that the test did work and showed that the program did not find a corresponding appointment to ID 1050, which is true. The program also outputted the correct error message in the correct position, making it clear to the user that their appointment couldn’t be found.

The final part of the implementation was the get the information that the user had entered, validate it, ensuring it does not collide with other appointments and their time periods, adhering to aim 3.

The first part of this was to get the information from the entry box and validate it accordingly.

If the user had entered a date, this date would need to be checked against all other appointments to see if any appointments exist on this day, if they do then the two times are checked against each other. This can be done by using the same function that I have previously created to stop appointments being booked on top of each other in version 5.1.

To implement this function, the function was called if the appointment ID had been found with another appointment.

This is the time corresponding to the appointment trying to be changed

This gets the date entry from the entry box; this is then used to reference against all other appointments

This is the variable ‘Type’ which is used to determine which error message to be packed.

This function will take the date and see if the date matches any other dates of appointments that have been booked. If there is a repeat, ‘repeat’ is set to True, and another process is started. The other process, that is dependent on ‘repeat’ being True, is comparing the time of the appointment to all other times of other appointments on that day, and a repeat sets ‘taken’ to True which means that there is another appointment on that date and time and cannot be booked. If ‘taken’ is False, then the edit will go ahead. ‘repeat’ being False will also allow the appointment to be booked.

There are 3 ways to test this function. Test 1 would be to input a valid unique ID and input a date which there are no appointments on top of it, therefore ‘repeat’ would stay False, and the edit would (if functionality was present) go ahead. Test 2 would be to input a valid ID and input a date which has another appointment on top of, but the times differ, and therefore the edit should take place. This test will keep ‘taken’ False and ‘repeat’ will be True. Test 3 will be to input an ID and a date which has the same date and time as another appointment. This test should make both variables False, pack an error message and the edit wouldn’t be carried out.

Test 1:

Shape

Description automatically generated with medium confidence

When test 1 was carried out, appointment 1050 was attempted to be changed from 05/03/2022 to 07/03/2022. This date (07) has no appointments booked on it, meaning ‘repeat’ stayed False and no error message was packed. This was the expected outcome and shows the function working successfully.

Shape

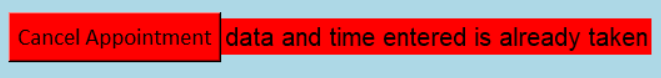
Description automatically generated with low confidenceTest 2:



After carrying out test 2, where appointment 1050 was attempted to be changed from 05/03/2022 to 06/03/2022 which has another appointment (1051 as shown above) on that date (06). This means that ‘repeat’ would be set to True, but since 10:30 AM and 9:00 AM are different times, ‘taken’ remains False and the edit would go through. This was the expected outcome of the test and shows the program successfully picking out repeats and checking if the times match.

A picture containing shape

Description automatically generatedTest 3:



Test 3 shows an appointment being identified as a duplicate where appointment 1049 was attempted to be changed from 05/03 to 06/03 but since these appointments share a time, the appointments would be booked on top of each other which wouldn’t be allowed. This process sets both ‘taken’ and ‘repeat’ to True meaning that the edit cannot go ahead, and the user needs to change the date or time first. This was the intended output and shows the program successfully identifies that the appointment data would be the same If the edit went ahead and prevented this.

Text

Description automatically generatedOne of the last steps for the version 5.3 implementation would be to do a similar task as shown above, but for times. This involved identifying if the appointment to be edited would also interfere with another appointment and their time slot.

Used to show position of function when being run

This searches for the appointment based on the entry and packs an error if it can’t be found.

This is the part of the function which would validate a time against other times of other appointments. An important point of this is that this only occurs if the drop-down menu to select a field is set to time. The first step is setting two Boolean values to False. These values have the same purpose of the previous function, repeat checks if there are any appointments on the same date, and taken checks if there is an appointment on that date and time.

A list is then created which is used to store all the positions of appointments which have matching dates, dates are then compared for the length of the list of appointments and positions of matching dates are appended to the list. ‘Repeat‘ becomes true if there are any matches.

The final for i in range and if statements compare each appointments time with matching dates to see if their times match, and if they do then ‘taken’ becomes True and the save would not go ahead. An error message is also packed to show the user that this position has been taken and is unavailable.

There are 3 ways to test this function, like how the date was function was tested. The first test would be to input a time for an appointment which has no other appointments on that date. They should keep ‘repeat’ and ‘taken’ False, meaning the save would be valid. The second test would be to input a time for an appointment which has more than 1 other appointment on that day. This test should set ‘repeat’ to True but keep ‘taken’ False which allows the same to go ahead. The final test would be to input a time on a day which already has an appointment on that date and time. Both ‘taken’ and ‘repeat’ should be set to True, an error message would be backed and the save would not go ahead.

Shape

Description automatically generated with medium confidence



When test 1 was carried out, a valid unique ID was input, and the time 12:00 pm was entered to change the 9:00 time to 12:00. The expected outcome was for both values to be False as there are no other appointments booked on the same day. But the actual result of the test showed that the program saw the appointment in question as an appointment on this day. Whilst this did not prevent the save from going through, it was not the expected outcome. It is important to note that if the used tried to book on top of their time, the program will see this as already taken and won’t allow them to save. This will pack the taken error message rather than explicitly telling the user this is their own time

Shape

Description automatically generated with medium confidence



This was test 2. Where an appointment which had another appointment (1050) was booked on this day. The test spotted this, turning ‘repeat’ to True but ‘repeat’ stayed False because 12:00 is not the same as 10:30 am and has allowed the save to go ahead. I know that this is a different result to test 1 as the program has picked out another appointment rather than itself.

The third test of the function was to input a time already taken on the date of the appointment.

Graphical user interface, text

Description automatically generated

This test shows the program picking out that the time of appointment 1050 and the new time of appointment 1049 was the same, turning the value ‘repeat’ to True which would prevent the save, and packs an error message to the user. This was the intended outcome and shows that the program will prevent duplicate times.

The final part of the implementation was to change the appointment information corresponding to their unique Id and re-write this back to the csv.

To do this, I used the variable ‘Appointment\_Position’ which is the index position of the appointment to be edited, and set index position 3 to the new date, providing it was valid and the appointment did not collide with other appointments.

Text

Description automatically generatedThe third line of code here is the line of code which changes the data. The other print lines are to show a before and after the change to ensure that it has taken effect. This was tested after the implementation.

Text

Description automatically generated with low confidenceShape

Description automatically generated with medium confidence

This test shows that the change didn’t take place. The expected output is that the date would change from 05/03 to 06/03 but this didn’t happen. This was because there was a double equals used when changing the data.



This change of code allowed the program to change the data in the list. The previous test was then carried out again.

Shape

Description automatically generated with medium confidenceA picture containing text

Description automatically generated

This test shows the program matches the expected output and shows the change of information.

Text

Description automatically generatedThis process was then replicated for the time and type of appointment.

Shape

Description automatically generated with medium confidence

Text

Description automatically generated with low confidence

A similar test to the date test was carried out when changing the time of an appointment. This involved entering a new time which wasn’t taken and changing the time of the appointment within the list. The expected output would be the change of time from 10:30 to 12:00 which as demonstrated is the output of the program.

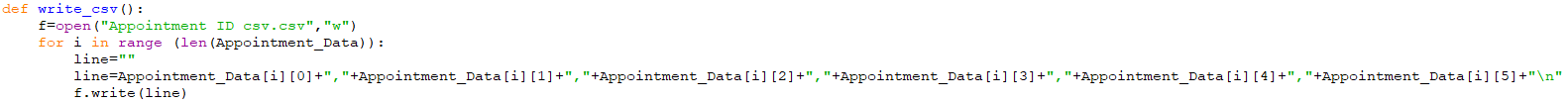
These two tests shown above wouldn’t take place unless the variable ‘taken’ was false, meaning there wasn’t an appointment contradicting the date and/or time.

The 3rd change, type of appointment, doesn’t require any validation as the type of appointment is picked from a drop-down menu and has no implications on any other appointments and their time slots.

Shape

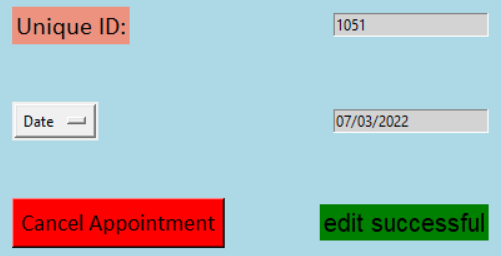
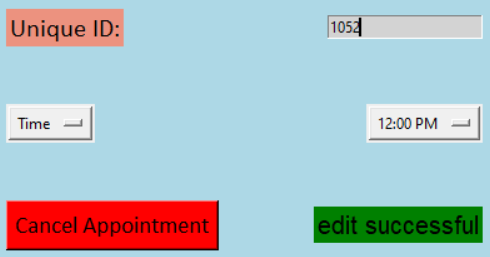
Description automatically generated with medium confidence

This was the final test to show that the program could change the appointment type to whatever the user had selected from the drop-down menu.

The next step in addition to changing the list within python, was writing it back to the csv with the added changes.

This is a function I created which opens a csv and writes as many appointments as there is back to the csv. At the end of this function, the csv is closed. The variable line is composed of all information corresponding to each appointment, separated with commas. This function is called each time an edit is made as is it not specific to what field is being changed. This function was tested after its implementation by attempting to change appointment 1051 from 06/03/2022 to 07/03/2022

The expected outcome of this test would be that the appointment is changed within the csv and the ‘edit successful’ label is packed. The test to the left shows this test and shows the expected outcome. This shows how the appointment 1051 had the date 06/03 but has been edited to 07/03/2022 which was the intended.



This process was then duplicated for the 2 other fields and tested. The test was to attempt to change the time of appointment 1052 to 12:00 from 1:30pm. The expected outcome would be for the appointment to have changed within the csv as no other appointment should have that time. The other expectation was for a green label to be packed suggesting that the appointment change has gone ahead.

The final test was to alter the type of appointment. This test doesn’t validate the appointment against others as a change of type will not interfere with other appointments. This means that the edit will have 100% of the time as it cannot fail. The test involved changing appointment 1052 from a manicure to a pedicure.

Graphical user interface, text, application

Description automatically generated The 3 screenshots to the left show the program changing the csv to reflect the changes that the user has made to their appointment.



### Errors

The errors with this version were largely associated with the display of labels and the output of the program. This is shown by labels being placed the in the wrong frames (page 11) or labels being placed side by side on page 15. The first error was a result of the wrong frame being used to place the labels into, when the frame was changed to the correct location, the label was packed properly. The second error was caused by using tkinter.RIGHT rather than anchor. The change of this makes the labels be placed on top of each other rather than side by side. A third error experienced with this implementation would be that the program would error if the csv was open was a change was attempted. This was fixed by adding a try and except loop which would try to open the csv in a writing format and if it couldn’t, it would pack the error message to say it was open. Other errors in this version were those associated with variables not being set correctly or spelt wrong when they were called.

### Testing

Version 5.3 had vast amounts of testing each scenario of possibilities in order to correct test each part of the implementation. This involved testing each check of the validation function e.g. checking for spaces, numbers, invalid characters etc. all possibilities were check through to ensure that the correct error message would get packed if that error did come up. Another part of test was testing each part of the cross-referencing function which checks for duplicate dates. These functions are based off three possibilities, duplicate date and time, duplicate date but unique time, unique date and therefore unique time. These 3 possibilities where testes for both time and date edits to ensure that appointments could not be booked on the same dates and times. Final parts of testing were ensure that the change that the user wanted is transferred to the csv and is updated within the program. This involved viewing the csv before the change, making the change and then viewing the csv again and verifying that the change had taken place.

### Analysis

To conclude version 5.3, the implementation was successful to a certain extent because aim 3 was 100% achieved whereas aim 4 was not. Aim 4 was not successfully implemented because there were 2 methods as to how to implement this each with drawbacks and advantages which were discussed, but both methods weren’t viable for the program. The user can now edit their appointment and view this updated appointment using the program. The user’s changes are validated against other appointment to ensure that the database does not allow duplicate appointments, adhering to aim 3. This version had a small number of errors because a large part of the implementation was validation which had been done previously throughout the program. This allowed me to re-use functions following a few changes to make it applicable. Re-using these functions removed a lot of errors as the methods I have previously used were tested and didn’t yield any errors.

## Version 5 conclusion

Version 5’s implementation was an important milestone for the program, completing the development of the user accessed side of the program. This version accomplished 3/4 aims and now allows the user to edit their appointment and removes the ability for users to edit or book appointments on top of other appointments. The implementation followed the initial interface and algorithm design and allowed me to develop the program following the client’s guidelines. The next versions focus will shift to the admin page and being to work on a stock levels window. This window will allow an admin to view stock levels of different products by selecting a product or a range of products which will then display the current levels to the user.

# Full code print out:

#library imports

**import** **tkinter**

**import** **datetime**

#checking for csv

csv\_notfound = False

csv\_found = False

**try**:

f=open("Appointment ID.csv","r")

csv\_found = True

**except**:

csv\_notfound = True

**if** csv\_found == True:

f=open("Appointment ID.csv","r")

Appointment\_Data=[]

counter = **0**

**for** line **in** f:

line = line.strip('**\n**')

Appointment\_ID,Appointment\_Surname,Appointment\_Forename,Appointment\_Date,Appointment\_Time,Appointment\_Treatment = line.split(",")

Appointment\_Data.append([])

Appointment\_Data[counter].append(Appointment\_ID)

Appointment\_Data[counter].append(Appointment\_Surname)

Appointment\_Data[counter].append(Appointment\_Forename)

Appointment\_Data[counter].append(Appointment\_Date)

Appointment\_Data[counter].append(Appointment\_Time)

Appointment\_Data[counter].append(Appointment\_Treatment)

counter = counter + **1**

f.close()

**def** **Unique\_val\_calc**():

#finds the most recently used entry

length = len(Appointment\_Data)

#increases unique value by 1 to create a new unique ID

Unique\_Value = int(Appointment\_Data[length-**1**][**0**]) + **1**

Unique\_Value\_Text = "Unique Value is = " + str(Unique\_Value)

**return** Unique\_Value, Unique\_Value\_Text

Unique\_Value,Unique\_Value\_Text = Unique\_val\_calc()

#functions to withdraw and update windows (traversing windows)

**def** **MainMenu\_Window**():

AdminPage.withdraw()

AppWindow.withdraw()

MainMenu.deiconify()

MainMenu.update()

**def** **AdminMenu\_Window**():

MainMenu.withdraw()

AdminPage.deiconify()

AdminPage.update()

**def** **AppMenu\_Window**():

BookApp.withdraw()

MainMenu.withdraw()

ViewApp.withdraw()

EditApp.withdraw()

AppWindow.deiconify()

AppWindow.update()

**def** **BookApp\_Window**():

AppWindow.withdraw()

BookApp.deiconify()

BookApp.update()

**def** **ViewApp\_Window**():

AppWindow.withdraw()

ViewAppSur.withdraw()

ViewApp.deiconify()

ViewApp.update()

**def** **ViewApp\_Window\_Surname**():

ViewApp.withdraw()

ViewAppSur.deiconify()

ViewAppSur.update()

**def** **EditApp\_Window**():

AppWindow.withdraw()

CancelApp.withdraw()

EditApp.deiconify()

EditApp.update()

**def** **CancelApp\_Window**():

EditApp.withdraw()

CancelApp.deiconify()

CancelApp.update()

**def** **Quit**():

quit()

#creating main menu within python tkinter

MainMenu = tkinter.Tk()

MainMenu.title("Main Menu")

MainMenu.geometry("500x400")

MainMenu.configure(bg="lightblue")

#MainMenu.attributes('-fullscreen', True)

#creating the frames for the menu

frame1=tkinter.Frame(MainMenu)

frame1.configure(bg="lightblue")

frame1.pack()

frame2=tkinter.Frame(MainMenu)

frame2.configure(bg="lightblue")

frame2.pack()

frame3=tkinter.Frame(MainMenu)

frame3.configure(bg="lightblue")

frame3.pack()

frame4=tkinter.Frame(MainMenu)

frame4.configure(bg="lightblue")

frame4.pack()

#creating title label within main menu

MainMenu\_Space = tkinter.Label(frame1, text="Nails By Katie" ,bg = "lightblue",fg="lightblue", font="calibri 20 bold")

MainMenu\_Space.pack()#seperation from top of window

MainMenu\_Title\_Lbl = tkinter.Label(frame1, text="Nails By Katie", bg="#EC927F",fg="black",font="calibri 25")

MainMenu\_Title\_Lbl.pack()#title label

MainMenu\_Space = tkinter.Label(frame1, text="Nails By Katie" ,bg = "lightblue",fg="lightblue", font="calibri 35 bold")

MainMenu\_Space.pack()#larger space between title and 1st button

#creating error label for if the csv is not found

Csv\_Appointment\_NotFound = tkinter.Label(frame2,text="Appointment Csv not found and wont be functional",bg="red",fg="black",font="calibri 17")

**if** csv\_notfound == True:

Csv\_Appointment\_NotFound.pack()

**else**:

MainMenu\_Admin\_bttn=tkinter.Button(frame2, text="Admin Log On", bg = "yellow",fg="black",font="calibri 15", command=AdminMenu\_Window)

MainMenu\_Admin\_bttn.pack()#first button for admin page

MainMenu\_Space = tkinter.Label(frame2, text="Nails By Katie" ,bg = "lightblue",fg="lightblue", font="calibri 20 bold")

MainMenu\_Space.pack()#smaller space between both buttons

MainMenu\_App\_bttn=tkinter.Button(frame3, text="Book An Appointment",bg="yellow",fg="black",font="calibri 15", command=AppMenu\_Window)

MainMenu\_App\_bttn.pack()#2nd button for appointment page

MainMenu\_Space = tkinter.Label(frame4, text="Nails By Katie" ,bg = "lightblue",fg="lightblue", font="calibri 20 bold")

MainMenu\_Space.pack()#space between appointment menu and quit button

MainMenu\_Quit\_bttn=tkinter.Button(frame4, text="Quit",bg="yellow",fg="black",font="calibri 15", command=Quit)

MainMenu\_Quit\_bttn.pack()#quit button

#creating appointment window and its geometry

AppWindow=tkinter.Toplevel(MainMenu)

AppWindow.title("Appointment Window")

AppWindow.geometry("500x405")

AppWindow.configure(bg="lightblue")

AppWindow.withdraw()

#AppWindow.attributes('-fullscreen', True)

#creating frames within the appointment window

frame1=tkinter.Frame(AppWindow)

frame1.configure(bg="lightblue")

frame1.pack()

frame2=tkinter.Frame(AppWindow)

frame2.configure(bg="lightblue")

frame2.pack()

frame3=tkinter.Frame(AppWindow)

frame3.configure(bg="lightblue")

frame3.pack()

frame4=tkinter.Frame(AppWindow)

frame4.configure(bg="lightblue")

frame4.pack()

frame5=tkinter.Frame(AppWindow)

frame5.configure(bg="lightblue")

frame5.pack()

#creating labels and buttons for the Appointment page

AppPage\_Space = tkinter.Label(frame1,text="Nails By Katie - Appointment Page",bg="lightblue",fg="lightblue",font="calibri 20 bold")

AppPage\_Space.pack()#create space between the top of the page page and the title

AppMenu\_Title\_lbl=tkinter.Label(frame1, text="Nails By Katie - Appointment Page", bg="#EC927F",fg="black",font="calibri 25")

AppMenu\_Title\_lbl.pack()#title label

AppPage\_Space = tkinter.Label(frame1,text="Nails By Katie - Appointment Page",bg="lightblue",fg="lightblue",font="calibri 20 bold")

AppPage\_Space.pack()#create space between the title and the 1st button

AppMenu\_book\_bttn = tkinter.Button(frame2, text="Book your Appointment", bg="yellow",fg="black",font="calibri 15", command=BookApp\_Window)

AppMenu\_book\_bttn.pack()#creating the button which will take the user to another page to book an appointment

AppPage\_Space = tkinter.Label(frame2,text="Nails By Katie - Appointment Page",bg="lightblue",fg="lightblue",font="calibri 20 bold")

AppPage\_Space.pack()#creating the space between the book appointment button and the view appointment button

AppMenu\_view\_bttn=tkinter.Button(frame3, text="View your Appointment",bg="yellow",fg="black",font="calibri 15", command=ViewApp\_Window)

AppMenu\_view\_bttn.pack()#creating the button which would take the user to a page to view their appointment specifically

AppPage\_Space = tkinter.Label(frame3,text="Nails By Katie - Appointment Page",bg="lightblue",fg="lightblue",font="calibri 20 bold")

AppPage\_Space.pack()#space between view appointment button and ammend appointment button

AppMenu\_edit\_bttn=tkinter.Button(frame4,text="Edit your Appointment",bg="yellow",fg="black",font="calibri 15",command = EditApp\_Window)

AppMenu\_edit\_bttn.pack()#button to allow users to ammend an appointment

AppPage\_Space = tkinter.Label(frame5,text="Nails By Katie - Appointment Page",bg="lightblue",fg="lightblue",font="calibri 20 bold")

AppPage\_Space.pack()#space between ammend appointment button and return to main menu button

AppMenu\_return\_bttn = tkinter.Button(frame5,text="Return to Main Menu",bg="yellow",fg="black",font="calibri 15",command=MainMenu\_Window)

AppMenu\_return\_bttn.pack()#button to re-open the main menu from the appointment page

#creating admin window and its corresponding geometry

AdminPage=tkinter.Toplevel(MainMenu)

AdminPage.title("Admin Page")

AdminPage.geometry("500x405")

AdminPage.configure(bg="lightblue")

AdminPage.withdraw()

#AdminPage.attributes('-fullscreen', True)

#creating frames for the the admin page

frame1=tkinter.Frame(AdminPage)

frame1.configure(bg="lightblue")

frame1.pack()

frame2=tkinter.Frame(AdminPage)

frame2.configure(bg="lightblue")

frame2.pack()

frame3=tkinter.Frame(AdminPage)

frame3.configure(bg="lightblue")

frame3.pack()

frame4=tkinter.Frame(AdminPage)

frame4.configure(bg="lightblue")

frame4.pack()

frame5=tkinter.Frame(AdminPage)

frame5.configure(bg="lightblue")

frame5.pack()

#creating the labels and buttons within the admin menu

AdminPage\_Space = tkinter.Label(frame1,text="Nails By Katie - Admin Page",bg="lightblue",fg="lightblue",font="calibri 20 bold")

AdminPage\_Space.pack()#create space between the top of the page page and the title

AdminPage\_Title\_lbl=tkinter.Label(frame1, text="Nails By Katie - Admin Page", bg="#EC927F",fg="black",font="calibri 25")

AdminPage\_Title\_lbl.pack()#title label

AdminPage\_Space = tkinter.Label(frame1,text="Nails By Katie - Admin Page",bg="lightblue",fg="lightblue",font="calibri 20 bold")

AdminPage\_Space.pack()#create space between the title and the 1st button

Admin\_app\_bttn = tkinter.Button(frame2, text="View Appointments", bg="yellow",fg="black",font="calibri 15")

Admin\_app\_bttn.pack()#creating the button which will take the user to another page to view appointments of various clients

AdminPage\_Space = tkinter.Label(frame2,text="Nails By Katie - Admin Page",bg="lightblue",fg="lightblue",font="calibri 20 bold")

AdminPage\_Space.pack()#creating the space between the view appointments button and the view stock levels button

Admin\_stock\_bttn=tkinter.Button(frame3, text="View Stock Levels",bg="yellow",fg="black",font="calibri 15")

Admin\_stock\_bttn.pack()#creating the button which would take the user to a page to view stock levels of products

AdminPage\_Space = tkinter.Label(frame3,text="Nails By Katie - Admin Page",bg="lightblue",fg="lightblue",font="calibri 20 bold")

AdminPage\_Space.pack()#space between stock levels button and view analytics of the business button

Admin\_analytics\_bttn=tkinter.Button(frame4,text="View Analytics",bg="yellow",fg="black",font="calibri 15")

Admin\_analytics\_bttn.pack()

#button to allow the user to view various analytics in the form of graphs using matplotlib libary

#also can show analytics using labels

AdminPage\_Space = tkinter.Label(frame5,text="Nails By Katie - Admin Page",bg="lightblue",fg="lightblue",font="calibri 20 bold")

AdminPage\_Space.pack()#space between view analytics button and return to main menu button

Admin\_return\_bttn = tkinter.Button(frame5,text="Return to Main Menu",bg="yellow",fg="black",font="calibri 15",command=MainMenu\_Window)

Admin\_return\_bttn.pack()#button to re-open the main menu from the appointment page

#creating book appointment window

BookApp=tkinter.Toplevel(AppWindow)

BookApp.title("Book an Appointment")

BookApp.geometry("630x650")

BookApp.configure(bg="lightblue")

BookApp.withdraw()

#BookApp.attributes('-fullscreen', True)

#creating frames for the book appointment page

frame1=tkinter.Frame(BookApp)

frame1.configure(bg="lightblue")

frame1.pack()

frame2=tkinter.Frame(BookApp)

frame2.configure(bg="lightblue")

frame2.pack()

frame3=tkinter.Frame(BookApp)

frame3.configure(bg="lightblue")

frame3.pack()

frame4=tkinter.Frame(BookApp)

frame4.configure(bg="lightblue")

frame4.pack()

frame5=tkinter.Frame(BookApp)

frame5.configure(bg="lightblue")

frame5.pack()

frame6=tkinter.Frame(BookApp)

frame6.configure(bg="lightblue")

frame6.pack()

frame7=tkinter.Frame(BookApp)

frame7.configure(bg="lightblue")

frame7.pack()

#creating space from top of the window to the title label

BookApp\_Space = tkinter.Label(frame1,text="Nails By Katie - Book an Appointment",bg="lightblue",fg="lightblue",font="calibri 20 bold")

BookApp\_Space.pack()

BookApp\_Title\_Lbl=tkinter.Label(frame1, text="Nails By Katie - Book an Appointment", bg="#EC927F",fg="black",font="calibri 25")

BookApp\_Title\_Lbl.pack()#creating title label for booking appointment page

BookApp\_Space = tkinter.Label(frame2,text="Nails By Katie - Admin Page",bg="lightblue",fg="lightblue",font="calibri 25 bold")

BookApp\_Space.pack()#space between title and a surname label and entry box

BookApp\_surname\_Lbl=tkinter.Label(frame2,text="Surname:",bg="#EC927F",fg="black",font="calibri 15")

BookApp\_surname\_Lbl.pack(side=tkinter.LEFT)#surname label

BookApp\_Space = tkinter.Label(frame3,text="Nails By Katie - Admin Page",bg="lightblue",fg="lightblue",font="calibri 25 bold")

BookApp\_Space.pack()#space between surname and forename

BookApp\_forename\_Lbl=tkinter.Label(frame3,text="Forename:",bg="#EC927F",fg="black",font="calibri 15")

BookApp\_forename\_Lbl.pack(side=tkinter.LEFT)#forename label

BookApp\_Space = tkinter.Label(frame4,text="Nails By Katie - Admin Page",bg="lightblue",fg="lightblue",font="calibri 25 bold")

BookApp\_Space.pack()#space between forename and date

BookApp\_Data\_Lbl=tkinter.Label(frame4,text="Date (dd/mm/yyyy):",bg="#EC927F",fg="black",font="calibri 15")

BookApp\_Data\_Lbl.pack(side=tkinter.LEFT)#date label

BookApp\_Space = tkinter.Label(frame5,text="Nails By Katie - Admin Page",bg="lightblue",fg="lightblue",font="calibri 25 bold")

BookApp\_Space.pack()#space between date and time

BookApp\_Time\_Lbl=tkinter.Label(frame5,text="Time:",bg="#EC927F",fg="black", font="calibri 15")

BookApp\_Time\_Lbl.pack(side=tkinter.LEFT)#time label

#drop down menu for times on the book appointment pag

times = ["09:00 AM","10:30 AM","12:00 PM","01:30 PM","03:00 PM"]

variable\_times=tkinter.StringVar()

variable\_times.set(times[**0**])

BookApp\_Time\_dropdown=tkinter.OptionMenu(

frame5,

variable\_times,

\*times)

BookApp\_Time\_dropdown.pack(side=tkinter.RIGHT)

BookApp\_Space = tkinter.Label(frame6,text="Nails By Katie - Admin Page",bg="lightblue",fg="lightblue",font="calibri 25 bold")

BookApp\_Space.pack()#space between time and drop down

services = ['Pedicure','Manicure','Gel Polish','Acrylics','Nail Art']#list to hold different available services

variable\_services=tkinter.StringVar()

variable\_services.set(services[**0**])

BookApp\_Type\_DropDown=tkinter.OptionMenu(

frame6,

variable\_services,

\*services)#drop down menu

BookApp\_Type\_DropDown.pack(side=tkinter.LEFT)

BookApp\_Space = tkinter.Label(frame7,text="Nails By Katie - Admin Page",bg="lightblue",fg="lightblue",font="calibri 25 bold")

BookApp\_Space.pack()#space between the drop down menu and the return to main menu button

BookApp\_Return\_Bttn=tkinter.Button(frame7, text="Return to Main Menu", bg="yellow",fg="black",font="calibri 15", command=AppMenu\_Window)

BookApp\_Return\_Bttn.pack(side=tkinter.LEFT)

#creating error messages for surname

Surname\_NoEnt=tkinter.Label(frame6, text="surname has no entry", bg="red", font="15")

Surname\_Double=tkinter.Label(frame6, text="surname contains repeated hyphen", bg="red", font="15")

Surname\_Space=tkinter.Label(frame6, text="surname contains a space(s)", bg="red", font="15")

Surname\_Digit=tkinter.Label(frame6, text="surname contains a digit", bg="red", font="15")

Surname\_Char=tkinter.Label(frame6, text="surname contains an invalid character", bg="red", font="15")

#creating error messages for forename

Forename\_NoEnt=tkinter.Label(frame6, text="forename has no entry", bg="red", font="15")

Forename\_Double=tkinter.Label(frame6, text="forename contains repeated hyphen", bg="red", font="15")

Forename\_Space=tkinter.Label(frame6, text="forename contains a space(s)", bg="red", font="15")

Forename\_Digit=tkinter.Label(frame6, text="forename contains a digit", bg="red", font="15")

Forename\_Char=tkinter.Label(frame6, text="forename contains an invalid character", bg="red", font="15")

#error messages for date

Date\_WrongLen=tkinter.Label(frame6, text="date is not a valid length", bg="red", font="15")

Date\_Year=tkinter.Label(frame6, text="must be 2022", bg="red", font="15")

Date\_Char6=tkinter.Label(frame6, text="character 6 is not a /", bg="red", font="15")

Date\_Month=tkinter.Label(frame6, text="month entered is not valid", bg="red", font="15")

Date\_Char3=tkinter.Label(frame6, text="charcter 3 is not a /", bg="red", font="15")

Date\_Day=tkinter.Label(frame6, text="day entered is not valid", bg="red", font="15")

d1\_Great = tkinter.Label(frame6, text="date entered has already happened", bg="red", font="15")

Date\_invalid = tkinter.Label(frame6, text="date contains something invalid e.g. day or month" , bg="red",font="15")

#error message when data and time entered has already been booked

Space\_taken = tkinter.Label(frame6, text="data and time entered is already taken", bg="red", font ="15")

**def** **validate\_name**(name, entry):

**if** name == "surname":

Surname\_Error = True

Forename\_Error = False

ViewSur\_Error = False

**elif** name == "forename":

Forename\_Error = True

Surname\_Error = False

ViewSur\_Error = False

**else**:

ViewSur\_Error = True

Surname\_Error = False

Forename\_Error = False

BookApp\_Valid = False

counter\_name = **0**

**if** entry == "":

**if** Surname\_Error == True:

Surname\_NoEnt.pack(anchor = 'e')

**elif** Forename\_Error == True:

Forename\_NoEnt.pack(anchor = 'e')

**elif** ViewSur\_Error == True:

ForSurname\_NoEnt.pack()

**elif** "--" **in** entry:

**if** Surname\_Error == True:

Surname\_Double.pack(anchor = 'e')

**elif** Forename\_Error == True:

Forename\_Double.pack(anchor = 'e')

**elif** ViewSur\_Error == True:

ForSurname\_Double.pack()

**elif** " " **in** entry:

**if** Surname\_Error == True:

Surname\_Space.pack(anchor = 'e')

**elif** Forename\_Error == True:

Forename\_Space.pack(anchor = 'e')

**elif** ViewSur\_Error == True:

ForSurname\_Space.pack()

**else**:

**for** i **in** range(len(entry)):

**if** entry[i].isdigit():

**if** Surname\_Error == True:

Surname\_Digit.pack(anchor = 'e')

**break**

**elif** Forename\_Error == True:

Forename\_Digit.pack(anchor = 'e')

**break**

**elif** ViewSur\_Error == True:

ForSurname\_Digit.pack()

**break**

**if** entry[i].isalpha() **or** entry[i] == "-":

counter\_name = counter\_name + **1**

**else**:

**if** Surname\_Error == True:

Surname\_Char.pack(anchor = 'e')

**elif** Forename\_Error == True:

Forename\_Char.pack(anchor = 'e')

**elif** ViewSur\_Error == True:

ForSurname\_Char.pack()

**if** counter\_name == len(entry):

BookApp\_Valid = True

**return** BookApp\_Valid

**return** False

**def** **validate\_date**(Type,entry):

date\_valid = False

counter\_date = **0**

**if** len(entry) != **10**:

**if** Type == 'Save':

Date\_WrongLen.pack(anchor='e')

**elif** Type == 'Edit':

Date\_WrongLen2.pack(anchor = 'e')

**else**:

**if** entry[**6**:**9**].isdigit():

**if** int(entry[**6**]) == **2** **and** int(entry[**7**]) == **0** **and** int(entry[**8**]) == **2** **and** int(entry[**9**]) == **1** **or** int(entry[**9**]) == **2**:

counter\_date = counter\_date + **1**

**pass**

**else**:

**if** Type == 'Save':

Date\_Year.pack(anchor='e')

**elif** Type == 'Edit':

Date\_Year2.pack(anchor = 'e')

**else**:

**if** Type == 'Save':

Date\_Year.pack(anchor='e')

**elif** Type == 'Edit':

Date\_Year2.pack(anchor = 'e')

**if** entry[**5**] == "/":

counter\_date = counter\_date + **1**

**pass**

**else**:

**if** Type == 'Save':

Date\_Char6.pack(anchor='e')

**elif** Type == 'Edit':

Date\_Char62.pack(anchor = 'e')

**if** entry[**3**].isdigit():

counter\_date = counter\_date + **1**

**pass**

**else**:

**if** Type == 'Save':

Date\_Month.pack(anchor='e')

**elif** Type == 'Edit':

Date\_Month2.pack(anchor = 'e')

**if** entry[**2**] == "/":

counter\_date = counter\_date + **1**

**pass**

**else**:

**if** Type == 'Save':

Date\_Char3.pack(anchor='e')

**elif** Type == 'Edit':

Date\_Char32.pack(anchor = 'e')

**if** entry[**0**].isdigit():

counter\_date = counter\_date + **1**

**pass**

**else**:

**if** Type == 'Save':

Date\_Day.pack(anchor='e')

**elif** Type == 'Edit':

Date\_Day2.pack(anchor = 'e')

**if** entry[**1**].isdigit():

counter\_date=counter\_date+**1**

**pass**

**else**:

**if** Type == 'Save':

Date\_Day.pack(anchor='e')

**elif** Type == 'Edit':

Date\_Day2.pack(anchor = 'e')

**if** counter\_date == **6**:

d1 = entry[**0**] + entry[**1**]

m1 = entry[**3**] + entry[**4**]

y1 = entry[**6**] + entry[**7**] + entry[**8**] + entry[**9**]

d = int(d1)

m = int(m1)

y = int(y1)

# date in yyyy/mm/dd format

d1 = datetime.datetime.now()

**try**:

d2 = datetime.datetime(y, m, d)

# Comparing the dates will return

# either True or False

**if** d1 > d2:

**if** Type == 'Save':

d1\_Great.pack(anchor='e')

**elif** Type == 'Edit':

d1\_Great2.pack(anchor = 'e')

**elif** d1 < d2:

date\_valid = True

**except**:

**if** Type == 'Save':

Date\_invalid.pack(anchor='e')

**elif** Type == 'Edit':

Date\_invalid2.pack(anchor = 'e')

**return** date\_valid

**def** **check\_data\_repeat**(Type,entry, time\_entry):

repeat = False

taken = False

appointment\_data\_repeat = []

**for** i **in** range (len(Appointment\_Data)):

**if** entry == Appointment\_Data[i][**3**]:

appointment\_data\_repeat.append(i)

repeat = True

**if** repeat == True:

**for** i **in** range (len(appointment\_data\_repeat)):

**if** time\_entry == Appointment\_Data[appointment\_data\_repeat[i]][**4**]:

taken = True

**if** taken == True:

**if** Type == 'Save':

Space\_taken.pack()

**elif** Type == 'Edit':

Space\_taken2.pack(side=tkinter.RIGHT)

**return** repeat, taken

#labels to show the user the status of the booking

Unique\_ID=tkinter.Label(frame6,text=Unique\_Value\_Text,bg="#EC927F",fg="white",font="15")

Data\_Saved=tkinter.Label(frame6,text="Data has been saved",bg="green",font="15")

Data\_NotSaved=tkinter.Label(frame6,text="Data has not been saved",bg="red",font="15")

#label to show the user that the csv is open

Csv\_open=tkinter.Label(frame6,text="Csv is open, close csv and click save again", bg= "red",font="15")

**def** **Save**():

**global** counter

Unique\_Value,Unique\_Value\_Text = Unique\_val\_calc()

Clear\_Errors()

Surname\_BookApp\_Valid = validate\_name("surname", BookApp\_surname\_Ent.get())

Forename\_BookApp\_Valid = validate\_name("forename", BookApp\_forename\_Ent.get())

Data\_BookApp\_Valid = validate\_date("Save",BookApp\_Date\_Ent.get())

repeat , taken = check\_data\_repeat("Save",BookApp\_Date\_Ent.get(),variable\_times.get())

**if** taken == False:

Time\_BookApp = variable\_times.get()

Type\_BookApp = variable\_services.get()

Surname\_BookApp = BookApp\_surname\_Ent.get().lower()

Forename\_BookApp = BookApp\_forename\_Ent.get().lower()

Date\_BookApp = BookApp\_Date\_Ent.get()

**if** Surname\_BookApp\_Valid == True **and** Forename\_BookApp\_Valid == True **and** Data\_BookApp\_Valid == True:

**try**:

f=open("Appointment ID.csv","a")

Appointment\_info = str(Unique\_Value) + "," + Surname\_BookApp + "," + Forename\_BookApp + "," + Date\_BookApp + "," + Time\_BookApp + "," + Type\_BookApp + "**\n**"

f.write(Appointment\_info)

f.close()

Unique\_ID.configure(text=Unique\_Value\_Text)

Unique\_ID.pack(anchor="e")

Data\_Saved.pack(anchor="e")

Appointment\_Data.append([])

Appointment\_Data[counter].append(str(Unique\_Value))

Appointment\_Data[counter].append(Surname\_BookApp)

Appointment\_Data[counter].append(Forename\_BookApp)

Appointment\_Data[counter].append(Date\_BookApp)

Appointment\_Data[counter].append(Time\_BookApp)

Appointment\_Data[counter].append(Type\_BookApp)

counter = counter + **1**

**except**:

Csv\_open.pack()

**else**:

Data\_NotSaved.pack(anchor='e')

**if** Surname\_BookApp\_Valid == False **or** Forename\_BookApp\_Valid == False **or** Data\_BookApp\_Valid == False **or** repeat == True **or** taken == True:

BookApp\_Clear\_Bttn.configure(text="Click to remove error and data")

#buttons

BookApp\_Save\_Bttn=tkinter.Button(frame7, text="Save Appointment", bg = "yellow", font = "calibri 15", command= Save)

BookApp\_Save\_Bttn.pack(side=tkinter.RIGHT)#save button to save data to a csv

BookApp\_surname\_Ent=tkinter.Entry(frame2,width=**20**,background="lightgrey")

BookApp\_surname\_Ent.pack(side=tkinter.RIGHT)#entry box for surname

BookApp\_forename\_Ent=tkinter.Entry(frame3, width=**20**,background="lightgrey")

BookApp\_forename\_Ent.pack(side=tkinter.RIGHT)#entry box for forename

BookApp\_Date\_Ent=tkinter.Entry(frame4,width=**20**,background="lightgrey")

BookApp\_Date\_Ent.pack(side=tkinter.RIGHT)#entry box for date

**def** **Clear\_Data**():

BookApp\_surname\_Ent.delete(**0**,**100**)

BookApp\_forename\_Ent.delete(**0**,**100**)

BookApp\_Date\_Ent.delete(**0**,**100**)

variable\_services.set(services[**0**])

variable\_times.set(times[**0**])

**def** **Clear\_Errors**():

Surname\_NoEnt.pack\_forget()

Surname\_Double.pack\_forget()

Surname\_Space.pack\_forget()

Surname\_Digit.pack\_forget()

Surname\_Char.pack\_forget()

Forename\_NoEnt.pack\_forget()

Forename\_Double.pack\_forget()

Forename\_Space.pack\_forget()

Forename\_Digit.pack\_forget()

Forename\_Char.pack\_forget()

Date\_WrongLen.pack\_forget()

Date\_Year.pack\_forget()

Date\_Char6.pack\_forget()

Date\_Month.pack\_forget()

Date\_Char3.pack\_forget()

Date\_Day.pack\_forget()

Data\_Saved.pack\_forget()

Data\_NotSaved.pack\_forget()

Unique\_ID.pack\_forget()

Csv\_open.pack\_forget()

d1\_Great.pack\_forget()

Date\_invalid.pack\_forget()

Space\_taken.pack\_forget()

**def** **Clear\_Button**():

Clear\_Data()

Clear\_Errors()

BookApp\_Clear\_Bttn.configure(text="Clear Data")

#button to clear data from booking page

BookApp\_Clear\_Bttn=tkinter.Button(frame7, text="Clear Data", bg="yellow",fg="black", font="calibri 15", command=Clear\_Button)

BookApp\_Clear\_Bttn.pack()

#view appointment

ViewApp=tkinter.Toplevel(AppWindow)

ViewApp.title("View your Appointment")

ViewApp.geometry("600x550")

ViewApp.configure(bg="lightblue")

ViewApp.withdraw()

#ViewApp.attributes('-fullscreen', True)

#viewapp frames

ViewAppframe1=tkinter.Frame(ViewApp)

ViewAppframe1.configure(bg="lightblue")

ViewAppframe1.pack()

ViewAppframe2=tkinter.Frame(ViewApp)

ViewAppframe2.configure(bg="lightblue")

ViewAppframe2.pack()

ViewAppframe3=tkinter.Frame(ViewApp)

ViewAppframe3.configure(bg="lightblue")

ViewAppframe3.pack()

ViewAppframe4=tkinter.Frame(ViewApp)

ViewAppframe4.configure(bg="lightblue")

ViewAppframe4.pack()

ViewAppframe5=tkinter.Frame(ViewApp)

ViewAppframe5.configure(bg="lightblue")

ViewAppframe5.pack()

#title label

ViewApp\_Space = tkinter.Label(ViewAppframe1,text="Nails By Katie - Book an Appointment",bg="lightblue",fg="lightblue",font="calibri 20 bold")

ViewApp\_Space.pack()

ViewApp\_Title\_Lbl=tkinter.Label(ViewAppframe1, text="Nails By Katie - View your Appointment", bg="#EC927F",fg="black",font="calibri 25")

ViewApp\_Title\_Lbl.pack()

#unique id label and entry box

ViewApp\_Space = tkinter.Label(ViewAppframe2,text="Nails By Katie - Admin Page",bg="lightblue",fg="lightblue",font="calibri 25 bold")

ViewApp\_Space.pack()

ViewApp\_UniID\_Lbl=tkinter.Label(ViewAppframe2,text="Unique ID for Appointment",bg="#EC927F",fg="black",font="calibri 15")

ViewApp\_UniID\_Lbl.pack(side=tkinter.LEFT)

ViewApp\_UniID\_Ent=tkinter.Entry(ViewAppframe2,width=**20**,background="lightgrey")

ViewApp\_UniID\_Ent.pack(side=tkinter.RIGHT)

#placeholder for space to display information

ViewApp\_SpaceMd = tkinter.Label(ViewAppframe3,text="Nails By Katie - Book an Appointment",bg="lightblue",fg="lightblue",font="calibri 30 bold")

ViewApp\_SpaceMd.pack()

ViewApp\_SpaceMd2= tkinter.Label(ViewAppframe3,text="Nails By Katie - Book an Appointment",bg="lightblue",fg="lightblue",font="calibri 30 bold")

ViewApp\_SpaceMd2.pack()

ViewApp\_SpaceMd3 = tkinter.Label(ViewAppframe3,text="Nails By Katie - Book an Appointment",bg="lightblue",fg="lightblue",font="calibri 30 bold")

ViewApp\_SpaceMd3.pack()

#search id button

ViewApp\_Space = tkinter.Label(ViewAppframe4,text="Nails By Katie - Admin Page",bg="lightblue",fg="lightblue",font="calibri 25 bold")

ViewApp\_Space.pack()

ViewApp\_Forgot\_Bttn=tkinter.Button(ViewAppframe4,text="Forgotten ID",bg="yellow",fg="black",font="calibri 15",command=ViewApp\_Window\_Surname)

ViewApp\_Forgot\_Bttn.pack()

#create error messages

ID\_NoEnt=tkinter.Label(ViewAppframe3, text="ID has no entry", bg="red", font="15")

ID\_Space=tkinter.Label(ViewAppframe3, text="ID contains a space(s)", bg="red", font="15")

ID\_Char=tkinter.Label(ViewAppframe3, text="ID contains an invalid character", bg="red", font="15")

#Appointment not found label

Appointment\_NotFound=tkinter.Label(ViewAppframe3,text="A corresponding appointment could not be found",bg="red",font="15")

#takes first position in list to declare variables

Appointment\_UniID = Appointment\_Data[**0**][**0**]

Appointment\_Surname = Appointment\_Data[**0**][**1**]

Appointment\_Forename = Appointment\_Data[**0**][**2**]

Appointment\_Date = Appointment\_Data[**0**][**3**]

Appointment\_Time = Appointment\_Data[**0**][**4**]

Appointment\_Type = Appointment\_Data[**0**][**5**]

#function to set data

**def** **set\_data**(Appointment\_UniID,Appointment\_Surname,Appointment\_Forename,Appointment\_Date,Appointment\_Time,Appointment\_Type):

Uni\_ID\_Str="Unique ID: " + str(Appointment\_UniID)

Surname\_Str="Surname: " + str(Appointment\_Surname)

Forename\_Str="Forename: " + str(Appointment\_Forename)

Date\_Str="Date: " + str(Appointment\_Date)

Time\_Str="Time: " + str(Appointment\_Time)

Type\_Str="Type of Appointment: " + str(Appointment\_Type)

**return** Uni\_ID\_Str,Surname\_Str,Forename\_Str,Date\_Str,Time\_Str,Type\_Str

#call function to set up data

Uni\_ID\_Str,Surname\_Str,Forename\_Str,Date\_Str,Time\_Str,Type\_Str = set\_data(Appointment\_UniID,Appointment\_Surname,Appointment\_Forename,Appointment\_Date,Appointment\_Time,Appointment\_Type)

#labels to display information

Appointment\_UniqueID\_Label=tkinter.Label(ViewAppframe3,text=Uni\_ID\_Str,bg="#EC927F",fg="white",font="15")

Appointment\_Surname\_Label=tkinter.Label(ViewAppframe3,text=Surname\_Str,bg="#EC927F",fg="white",font="15")

Appointment\_Forename\_Label=tkinter.Label(ViewAppframe3,text=Forename\_Str,bg="#EC927F",fg="white",font="15")

Appointment\_Date\_Label=tkinter.Label(ViewAppframe3,text=Date\_Str,bg="#EC927F",fg="white",font="15")

Appointment\_Time\_Label=tkinter.Label(ViewAppframe3,text=Time\_Str,bg="#EC927F",fg="white",font="15")

Appointment\_Type\_Label=tkinter.Label(ViewAppframe3,text=Type\_Str,bg="#EC927F",fg="white",font="15")

#function to find the appointment position within list

**def** **find\_app**(ID\_ent):

position = **0**

**for** i **in** range (len(Appointment\_Data)):

**if** Appointment\_Data[i][**0**] == ID\_ent:

Appointment\_Position = i

Appointment\_Found = True

**break**

**else**:

position = position + **1**

**if** position == len(Appointment\_Data) - **1**:

Appointment\_Found = False

Appointment\_Position = **0**

**return** Appointment\_Found, Appointment\_Position

#command to search the id

**def** **search\_ID**():

clear\_ID\_Error()

valid\_ID,ID\_ent = validate\_ID('Search')

**if** valid\_ID == True:

Appointment\_Found, Appointment\_Position = find\_app(ID\_ent)

**if** Appointment\_Found ==True:

Appointment\_UniID = Appointment\_Data[Appointment\_Position][**0**]

Appointment\_Surname = Appointment\_Data[Appointment\_Position][**1**]

Appointment\_Forename = Appointment\_Data[Appointment\_Position][**2**]

Appointment\_Date = Appointment\_Data[Appointment\_Position][**3**]

Appointment\_Time = Appointment\_Data[Appointment\_Position][**4**]

Appointment\_Type = Appointment\_Data[Appointment\_Position][**5**]

Uni\_ID\_Str,Surname\_Str,Forename\_Str,Date\_Str,Time\_Str,Type\_Str = set\_data(Appointment\_UniID,Appointment\_Surname,Appointment\_Forename,Appointment\_Date,Appointment\_Time,Appointment\_Type)

Appointment\_UniqueID\_Label.configure(text=Uni\_ID\_Str)

Appointment\_Surname\_Label.configure(text=Surname\_Str)

Appointment\_Forename\_Label.configure(text=Forename\_Str)

Appointment\_Date\_Label.configure(text=Date\_Str)

Appointment\_Time\_Label.configure(text=Time\_Str)

Appointment\_Type\_Label.configure(text=Type\_Str)

ViewApp\_SpaceMd.pack\_forget()

ViewApp\_SpaceMd2.pack\_forget()

Appointment\_UniqueID\_Label.pack()

Appointment\_Surname\_Label.pack()

Appointment\_Forename\_Label.pack()

Appointment\_Date\_Label.pack()

Appointment\_Time\_Label.pack()

Appointment\_Type\_Label.pack()

**elif** Appointment\_Found == False:

Appointment\_NotFound.pack()

#validation function for the ID entry

**def** **validate\_ID**(Type):

**if** Type == 'Search':

ID\_ent = ViewApp\_UniID\_Ent.get()

valid\_ID=False

**if** ID\_ent.isalpha():

ID\_Char.pack()

**elif** ID\_ent == "":

ID\_NoEnt.pack()

**elif** " " **in** ID\_ent:

ID\_Space.pack()

**else**:

**if** ID\_ent.isdigit():

valid\_ID=True

**else**:

ID\_Char.pack()

**elif** Type == 'Edit':

ID\_ent = EditApp\_UniID\_Ent.get()

valid\_ID=False

**if** ID\_ent.isalpha():

ID\_Char2.pack(anchor = 'e')

**elif** ID\_ent == "":

ID\_NoEnt2.pack(anchor = 'e')

**elif** " " **in** ID\_ent:

ID\_Space2.pack(anchor = 'e')

**else**:

**if** ID\_ent.isdigit():

valid\_ID=True

**else**:

ID\_Char2.pack(anchor = 'e')

**return** valid\_ID,ID\_ent

**def** **clear\_ID\_Error**():

ID\_NoEnt.pack\_forget()

ID\_Char.pack\_forget()

ID\_Space.pack\_forget()

Appointment\_NotFound.pack\_forget()

ViewApp\_SpaceMd.pack()

ViewApp\_SpaceMd2.pack()

Appointment\_UniqueID\_Label.pack\_forget()

Appointment\_Surname\_Label.pack\_forget()

Appointment\_Forename\_Label.pack\_forget()

Appointment\_Date\_Label.pack\_forget()

Appointment\_Time\_Label.pack\_forget()

Appointment\_Type\_Label.pack\_forget()

#button to return to main menu

ViewApp\_Return\_Bttn=tkinter.Button(ViewAppframe5,text="Return to previous menu",bg="yellow",fg="black",font="calibri 15",command=AppMenu\_Window)

ViewApp\_Return\_Bttn.pack(side=tkinter.LEFT)

#search button

ViewApp\_Srch\_Bttn=tkinter.Button(ViewAppframe5,text="Search ID for appointment",bg="yellow",fg="black",font="calibri 15",command=search\_ID)

ViewApp\_Srch\_Bttn.pack(side=tkinter.RIGHT)

#creating the branch window where the user can input their surname

ViewAppSur=tkinter.Toplevel(ViewApp)

ViewAppSur.title("Forgot ID")

ViewAppSur.geometry("650x500")

ViewAppSur.configure(bg="lightblue")

ViewAppSur.withdraw()

#creating frames for the window

frame1=tkinter.Frame(ViewAppSur)

frame1.configure(bg="lightblue")

frame1.pack()

frame2=tkinter.Frame(ViewAppSur)

frame2.configure(bg="lightblue")

frame2.pack()

frame3=tkinter.Frame(ViewAppSur)

frame3.configure(bg="lightblue")

frame3.pack()

frame4=tkinter.Frame(ViewAppSur)

frame4.configure(bg="lightblue")

frame4.pack()

#title label

ViewAppSur\_Space = tkinter.Label(frame1,text="Nails By Katie - Book an Appointment",bg="lightblue",fg="lightblue",font="calibri 20 bold")

ViewAppSur\_Space.pack()

ViewAppSur\_Title\_Lbl=tkinter.Label(frame1, text="Nails By Katie - Forgot ID", bg="#EC927F",fg="black",font="calibri 25")

ViewAppSur\_Title\_Lbl.pack()

#surname label and entry box

ViewAppSur\_Space = tkinter.Label(frame2,text="Nails By Katie - Admin Page",bg="lightblue",fg="lightblue",font="calibri 25 bold")

ViewAppSur\_Space.pack()

ViewAppSur\_Surname\_Lbl=tkinter.Label(frame2,text="Surname for Appointment",bg="#EC927F",fg="black",font="calibri 15")

ViewAppSur\_Surname\_Lbl.pack(side=tkinter.LEFT)

ViewAppSur\_Surname\_Ent=tkinter.Entry(frame2,width=**20**,background="lightgrey")

ViewAppSur\_Surname\_Ent.pack(side=tkinter.RIGHT)

#reserve space between entry and buttons at the bottom of the page

ViewAppSur\_Space = tkinter.Label(frame3,text="Nails By Katie - Book an Appointment",bg="lightblue",fg="lightblue",font="calibri 15 bold")

ViewAppSur\_Space.pack()

ViewAppSur\_Space = tkinter.Label(frame3,text="Nails By Katie - Book an Appointment",bg="lightblue",fg="lightblue",font="calibri 30 bold")

ViewAppSur\_Space.pack()

#return to previous menu button

ViewAppSur\_Space = tkinter.Label(frame4,text="Nails By Katie - Admin Page",bg="lightblue",fg="lightblue",font="calibri 25 bold")

ViewAppSur\_Space.pack()

ViewAppSur\_Return\_Bttn=tkinter.Button(frame4,text="Return to previous menu",bg="yellow",fg="black",font="calibri 15",command=ViewApp\_Window)

ViewAppSur\_Return\_Bttn.pack(side=tkinter.LEFT)

#error message labels

ForSurname\_NoEnt=tkinter.Label(frame3, text="surname has no entry", bg="red", font="15")

ForSurname\_Double=tkinter.Label(frame3, text="surname contains repeated hyphen", bg="red", font="15")

ForSurname\_Space=tkinter.Label(frame3, text="surname contains a space(s)", bg="red", font="15")

ForSurname\_Digit=tkinter.Label(frame3, text="surname contains a digit", bg="red", font="15")

ForSurname\_Char=tkinter.Label(frame3, text="surname contains an invalid character", bg="red", font="15")

#error message for no appointment found

ForSurname\_SurnameNotFound=tkinter.Label(frame3,text="surname has no corresponding appointment(s)",bg="red",font="15")

#search button

appointment\_surname\_position = []

appointment\_surname\_position\_past = []

appointment\_surname\_dates = []

temp\_position = []

distance\_date = []

**def** **search\_surname**():

appointment\_surname\_position.clear()#clears list when the function is run

position = **0**

surname\_tosearch = ViewAppSur\_Surname\_Ent.get().lower()

**for** i **in** range(len(Appointment\_Data)):

**if** Appointment\_Data[i][**1**] == surname\_tosearch:

appointment\_surname\_position.append(i)

Appointment\_Found = True

**else**:

position = position + **1**

**if** position == len(Appointment\_Data):

Appointment\_Found = False

**return** Appointment\_Found,surname\_tosearch

**def** **compareDate**():

appointment\_surname\_position\_past.clear()

appointment\_surname\_dates.clear()

**for** i **in** range(len(appointment\_surname\_position)):

CurrentAppDate = Appointment\_Data[appointment\_surname\_position[i]][**3**]

d1 = CurrentAppDate[**0**] + CurrentAppDate[**1**]

m1 = CurrentAppDate[**3**] + CurrentAppDate[**4**]

y1 = CurrentAppDate[**6**] + CurrentAppDate[**7**] + CurrentAppDate[**8**] + CurrentAppDate[**9**]

d = int(d1)

m = int(m1)

y = int(y1)

# date in yyyy/mm/dd format

d1 = datetime.datetime.now()

d2 = datetime.datetime(y, m, d)

**if** d2 > d1:

appointment\_surname\_position\_past.append(appointment\_surname\_position[i])

**def** **orderDate**(appointment\_surname\_dates,surname\_tosearch,distance\_date):

temp\_dates = []

new\_date\_order = []

**for** i **in** range (len(appointment\_surname\_position\_past)):

appointment\_surname\_dates.append(Appointment\_Data[appointment\_surname\_position\_past[i]][**3**])

#order the dates based on their distances, pass the list back of ordered dates and use same i and y range loops

date\_current = datetime.datetime.now()

**for** i **in** range(len(appointment\_surname\_dates)):

CurrentAppDate = appointment\_surname\_dates[i]

d1 = CurrentAppDate[**0**] + CurrentAppDate[**1**]

m1 = CurrentAppDate[**3**] + CurrentAppDate[**4**]

y1 = CurrentAppDate[**6**] + CurrentAppDate[**7**] + CurrentAppDate[**8**] + CurrentAppDate[**9**]

d = int(d1)

m = int(m1)

y = int(y1)

d2 = datetime.datetime(y, m, d)

temp\_dates.append(d2)

distance = d2 - date\_current

distance = distance.days

distance\_date.append(distance)

distance\_date = sorted(distance\_date)

**for** i **in** range (len(appointment\_surname\_dates)):

order = False

pointer = **0**

**while** order == False:

value = temp\_dates[pointer] - date\_current

**if** distance\_date[i] == value.days:

new\_date\_order.append(appointment\_surname\_dates[pointer])

order = True

**else**:

pointer = pointer + **1**

**for** i **in** range(len(appointment\_surname\_dates)):

**for** y **in** range(len(Appointment\_Data)):

**if** new\_date\_order[i] == Appointment\_Data[y][**3**] **and** surname\_tosearch == Appointment\_Data[y][**1**]:

temp\_position.append(y)

**return** temp\_position

#creating strings for appointment display labels

SearchSurname\_AppInfo\_Str = "0"

#label to display information

SearchSurname\_AppInfo\_Lbl\_1=tkinter.Label(frame3,text=SearchSurname\_AppInfo\_Str,bg="#EC927F",fg="white",font="calibri 9")

SearchSurname\_AppInfo\_Lbl\_2=tkinter.Label(frame3,text=SearchSurname\_AppInfo\_Str,bg="#EC927F",fg="white",font="calibri 9")

SearchSurname\_AppInfo\_Lbl\_3=tkinter.Label(frame3,text=SearchSurname\_AppInfo\_Str,bg="#EC927F",fg="white",font="calibri 9")

**def** **app\_SearchSurname\_variables**(i):

#the function will now create variables based on the position and back them using a different function

UniqueID\_SearchSurname = Appointment\_Data[temp\_position[i]][**0**]

Surname\_SearchSurname = Appointment\_Data[temp\_position[i]][**1**]

Forename\_SearchSurname = Appointment\_Data[temp\_position[i]][**2**]

Date\_SearchSurname = Appointment\_Data[temp\_position[i]][**3**]

Time\_SearchSurname = Appointment\_Data[temp\_position[i]][**4**]

Type\_SearchSurname = Appointment\_Data[temp\_position[i]][**5**]

UniqueID\_SearchSurname\_Str="Unique ID: " + str(UniqueID\_SearchSurname)

Surname\_SearchSurname\_Str=" Surname: " + str(Surname\_SearchSurname)

Forename\_SearchSurname\_Str=" Forename: " + str(Forename\_SearchSurname)

Date\_SearchSurname\_Str=" Date: " + str(Date\_SearchSurname)

Time\_SearchSurname\_Str=" Time: " + str(Time\_SearchSurname)

Type\_SearchSurname\_Str=" Type of Appointment: " + str(Type\_SearchSurname)

SearchSurname\_AppInfo\_Str = UniqueID\_SearchSurname\_Str + Surname\_SearchSurname\_Str + Forename\_SearchSurname\_Str + Date\_SearchSurname\_Str + Time\_SearchSurname\_Str + Type\_SearchSurname\_Str

**return** SearchSurname\_AppInfo\_Str

**def** **packlabels**(SearchSurname\_AppInfo\_Str,pointer):

**if** pointer == **1**:

SearchSurname\_AppInfo\_Lbl\_1.configure(text = SearchSurname\_AppInfo\_Str)

SearchSurname\_AppInfo\_Lbl\_1.pack()

**elif** pointer == **2**:

SearchSurname\_AppInfo\_Lbl\_2.configure(text=SearchSurname\_AppInfo\_Str)

SearchSurname\_AppInfo\_Lbl\_2.pack()

**elif** pointer == **3**:

SearchSurname\_AppInfo\_Lbl\_3.configure(text=SearchSurname\_AppInfo\_Str)

SearchSurname\_AppInfo\_Lbl\_3.pack()

**def** **clearErrors**():

SearchSurname\_AppInfo\_Lbl\_1.pack\_forget()

SearchSurname\_AppInfo\_Lbl\_2.pack\_forget()

SearchSurname\_AppInfo\_Lbl\_3.pack\_forget()

ForSurname\_NoEnt.pack\_forget()

ForSurname\_Double.pack\_forget()

ForSurname\_Space.pack\_forget()

ForSurname\_Digit.pack\_forget()

ForSurname\_Char.pack\_forget()

**def** **SearchSurname**():

pointer = **1**

clearErrors()

ForSurname\_valid = validate\_name("forgot-surname", ViewAppSur\_Surname\_Ent.get())

**if** ForSurname\_valid == True:

Appointment\_Found,surname\_tosearch = search\_surname()

**if** Appointment\_Found == True:

compareDate()

temp\_position = orderDate(appointment\_surname\_dates,surname\_tosearch,distance\_date)

**if** len(appointment\_surname\_position\_past) <= **3**:

**for** i **in** range (len(appointment\_surname\_position\_past)):

SearchSurname\_AppInfo\_Str = app\_SearchSurname\_variables(i)

packlabels(SearchSurname\_AppInfo\_Str,pointer)

pointer = pointer + **1**

**else**:

**for** i **in** range (**3**):

SearchSurname\_AppInfo\_Str = app\_SearchSurname\_variables(i)

packlabels(SearchSurname\_AppInfo\_Str,pointer)

pointer = pointer + **1**

**if** Appointment\_Found == False:

ForSurname\_SurnameNotFound.pack()

#search button for surname

ViewAppSur\_Srch\_Bttn=tkinter.Button(frame4,text="Search surname for appointment(s)",bg="yellow",fg="black",font="calibri 15",command=SearchSurname)

ViewAppSur\_Srch\_Bttn.pack(side=tkinter.RIGHT)

#edit appointment window

EditApp=tkinter.Toplevel(AppWindow)

EditApp.title("Edit your Appointment")

EditApp.geometry("550x450")

EditApp.configure(bg="lightblue")

EditApp.withdraw()

#frame creation

EditAppframe1=tkinter.Frame(EditApp)

EditAppframe1.configure(bg="lightblue")

EditAppframe1.pack()

EditAppframe2=tkinter.Frame(EditApp)

EditAppframe2.configure(bg="lightblue")

EditAppframe2.pack()

EditAppframe3=tkinter.Frame(EditApp)

EditAppframe3.configure(bg="lightblue")

EditAppframe3.pack()

EditAppframe4=tkinter.Frame(EditApp)

EditAppframe4.configure(bg="lightblue")

EditAppframe4.pack()

EditAppframe5=tkinter.Frame(EditApp)

EditAppframe5.configure(bg="lightblue")

EditAppframe5.pack()

#space and title of the window

EditApp\_Space = tkinter.Label(EditAppframe1,text="Nails By Katie - Book an Appointment",bg="lightblue",fg="lightblue",font="calibri 20 bold")

EditApp\_Space.pack()

EditApp\_Title\_Lbl=tkinter.Label(EditAppframe1, text="Nails By Katie - Edit your Appointment", bg="#EC927F",fg="black",font="calibri 25")

EditApp\_Title\_Lbl.pack()

#space, entry box and label

EditApp\_Space = tkinter.Label(EditAppframe2,text="Nails By Katie - Admin Page",bg="lightblue",fg="lightblue",font="calibri 25 bold")

EditApp\_Space.pack()

EditApp\_UniID\_Lbl = tkinter.Label(EditAppframe2,text="Unique ID:",bg="#EC927F",fg="black",font="calibri 15")

EditApp\_UniID\_Lbl.pack(side=tkinter.LEFT)

EditApp\_UniID\_Ent = tkinter.Entry(EditAppframe2,width=**20**,background="lightgrey")

EditApp\_UniID\_Ent.pack(side=tkinter.RIGHT)

#space between unique ID and drop down menus

EditApp\_Space = tkinter.Label(EditAppframe3,text="Nails By Katie - Admin Page",bg="lightblue",fg="lightblue",font="calibri 25 bold")

EditApp\_Space.pack()

#create lists for drop down to choose field to change

field\_change = ["Date","Time","Type"]

variable\_field=tkinter.StringVar()

variable\_field.set(field\_change[**0**])

EditApp\_field\_dropdown=tkinter.OptionMenu(

EditAppframe3,

variable\_field,

\*field\_change)

EditApp\_field\_dropdown.pack(side=tkinter.LEFT)

#times drop down menu

#already declared on line 260

EditApp\_Time\_dropdown=tkinter.OptionMenu(

EditAppframe3,

variable\_times,

\*times)

#entry box to change date

EditApp\_Date\_Ent = tkinter.Entry(EditAppframe3,width = **20**, background = "lightgrey")

EditApp\_Date\_Ent.pack(side=tkinter.RIGHT)

#type drop down menu

#declared line 272

BookApp\_Type\_DropDown=tkinter.OptionMenu(

EditAppframe3,

variable\_services,

\*services)

#loop to look at field\_change option and pack accordingly

**def** **check\_choice**():

**global** choice

choice = variable\_field.get()

**if** choice == "Time":

BookApp\_Type\_DropDown.pack\_forget()

EditApp\_Date\_Ent.pack\_forget()

EditApp\_Time\_dropdown.pack(side=tkinter.RIGHT)

**if** choice == "Type":

BookApp\_Type\_DropDown.pack(side=tkinter.RIGHT)

EditApp\_Date\_Ent.pack\_forget()

EditApp\_Time\_dropdown.pack\_forget()

**if** choice == "Date":

BookApp\_Type\_DropDown.pack\_forget()

EditApp\_Date\_Ent.pack(side=tkinter.RIGHT)

EditApp\_Time\_dropdown.pack\_forget()

EditApp.after(**500**,check\_choice)

#validating entries

**def** **write\_csv**():

**try**:

f=open("Appointment ID.csv","w")

**for** i **in** range (len(Appointment\_Data)):

line=""

line=Appointment\_Data[i][**0**]+","+Appointment\_Data[i][**1**]+","+Appointment\_Data[i][**2**]+","+Appointment\_Data[i][**3**]+","+Appointment\_Data[i][**4**]+","+Appointment\_Data[i][**5**]+"**\n**"

f.write(line)

f.close()

Edit\_success.pack(side=tkinter.RIGHT)

**except**:

Csv\_open2.pack(side=tkinter.RIGHT)

**def** **save\_edit**():

clear\_edit\_errors()

valid\_ID,ID\_ent = validate\_ID('Edit')

**if** choice == "Date":

date\_valid = validate\_date('Edit',EditApp\_Date\_Ent.get())

**if** valid\_ID == True **and** date\_valid == True:

Appointment\_Found, Appointment\_Position = find\_app(ID\_ent)

**if** Appointment\_Found == True:

**if** choice == 'Date':

repeat , taken = check\_data\_repeat("Edit",EditApp\_Date\_Ent.get(),Appointment\_Data[Appointment\_Position][**4**])

**if** taken == False:

**print**(" hi save")

#save edit

Appointment\_Data[Appointment\_Position][**3**] = EditApp\_Date\_Ent.get()

write\_csv()

**elif** Appointment\_Found == False:

Appointment\_NotFound2.pack(side=tkinter.RIGHT)

**else**:

**if** valid\_ID == True:

Appointment\_Found, Appointment\_Position = find\_app(ID\_ent)

**if** Appointment\_Found == False:

Appointment\_NotFound2.pack(side=tkinter.RIGHT)

**else**:

**if** choice == 'Time':

repeat = False

taken = False

appointment\_data\_repeat = []

**for** i **in** range (len(Appointment\_Data)):

**if** Appointment\_Data[Appointment\_Position][**3**] == Appointment\_Data[i][**3**]:

appointment\_data\_repeat.append(i)

repeat = True

**if** repeat == True:

**for** i **in** range (len(appointment\_data\_repeat)):

**if** Appointment\_Data[appointment\_data\_repeat[i]][**4**] == variable\_times.get():

taken = True

Space\_taken2.pack(side=tkinter.RIGHT)

**if** taken == False:

#save edit

Appointment\_Data[Appointment\_Position][**4**] = variable\_times.get()

write\_csv()

**elif** choice == 'Type':

Appointment\_Data[Appointment\_Position][**5**] = variable\_services.get()

write\_csv()

#save edit

EditApp.after(**0**,check\_choice)

#function to clear entry boxes and remove errors.

**def** **clear\_edit\_entries**():

EditApp\_UniID\_Ent.delete(**0**,**100**)

EditApp\_Date\_Ent.delete(**0**,**100**)

variable\_field.set(field\_change[**0**])

variable\_times.set(times[**0**])

variable\_services.set(services[**0**])

clear\_edit\_errors()

#function to clear error messages

**def** **clear\_edit\_errors**():

ID\_NoEnt2.pack\_forget()

ID\_Char2.pack\_forget()

ID\_Space2.pack\_forget()

Date\_WrongLen2.pack\_forget()

Date\_Year2.pack\_forget()

Date\_Char62.pack\_forget()

Date\_Month2.pack\_forget()

Date\_Char32.pack\_forget()

Date\_Day2.pack\_forget()

d1\_Great2.pack\_forget()

Date\_invalid2.pack\_forget()

Appointment\_NotFound2.pack\_forget()

Space\_taken2.pack\_forget()

Edit\_success.pack\_forget()

Csv\_open2.pack\_forget()

#space between field choice and cancel button

EditApp\_Space = tkinter.Label(EditAppframe4,text="Nails By Katie - Admin Page",bg="lightblue",fg="lightblue",font="calibri 25 bold")

EditApp\_Space.pack()

#creating cancel appointment button

EditApp\_Cancel\_Bttn=tkinter.Button(EditAppframe4,text="Cancel Appointment",bg="red",fg="black",font="calibri 14",command=CancelApp\_Window)

EditApp\_Cancel\_Bttn.pack(side=tkinter.LEFT)

#space between cancel button and return buttons

EditApp\_Space = tkinter.Label(EditAppframe5,text="Nails By Katie - Admin Page",bg="lightblue",fg="lightblue",font="calibri 25 bold")

EditApp\_Space.pack()

#return button

EditApp\_Return\_Bttn=tkinter.Button(EditAppframe5,text="Return to Menu",bg="yellow",fg="black",font="calibri 15",command=AppMenu\_Window)

EditApp\_Return\_Bttn.pack(side=tkinter.LEFT)

#clear button

EditApp\_Clear\_Bttn=tkinter.Button(EditAppframe5,text="Clear Entries",bg="yellow",fg="black",font="calibri 15",command=clear\_edit\_entries)

EditApp\_Clear\_Bttn.pack(side=tkinter.LEFT)

#save button

EditApp\_Save\_Bttn=tkinter.Button(EditAppframe5,text="Save Changes",bg="yellow",fg="black",font="calibri 15",command=save\_edit)

EditApp\_Save\_Bttn.pack(side=tkinter.RIGHT)

#id error messages

ID\_NoEnt2=tkinter.Label(EditAppframe4, text="ID has no entry", bg="red", font="15")

ID\_Space2=tkinter.Label(EditAppframe4, text="ID contains a space(s)", bg="red", font="15")

ID\_Char2=tkinter.Label(EditAppframe4, text="ID contains an invalid character", bg="red", font="15")

#date error messages

Date\_WrongLen2=tkinter.Label(EditAppframe4, text="Date is not a valid length", bg="red", font="15")

Date\_Year2=tkinter.Label(EditAppframe4, text="Must be 2022", bg="red", font="15")

Date\_Char62=tkinter.Label(EditAppframe4, text="Character 6 is not a /", bg="red", font="15")

Date\_Month2=tkinter.Label(EditAppframe4, text="Month entered is not valid", bg="red", font="15")

Date\_Char32=tkinter.Label(EditAppframe4, text="Charcter 3 is not a /", bg="red", font="15")

Date\_Day2=tkinter.Label(EditAppframe4, text="Day entered is not valid", bg="red", font="15")

d1\_Great2 = tkinter.Label(EditAppframe4, text="Date entered has already happened", bg="red", font="15")

Date\_invalid2 = tkinter.Label(EditAppframe4, text="Date contains something invalid e.g. day or month" , bg="red",font="15")

#appointment not found error message

Appointment\_NotFound2=tkinter.Label(EditAppframe4,text="Appointment not be found",bg="red",font="15")

#appointment time already taken error message

Space\_taken2 = tkinter.Label(EditAppframe4, text="data and time taken", bg="red", font ="15")

#label to confirm save

Edit\_success = tkinter.Label(EditAppframe4, text="edit successful",bg = "green",font= "15")

#csv is open and cannot make changes

Csv\_open2=tkinter.Label(EditAppframe4,text="Csv is open", bg= "red",font="15")

#creating window for cancel app window

CancelApp=tkinter.Toplevel(EditApp)

CancelApp.title("Cancel your Appointment")

CancelApp.geometry("440x100")

CancelApp.configure(bg="grey")

CancelApp.withdraw()

#frames for cancel app window

CancelAppframe1=tkinter.Frame(CancelApp)

CancelAppframe1.configure(bg="grey")

CancelAppframe1.pack()

CancelAppframe2=tkinter.Frame(CancelApp)

CancelAppframe2.configure(bg="grey")

CancelAppframe2.pack()

#label to ask user wether they want to cancel

ConfirmCancel\_Lbl = tkinter.Label(CancelAppframe1,text="Are you sure you want to cancel your appointment?",bg="lightgrey",fg="black",font="calibri 14")

ConfirmCancel\_Lbl.pack()

#space between title and buttons

CancelApp\_Space = tkinter.Label(CancelAppframe2,text="Nails By Katie - Admin Page",bg="grey",fg="grey",font="calibri 12 bold")

CancelApp\_Space.pack()

#yes button

Cancel\_Y\_Bttn=tkinter.Button(CancelAppframe2,text=" Yes ",bg="lightgrey",fg="green",width = "10",font="calibri 12",command=EditApp\_Window)

Cancel\_Y\_Bttn.pack(side=tkinter.LEFT)

Cancel\_N\_Bttn=tkinter.Button(CancelAppframe2,text=" No ",bg="lightgrey",fg="red",width = "10" ,font="calibri 12",command=EditApp\_Window)

Cancel\_N\_Bttn.pack(side=tkinter.RIGHT)