OCR GCE A COMPUTER SCIENCE PROJECT H446-03

Name :	Miguel Abreu
Candidate Number:	3001
Abbeygate Sixth Form College:	19364
Title of Project:	BSL comprehension

H446-03 – PROJECT CONTENTS

CONTENTS

A. Analysis 6

Problem Identification 6

Stakeholders 6

Research 7

Existing Solutions 7

Features of the proposed solution 7

Limitations of development 8

Project Requirements – Hardware and Software 8

Milestones 9

B. Design 10

Systems diagram 10

Project Design 11

Login form 11

Main Window 11

New-User Window 12

Change Password Window 13

Export Window 13

Input validation 14

Login inputs 14

New-user inputs 14

Change Password 15

Modules and Algorithms 15

Hashing algorithm 15

Iterative search 16

Authorisation 17

Menu 17

Menu Commands 18

Shutter Frame 19

Show Frame – Pipeline Display 20

Show Frame – PhotoImage Return 21

Test Data During Development/ Post Development 22

Milestone 1/2: Login Form 22

Milestone 3: Loading Screen 22

Milestone4/6: Camera and Pipeline 22

Milestone 7/8: Character Comprehension Display 23

Milestone 9: Adding Credentials 24

Milestone 10: Delete All Credentials 24

Milestone 11: Change User Password 24

Milestone 12: Export File 25

Milestone 13: Pause Feed 25

Milestone 14: Save File 25

C. Developing the coded solution (“The development story”) 26

Milestone 0 – Main Python 26

Milestone 1 – Creating the login form GUI 27

Milestone 2 – Authorising user login 31

Milestone 3 – Loading screen implementation 36

Milestone 4 – Displaying camera view 37

Milestone 5 – Navigation Menu 40

Milestone 6 – Displaying Pipeline 41

Milestone 7 – Character Comprehension 44

Milestone 8 – Character Display 51

Milestone 9 – Adding Credentials 53

Milestone 10 – Delete All Credentials 57

Milestone 11 – Change User Password 60

Milestone 12 – Export File 64

Milestone 13 – Pause Feed 67

Milestone 14 – Save File 68

D. Evaluation 70

Post Development Testing 70

Milestone 1/2: Login Form 70

Milestone 3 – Loading Screen 71

Milestone 4/6 – Camera View and Pipeline 71

Milestone 7/8 – Character Comprehension/Display 72

Milestone 9 – Adding Credentials 73

Milestone 10 – Delete All Credentials 74

Milestone 11 – Change User Password 75

Milestone 12 – Export File 76

Milestone 13 – Pause Feed 77

Milestone 14 – Save File 77

Evaluating The Program 78

Limitations 78

Final Evaluation 79

Project Appendixes 84

Main.py 84

Loginform.py 88

Loadscreen.py 90

Hashingalgorithm.py 91

Iterateauthorise.py 91

Menubar.py 92

Filemenucommands.py 93

Usermenucommands.py 94

Toolbarmovement.py 99

SQLCreation.py 99

BSLDetectionmodel.py 100

CNNModelcreation.py 102

A. ANALYSIS

PROBLEM IDENTIFICATION

Mental Health and unemployment rates are much greater in those with loss of hearing or deafness compared to the general population, up to 25% worse mental health and 14% worse employment rates. Amongst the population of the UK 11 million are deaf or hard of hearing and from that statistic 151,000 British Sign Language users across the UK. Expanding work with Computer Vision and machine learning to create a program which is able to recognise the characters will increase overall mental health within the hearing loss community and if the program succeeds, might even lead to better employment opportunities. The program will allow for much easier communication between BSL and non-BSL users since a translator will not be required.

The program will be user-friendly, displaying a camera view as to see the actions currently being performed and an on-screen typed version of the signed character. The following stages will consist of learning about detection using Python and how to create a recognizable gesture/hand position.
This project is suitable for computational methods for two main reasons. The processing of data with a machine learning project is extremely large, big data processing is essentially solely completed by computational approaches due to the sheer volume of data. Pattern recognition in large volumes of data for humans is not viable either since we can't keep a hold of all the information. My programs main features rely on the pattern recognition of hand positions and the processing of this large volume of data therefore the program is essentially made to be solved using computational methods.

STAKEHOLDERS

The program will be deployed across multiple locations and is supposed to ease communication between non-BSL and BSL communicators. It will not require an elevated level of technology understanding however it might frustrate some users due to the need for sign recognition, meaning that will have to slow down.
My main stakeholder within this project will be sign users of any age. They will utilise the program for communication with non BSL users, not needing to rely on a translator following them around throughout the day. The program will provide these users with independence and a better overall experiences of establishments where my project will be present. As important as it is that the program is fully functional and helpful at the end of testing, they will also be able to provide criticism to which areas of the program have to be improved so that User Experience is better and so that problems do not occur when someone does not have technical experience.
The secondary stakeholder that is mentioned is the companies/ individuals who will be communicating with the primary stakeholders. For them, this program must be accurate and responsive since it could mean the difference between communicating with the client and having to defer them. They must also be able to read back on conversations in case any information was missed.

RESEARCH

EXISTING SOLUTIONS

GNOSYS

An app created by a Netherlands start-up company. It was dubbed as the Google translator for the deaf and mute and works by placing a smartphone camera in front of the user while they gesture, and it translates to speech or text. However, it is reversible as well allowing for text to sign communication between individuals as well.

The app named GnoSys uses neural networks and computer vision to recognise video, it then uses complex algorithms to translate it into speech accurately. The solution is well-welcomed since it provides an affordable and convenient interpreter service which is in huge demand within the deaf community.

Similar to my solution, it aims to tackle problems faced in local business and communities around the world when providing services for deaf people. The new application can be found within the business structure as well, paving a new path for businesses who want to employ deaf and mute employees. They can use It to convey employee messages to the end consumer. It will drive inclusivity and communication barriers.

This solution was meant for the Indian market, as of now solutions for this are not widely available within the UK.
The app is of low requirements, with its only requirement being a phone camera. It is mostly software based and relies on their algorithms for all the complex calculations.

HANDTALK

Hand talk is the largest platform for automatic translation in Sign Language. This text-to-sign converter uses virtual avatars to break communication barriers between deaf and hearing people, not only in north America with their ASL translator but in parts of South America with the LIBRAS translation as well. It is based in the Americas and was voted the best social app by the United Nations in 2013.

This app is an instantaneous translation app that can be used for both communication and education of sign language. When creating the solutions, they had to think ahead, providing their virtual assistants with defined facial expressions and large hands.

This solution is slightly different to my project however the basis of the solution is similar, providing accessibility for deaf people in a certain part of the world.

FEATURES OF THE PROPOSED SOLUTION

Based on the concluded research, it would be in my best interest to complete a computational solution that includes separate TensorFlow models for each sign. The prior solution of determining sign by relative positions of nodes is too complex and sets an unrealistic standard for the timeframe given. By using TensorFlow models you can quickly train and identify signs instantaneously within the program, since we will be outputting each character to the screen it would ideally have to be immediately displayed so that both end users will be satisfied with the program. The proposed solutions will also include a secure login system which authenticates via SQL database information. Requirements for this come from that fact that sensitive user information may be stored by the program, since all signed values and characters will be displayed. Adding an extra layer of security means that only the correct users can access this information. Finally, all the data that the program stores will be able to be exported via csv. This export can be used to renew one's memory of what was previously said or even imported back into the program so that changes can be made.

If the model solution becomes too difficult leading me past the time deadline that I have, I will utilise a heuristic approach. This approach will use the relative position of the hands in frame and the distance between nodes. This method breaks down into a mathematically intense method which suits a computational solution as well.

LIMITATIONS OF DEVELOPMENT

Since the project will be using deep learning approximation on a mid-resolution webcam a degree of inaccuracy will be expected. Due to time constraints, I will have to limit myself on the quantity of gestures I will be able to produce, opting for commonly used gestures so that the program can still be used effectively by the end-user. Furthermore, the use of Tkinter means that the GUI might be slightly less modernised than expected. Use of colour schemes and pictures will improve the aspect of it however it will never look completely modern.

PROJECT REQUIREMENTS – HARDWARE AND SOFTWARE

During the course of my Project, I will be using Atom, GitHub own open-source development platform. I have opted for this since it is extremely customisable. The base program does not function as an IDE however by adding certain chosen packages I can create development platform which works will for me and will make the process of development easier. Furthermore, Atom provides an in-built GitHub tab which allows me to view the project repository and commit current development versions to the branch directly from the program. For my GUI, I will be using Python's Tkinter library. This is a popular and repeatedly assessed GUI library which meets all the development requirements and will allow for user interaction within the program. OpenCV is Pythons Computer Vision library and will allow me to access the webcam used for the program.

Of Course, a webcam will be necessary to detect motion via video along with a Mouse to navigate the program.

The nature of the program lead to me to believe that the code will not be able to run on all kinds of systems. A machine learning project such as this requires a vast amount of data processing especially since it won't simple be a camera view but rather a camera view built around a fully functioning program with database integration. As a developer I will try my best to reduce the requirements for this program however you will more than likely require a desktop computer or a upper-mid end laptop to be able to run this program.

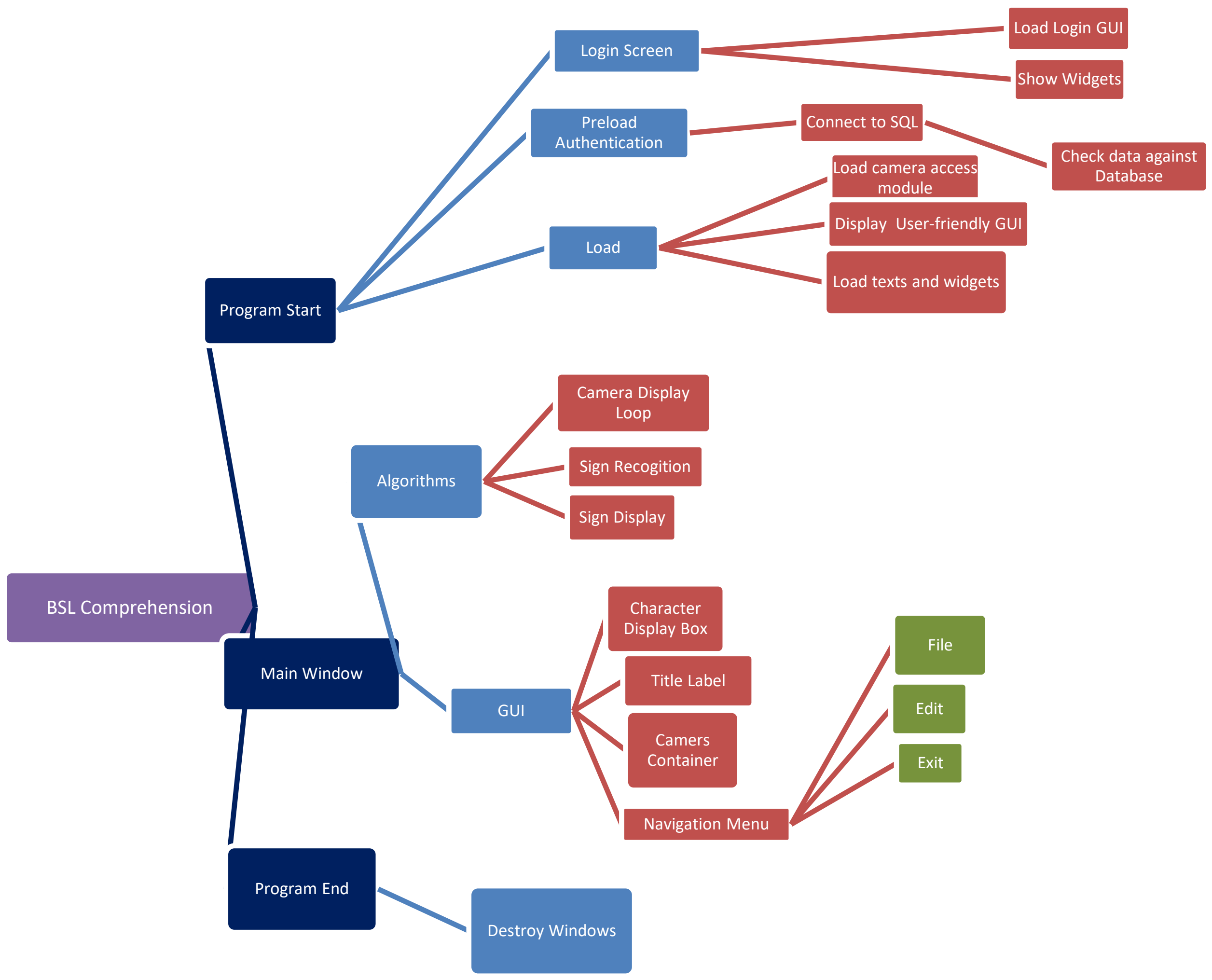
MILESTONES

No.	Milestones	Justification	Evidence Location
1	Login Form GUI	Create a login form GUI which attracts user since this will be the first piece of the program that they will view	Page 27
2	Login Form Authorisation	Create an authorisation algorithm which will hold user credentials in an SQL database. Values entered into the login form GUI will be compared to these, denying, or accepting user access.	Page 31
3	Loading Screen	Create a loading screen that buffers between the closing of the login form and loading of the main file.	Page 36
4	Display Camera	Display a live camera feed within a Tkinter GUI so that the user can see themselves when signing characters later in the program.	Page 37
5	Navigation Menu	Create a navigation menu that allows users to access program handling features that will come later in the development.	Page 40
6	Display Pipeline	Use Googles media pipe hand model to detect and display a pipeline on visible hands in the camera live feed. This will be visible to users.	Page 41
7	Character Comprehension	Create a method of detecting Sign Language characters using a model or heuristic approaches.	Page 44
8	Character Display	Display an array of signed characters from the character comprehension milestone within a Tkinter label visible to the user.	Page 51
9	Adding credentials	Create a sister window which contains a separate GUI allowing the user to add a new user and password to login screen.	Page 53
10	Delete all credentials	Allow users to remove outdated credentials	Page 57
11	Change Password	Allow users to change the password connected to a specific username	Page 60
12	Export File	Allow users to save displayed characters into different file formats	Page 64
13	Pause Feed	Users should be able to pause and continue the visible live feed in the main program. The buttons for this process should be clearly visible in the main program for ease of use.	Page 67

14	Save	Save contents of the translation bar so that it is there when the program is reloaded.	Page 68

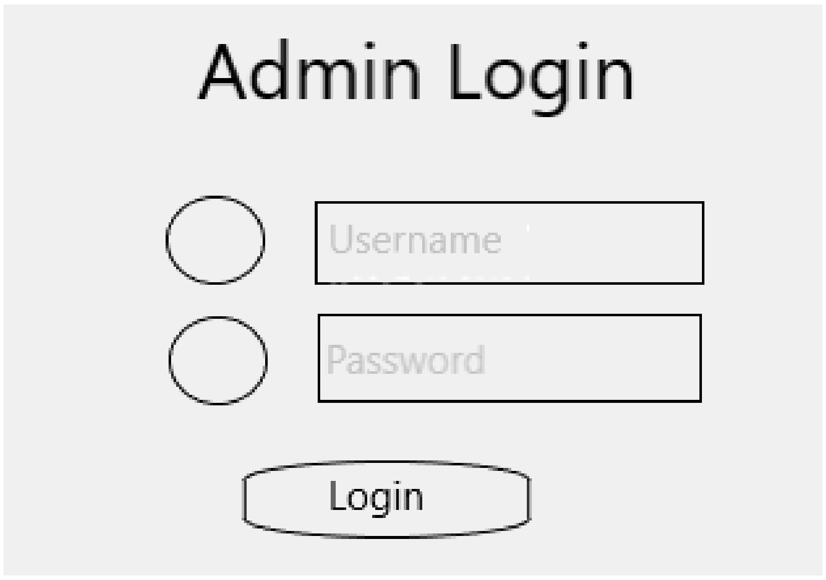
B. DESIGN

SYSTEMS DIAGRAM



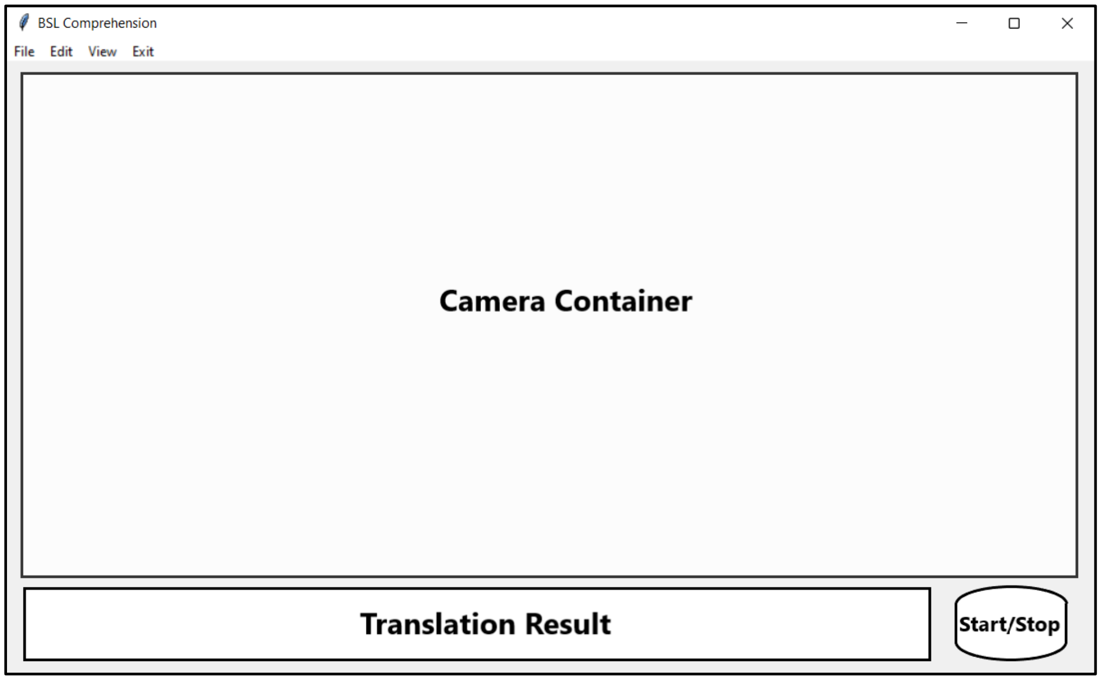
PROJECT DESIGN

LOGIN FORM



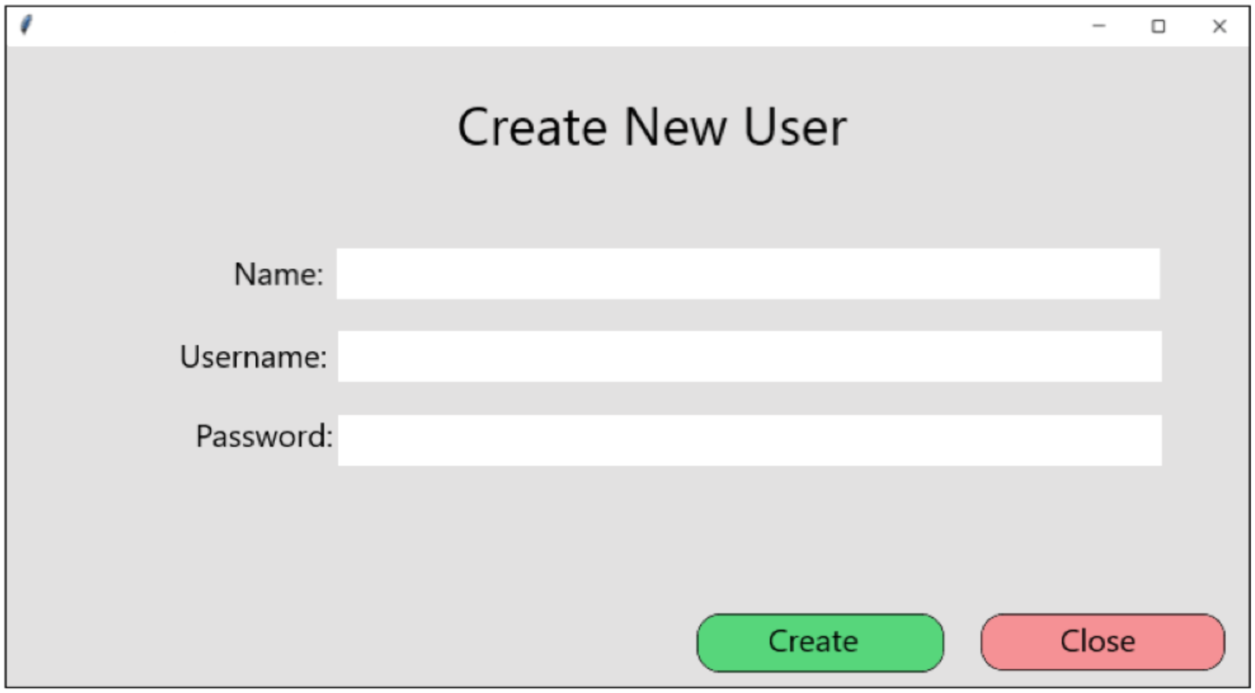
The concept design shows the Login Window for the BSL comprehension program. Username and Password widgets will take user credential inputs and store them within a Tkinter variable. Content within must be able to be exported since not only will the password have to be hashed due to security reasons, but it must be compared to a Database for authentication. The Login button will begin the authentication algorithm when pressed and will return a failed message if incorrect. The rest of the window is mostly visual additions to make the program user friendly.

MAIN WINDOW



The concept design shows the main window of my program in its most basic form. Majority of the root window is taken up by the Camera Container, this widget will house the frames transmitted by the video feed. This container must always function correctly as it will be vital for user interaction with the program. Below the camera container is the translation result textbox. All signed characters and words will be displayed underneath since it is convenient for the end-user and allows for efficient communication, all displayed results will be transferred to a separate file which can later be accessed so it must have export capabilities as well. To the side lies the Start/Stop button for display within the camera container. The button works as a live feed blocker meaning that the program doesn't have to be closed and opened between uses and the detection can just be paused while the non-BSL user is communicating. Finally, an ease of access menu will also be incorporated. This menu will store the export file function for the translation result and even an import function so that users can add to previously edited files. The menu will also include the basic must haves within a program such as the close dropdown, help dropdown and an edit dropdown for overwriting login credentials.

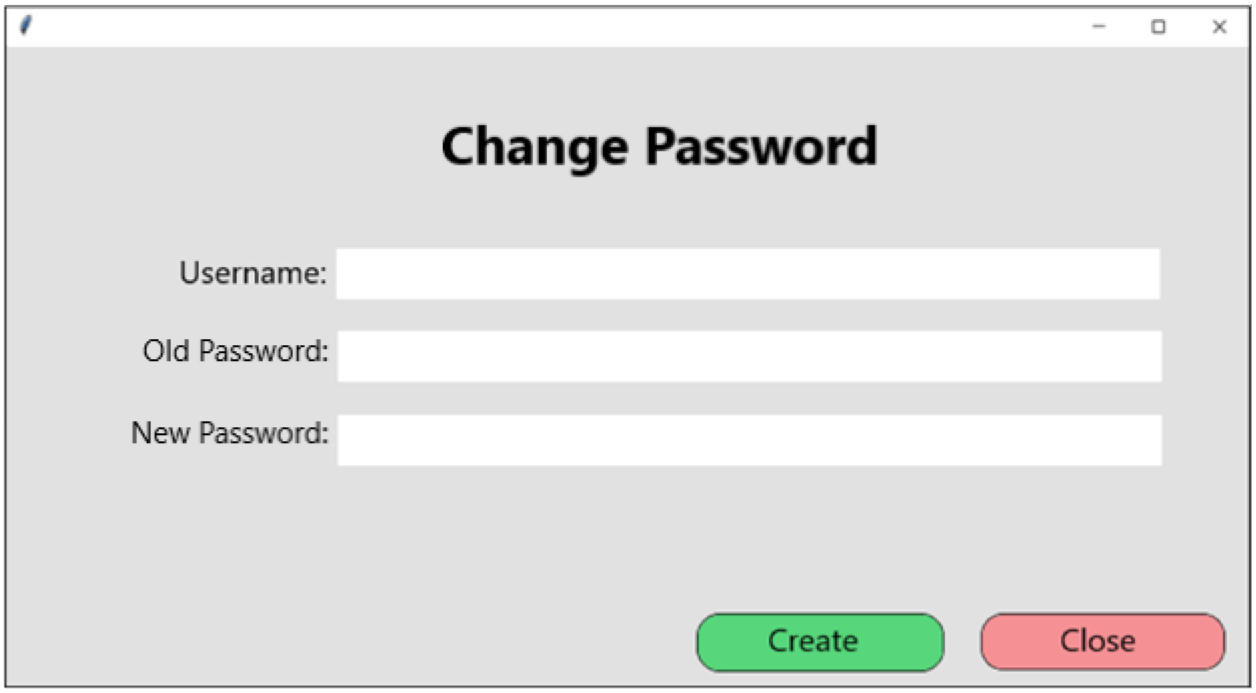
NEW-USER WINDOW


Create New User

Name:	
Username:	
Password:	

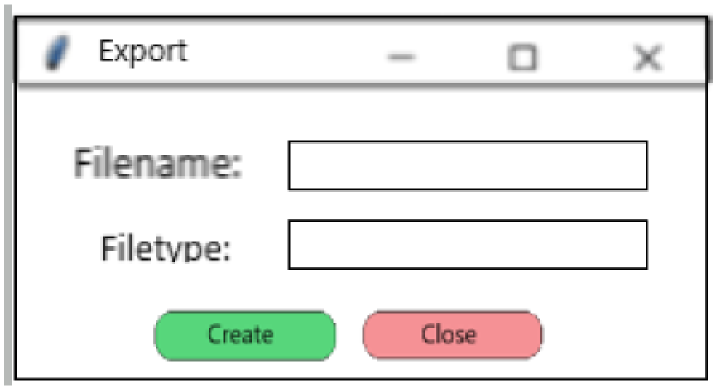
The new-user window will allow users of the program to create login credentials for the required people. This feature makes the program more secure since people can't access potentially sensitive information within files. Like the login form, it includes necessary information such as username and password but also stores the users. Potentially, this window could have more information boxes for improved features in the program. Two buttons are included at the bottom which give the user a choice between closing the window and going back to the root or creating a new account which would log you out and expect you to use the new credentials.

CHANGE PASSWORD WINDOW



The change password window will allow users to change enrolled user's password credential. It has three entry inputs; the username entry is used to search for the record since this piece of data must be unique. The old password is compared the searched record and if it matches, I can assume that the user is the owner of the account. If both these check out, then the users new password will replace the old password. Like in the rest of the GUI windows in the program there are two options at the bottom. One to opt out of the module execution and one to confirm the change in the user's password.

EXPORT WINDOW



The export window will allow users to export the contents signed into the translation bar into a separate file, the file will have selectable file types to save.

The filename entry will contain the filename and filetype the file extension. Again, the user has the choice to create the export file or close the module.

INPUT VALIDATION

LOGIN INPUTS

USERNAME ENTRY
Format Check: Data must be confined to string values and cannot be numerical. Must be unique.
Length Check: Dataset must be no shorter than 3 characters
Presence check: A piece of data must be entered into this input. It must be entered before continuing.
Check String: The piece of data must match a pre-exiting table to proceed to next stage.

PASSWORD ENTRY
Format Check: Data consists of at least one symbol and one integer.
Length Check: Dataset should consist of at least 8 characters for security purposes.
Presence check: Field must be entered for validation, cannot proceed without it.
Check String: Data will be checked against pre-existing table for authorization.

NEW-USER INPUTS

NAME ENTRY
Format Check: Data must be string value since names cannot include numerical values. Names must also be split by spaces to allow for sub-stringing later, format JOHN SMITH.
Length Check: Since names can variable there is no specific required length
Presence Check: Data must not be left empty as it is a requirement
Range Check: I'm limiting the range of names to 5 since this is likely to be the upper limit of many users

EMAIL ENTRY
Format Check: The data must be solely as string apart from one @ and a full stop.
Length Check: Emails vary in length so I can't set a boundary for the length
Presence Check: An email is required to create a user account
Check String: An account registered with the email can't be created

USERNAME ENTRY
Format Check: Data must be confined to string values and cannot be numerical. Must be unique.
Length check: Data must be no shorter than 3 characters
Presence check: A piece of data must be entered

PASSWORD ENTRY

Format Check: Data consists of at least on symbol and one integer value
Length Check: Dataset should be at least 8 characters in length.
Presence Check: A piece of data must be entered

CHANGE PASSWORD

USERNAME ENTRY
Format Check: Data must be confined to string values and cannot be numerical. Must be unique.
Length Check: Data must be no shorter than 3 characters
Presence Check: Data is a requirement
Check String: The username string will be checked against preexisting usernames in database

OLD PASSWORD ENTRY
Format Check: Data consists of at least on symbol and one integer value
Length Check: Dataset should be at least 8 characters in length.
Presence Check: Data is a requirement.
Check String: The Old Password string will be checked against database values to search for record

NEW PASSWORD ENTRY
Format Check: Data consists of at least on symbol and one integer value
Length Check: Dataset should be at least 8 characters in length.
Presence Check: Data is a requirement.

MODULES AND ALGORITHMS

HASHING ALGORITHM

Once the user has entered the password and pressed the enter button on the login form, the password should be hashed. Visible passwords should not be available on the SQL table that I will use later in the program for safety purposes. This algorithm should not affect the way that the user interacts with the input in any way.

For encoding of the password, I will be using the base64 python library which I have researched and found to be the most reliable and well-known encrypting library. The module will take in both the password and username provided as parameters, the username provided by the user will determine the key used to encrypt the password making it reliant on the username and therefore providing the username with primary key status for later use. As to not make the encryption an entity with simple decryption, I run through the hashing process multiple times using different base64 objects. A list will store the data of all the hashed value which will be passed forward to the authorization module.

1 import base64	1 Imports base64 python library
2 def hash (key, password):	2 Defines hash function with two parameters, key and password
3 hash= []	3 Hash list is created
4 for n in range(length(password)):	4 Count-controlled loop that iterates dependent on password string length
5 hashkey=key (n % length(key))	5 Produces a hash key from the length of password and MOD of the key. The hash key is a single character, changing the size of the password by one will increment the hashkey one string character to the right.
6 hash.append(chr ((ord(password[n]) +ord(hashkey)) % 256))	6 Each character in the password will produce a 7-character hash while iterating. This line creates and appends all the values into the hash list.
7 return base64. b64encode (“”. join(hash).encode())	7 Hash list is converted from string values into byte values that b64 uses to encode. Password hash is returned from the function

ITERATIVE SEARCH

Before the program can authorize user details it must fetch the values stored in the tables in SQL. For this I’m going to use an SQL cursor that iterates through each table.

First ill import the sqlite3 python library which enables communication between SQL databases and the program. The search will connect to the database and separately iterate through each table with an SQL cursor storing the necessary values within lists which will be returned for use in the authorisation module.

1 import sqlite3	1 Import sqlite3 python library
2 def iterativeSearch ():	2 Define the iterativeSearch function
3 DBusername= []	3 Create DBusername stack-list
4 DBpassword= []	4 Create DBpassword stack-list
5 connection= sqlite3.connect(‘databasename’)	5 Establish a connection between the program and the SQL database using sqlite3 connect object
6 cursor = connection.cursor()	6 Define a cursor with previously established connection so that data can be searched and transferred
7 cursor.execute(‘SELECT username from usernametable’)	7 Execute commands for the cursor to partake in. Using SQL, the cursor will execute the selection of all the record in the username field from the username table
8 DBusername = cursor.fetchall()	8 The cursor will fetch all the selected information and store it within the DBusername list we previously defined
9 cursor.execute(‘SELECT password from passwordtable’)	9 Execute commands for the cursor to partake in. Using SQL, the cursor will execute the selection of all the record in the password field from the password table
10 DBpassword = cursor.fetchall()	10 The cursor will fetch all the selected information and store it within the DBpassword list we previously defined
11 connection.close()	
12 return(DBusername, DBpassword)	

	11 Close established connection between program and sqlite3 database 12 Return the stored table values within the independent lists. Return value will be a 2-dimensional list

AUTHORISATION

This module is necessary for the functioning of the login form. Authorise will take in the widget values for username and password and compare them to the values previously fetched in the iteration module.

The module will contain the values returned from iterative search in a two-lane stack. Using a count-controlled loop dependent on the length of the first lane, I pop top-level values from each lane and place them into variables with appropriate names. Finally, these variables are compared to the values passed by the previously mentioned hashing algorithm. If they match the program returns True and the login form is destroyed, and the translation window begins loading however if an inconsistency is present the module will return False and a Tkinter message box error pop-up will be seen by the user.

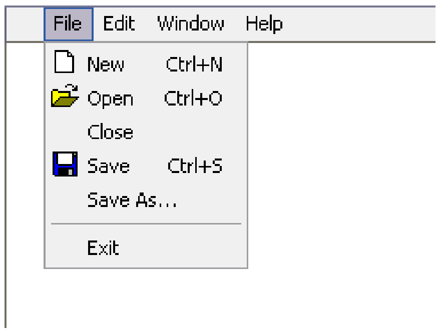
```
1 def authorise(username, password):
2     n=iterativeSearch()
3     for i in range(length(n[0])):
4         usernameSQL = list(n[0][i]).pop()
5         passwordSQL = list(n[1][i]).pop()
6         if usernameSQL == username:
7             if passwordSQL == password:
8                 return True
9         else:
10            return False
```

1 Define authorisation function which takes in username and password parameters from main.py
2 Call iterativeSearch function and assign the returned SQL table values to the n tuple
3 Iterate through a loop n quantity of times
4 Turn the 1st dimension of the tuple into a list-stack and pop the i value from n. All values iterated through are usernames placed into the usernameSQL list.
5 Turn the 2nd dimension of the tuple into a list-stack and pop the i value from n. All values iterated through are hashed passwords placed into the passwordSQL list
6 Compare values of usernameSQL and username
7 Compare values of passwordSQL and password
8 If both the usernames and the password match the credentials stored in the SQL database then a True value will be returned
9-10 If one of the values doesn't match then return a False value.

MENU

The program will contain a quick access menu anchored to the top similar to many other familiar programs. It will contain necessary commands for the functioning of the program such as providing the ability to create a new user, import and export files.

This menu will be created using the Tkinter menu widget. This widget uses the styling format of the default operating system and will give me functionality like what you would see in a default window.



1 import tkinter as tk 2 def menu(root): 3 mainwindow_menu=tk.Menu(root) 4 root.config(menu=mainwindow_menu) 5 file_dropdown=tk.Menu(mainwindow_menu, tearoff=False) 6 mainwindow_menu.add_cascade(label='File', menu=file_dropdown) 7 file_dropdown.add_command(label='New') 8 file_dropdown.add_command(label='Open') 9 file_dropdown.add_separator()	1 I import the Tkinter library 2 I define the menu which will take in the root as a parameter 3 I define a new menu widget which has root as its master 4 I configure this new menu widget as the root menu 5 I will create a file drop down for the menu which will be repeated for all other drop downs. 6 the file drop down was defined as a menu widget I now add it as a cascade on the main window menu 7 add a new command labelled new 8 add a new command labelled open 9 add a separatot to the menu

MENU COMMANDS

Every drop down of the program menu will have executable modules, each serving different purposes that will require and make-up much of the code going into the program such as save function for the program, and the user editing capabilities. For decomposition purposes I have split this task into manageable chunks some of which will be covered below.

FILE MENU COMMANDS :

1 def saveAs(root, text): 2 def createfile(): 3 file = open(f'{filename.get()}{filetype.get()},'w') 4 file.write(text.get()) 5 text.set(value="") 6 saveAsform.destroy() 7 "Main Chunk of repeated Tkinter code explained in previous sections" 8 options = ['.txt', '.docx', '.pdf'] 9 filetypeOption = tk.OptionMenu(saveAsform, filetype, *options)	1 defines the SaveAs function which takes in the root master and main program text to save as parameters 2 defines createfile function, code within the function will create a new file in system 3 opens or creates a file (if not exists) and uses the Tkinter filename variable and Tkinter filetype variable entered by the user 4 writes the contents of the main programs text into the newly created or opened file 5 clears the text element of the main program so that the suer can carry on using the program without having to delete previously written text 6 destroys the SaveAsform and returns to the main program so that the user can continue 7 There is a large section of code that simply defines the GUI layout and appearance along with some variables 8 defines the available filetype options in a list

	9 creates a Tkinter option menu, using the option list the user will get given a discrete choices whose data will be store sin the filetype variable used in the createfile function

USER MENU COMMANDS:

1 def newuser(root): 2 def newuserSQLinput(): 3 newpassword.set(hash(newusername.get(), newpassword.get()) 4 newnamelist=newname.get().split() 5 connection=sqlite3.connect("database") 6 connection.execute("INSERT INTO username VALUES(:firstname, :lastname, :username)" {"firstname": newnamelist[0], "lastname": newnamelist[1], "username": newusername.get()}) 7 connection.execute("INSERT INTO password VALUES(:firstname, :lastname, :password)" {"firstname": newnamelist[0], "lastname": newnamelist[1], "password": newpassword.get()}) 8 connection.commit() 9 connection.close()	1 I define the function which will handle all new user activities. It takes in the root window as a parameter so that a GUI can be created from the main program within the function. 2 I define the function which will perform the SQL transfer of new data. 3 Since all Tkinter variables are enclosed inside classes must use the set and get methods to get the raw stores values of the variable. I set the new password from the form as the hashed value of the inputted new password. The hash module was created earlier and provides security to the suer when transferring password over a connection. 4 I use the python split method to create a list of names input by the users which separate at the spaces input by the user. 5 I create a connection with the sqlite3 database which currently exists, this database holds the records for username and passwords of the program. 6 I execute an SQL command to the username table in the database. I insert the values of the Tkinter variables into the respective fields, using location 0 of newnamelist as the first name and location 1 as the surname of the user. The password does not have to be accessed in this table because this solely holds the records for username. 7 I execute an SQL command to the password table in the database. I proceed to execute the same SQL command as I did for the username table however this time instead of getting the username value, I get the password value to input into the table. 8 I commit all the changes that I have made to the SQL tables, essentially saving the changes. 9 I close the connection between the SQL database and the program.

SHUTTER FRAME

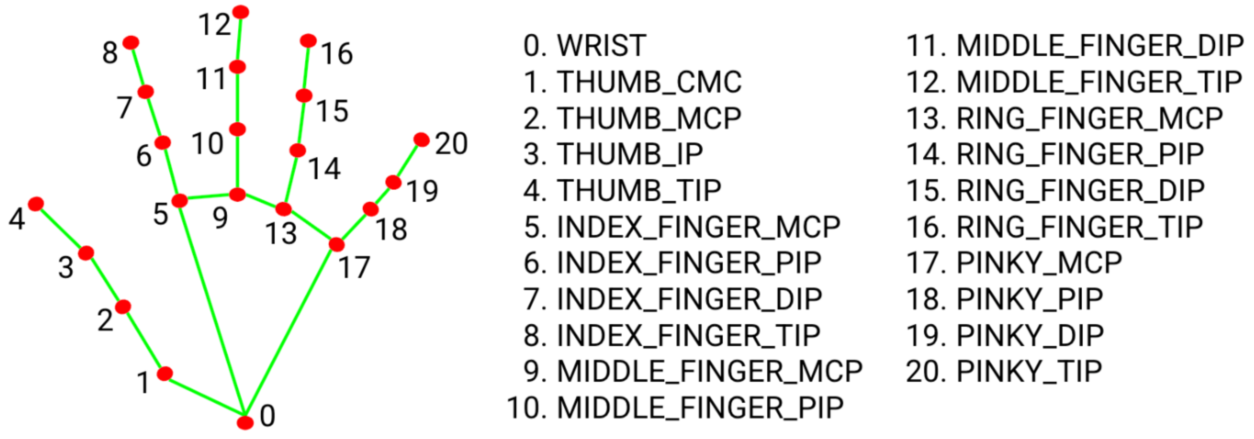
Shutter Frame will allow the user to pause and continue the Camera Container video feed. Users should be able to interact with other aspects of the program while the machine learning algorithm isn’t processing. This algorithm will use a ghost widget which hasn’t been packed onto the main frame and a Boolean Tkinter variable. The Boolean variable will store a True value when the Camera Container is processing live feed however when the shutter button is pressed by the user, it will activate the shutter function. The function will check the Boolean value and proceed to unpack the camera container so that it is no longer viewable to the user, the ghost widget, camera cover, will be packed and replace the camera container slot within the layout of the main window. It will state that the live feed has been paused as per request of the user.

1 def shutter():	1 Define shutter function
2 if booleanshutter.get() == True:	2 If the Tkinter variable booleanshutter is true proceed
3 cameracontainer.grid_forget()	3 Un-Grid the cameracontainer from main.py
4 cameracover.grid()	4 Grid the cameracover in position of camera
5 booleanshutter.set(value=False)	5 Set the booleanshutter value to False for next pause/continue cycle
6 else:	6 If booleanshutter value doesn't equal True
7 cameracover.grid_forget()	7 Un-grid cameracover widget
8 cameracontainer.grid()	8 Grid cameracontainer along with camera feed
9 booleanshutter.set(value=True)	9 Set the booleanshutter value to True for next pause/continue cycle.

SHOW FRAME – PIPELINE DISPLAY

By using the pre-built media pipe library in Python, we can display landmarks on the user's hands so that the visual tracking of position is easier. By using OpenCV we can read captures and convert to CV2 BGR, this is an essential step because the media pipe object creates by machine learning models independently learnt hand tracking using black and white images. If an RGB image was to be used, the object would not be tracked correctly or at all.

I process the CV2image within the hands.process() function that media pipe contains. Then we check for the presence of a hand and return a Boolean value. If the value is True, I begin a count-controlled loop that iterates through and places landmarks on top of the media pipe CV2image. By using the .HAND_CONNECTIONS we create a graphing atop the hand image which can then be passed through to the rest of the program.



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

1 mphands = mediapipe.solutions.hands 2 hands=mphands.Hands() 3 mediapipe = mediapipe.solution.drawing_utils	1 Importing the mediapipe hands solution, one of many classes of models for machine learning solutions created by google

	2 I create a hand detection model from the solutions I just imported. It will allow me to detect hands on the live feed. 3 The mediapipe drawing utilities solution will allow me to draw the visual node and edges of the hands which the model is detecting.

1 def show_frame(): 2 success, frame = capture.read() 3 frame=flip(frame, 1) 4 cv2image = cvtColor(frame, color_bgr2rgb) 5 display=hands.process(cv2image) 6 if display.multi_hand_landmarks: 7 for handlms in display.multi_hand_landmarks: 8 mediapipeDraw.draw_landmarks (cv2image, handlms, mphands.hand_conn)	1 I define the show frame function which will contain the pipeline display 2 I capture a frame from the live feed and place the return values in two variables. The frame variable stores the image and the success stores whether the capture has been successful or not. 3 I flip the frame so that the image is displayed un-mirrored. 4 The image converts the RGB image into a grayscale image that the hand model can process. 5 I process the image from the frame and put it in the display variable 6 I use an if loop to detect whether there any hands present 7 I iterate through the number of landmarks detected on the hands or nodes 8 I use the drawing utility's to draw nodes and edges on the displayed hand.

SHOW FRAME – PHOTOIMAGE RETURN

After I extract all necessary data from the camera frames and overlay what the user will see, I must return the frame to be used within the camera container. Tkinter label widgets don't support certain file types so I must import the PIL library to counteract this.

I take the image produced by OpenCv and convert it to a PIL readable image using an in-built function. I will proceed to convert this compatible image into a format that can be displayed within the Tkinter camera container, this image will be configured to show within the frame after the formatting has been completed. This function must be recursive in order that video frames can be updated alongside the running of the program, using the Tkinter after function we can call show_frame from within show_frame and set the frame speed by deciding upon how fast the n+1 function call occurs after the n call.

1 image= Image.fromarray(cv2image) 2 tkimage=ImageTk.PhotoImage(image=image) 3 cameracontainer.imgtk =tkimage 4 cameracontainer.configure(image=tkimage) 5 camercontainer.after(1, show_frame)	1 I convert the cv2 capture from the live feed into an array of components. This is because it has to be an array image to be inputted into the label widget 2 The image is currently a numpy array however to be compatible with tk it must be a photimage array object. I use the photoimage library to convert the numpy array into a PIL Photo Image 3 I set the image parameter of the caemracontainer label to the newly created Photoimage array.

	4 Using the label configure method, I show the new image parameter on the label 5 After 1 ms the function repeats creating a recursive loop and therefore a refresh rate of the frames on the label widget.

TEST DATA DURING DEVELOPMENT/ POST DEVELOPMENT

For my testing during development, I will only be stating the milestone that contain some kind of user input. Much of my project is interactive and based on events such as mouse clicking and camera activation, these types of milestones can't be tested with physical user inputs however I will include the information that they take in as inputs

MILESTONE 1/2: LOGIN FORM

Test Number	What is being tested	Output
1	Verify if use can login with the correct credentials. Inputs: (master, master) (user1, user1password)	The credentials should be accepted. Mismatched usernames and password should not authorise login.
2	Check what happens when fields are left blank with no inputs. Inputs: Not Applicable	Leaving either of the fields blank should return an error message box to the user stating that the field cannot be left blank.
3	Confirm error message provided when user enters incorrect credentials. Inputs: (errortest, errortest)	Entering the incorrect credentials should return a message box to the users stating that one or more of their inputs are incorrect.
4	Verify shortcut key binds. This includes the functionality of the arrow keys, enter key and escape button. Inputs: Key Presses from Up arrow, Down arrow, Enter and Escape keys	Pressing the up and down arrow keys should make the focus switch between entry widgets. The enter key should begin the authorisation function once clicks and the escape key should halt the program.

MILESTONE 3: LOADING SCREEN

Test Number	What is being tested	Output
1	Check to see if loading screen appears on program initialisation. Inputs: Not Applicable	the load screen should appear almost immediately following the user authorisation displaying a loading bar and time remaining
2	Displays the remaining loading time until main opens. Inputs: Not Applicable	Users should be able to see an estimate for time remaining on the loading screen bar.

MILESTONE4/6: CAMERA AND PIPELINE

Test Number	What is being tested	Output

1	Does The camera view loads instantly after main window starts up using a separate thread. Inputs: Camera Live Feed	Camera feed should load with a small delay, once it has loaded live feed will take up the top half of the program
2	Does the live feed produce minimal motion blur when a user moves suddenly. Inputs: Camera Live Feed	Live feed should only have a small amount of motion blur and the camera frame should not begin to stutter as movement speed increases
3	Does the media pipe pipeline only begin to display to the user when both hands appear on the camera feed. Inputs: Camera Live Feed, Hand Frames	The pipeline will not display when one hand is in the feed but once both hands are in the feed, visible nodes and edges will appear on the users' hands remaining in a fixed position on the hands even when movement occurs
4	Does the pipeline remain on the live feed even when user hands are overlapped and moves suddenly. Does exposure to light effect the display. Inputs: Camera Live Feed, Hand Frames	Overlapping hands will cause some problems but should be fixed by slow movement and light will heavily effect the display capabilities.

MILESTONE 7/8: CHARACTER COMPREHENSION DISPLAY

Test Number	What is being tested	Output
1	Does the model correctly translate the signed character of the user Inputs: Camera Feed	The model should be able to comprehend a limited quantity of characters in BSL with a varying accuracy depending on the character
2	Does model respond correctly from varying distances from the camera feed. Inputs: Camera Feed	The model should not have any trouble detecting from close distances or from medium distance however will struggle with detection from long ranges
3	Do illuminated spaces slow/Halt the comprehension program. Input: Camera Feed, Increased light exposure	The program should be able to detect even when higher levels of light exposure are present, this doesn't include exposure levels which change the visible screen such as directly towards a light source.
4	Can the model respond to fast hand movements by the user	The model should be able to handle movement which fall within the normal range of speed such as moving across the screen
5	Do the characters so when values are appended to the translation append list	The characters appended onto the translation append list should be shown in the program.
6	Can the user manually write within the translation bar, which they should not be able to do. Input: Random characters	The user should not be able to manually type into the translation bar. Any attempt at typing within the bar will wipe the typed characters.

MILESTONE 9: ADDING CREDENTIALS

Test Number	What is being tested	Output
1	Does the form move across the window when the toolbar is dragged with minimal error. Input: Click event on toolbar	Form will move across the window however might slightly drag behind if computer is already running intensive tasks.
2	Is an error message provided when one of the entry field holds the wrong format of data. Input: Name entry (John, JohnSmith, John Smith Junior, John Smith)	An error message should be provided if only first name is entered. Any name that contains a first and last name will be accepted.
3	Does the program provide an error if all the input field are left empty. Inputs: N/A	The program should provide an error message that doesn't allow a user to enter information while fields are empty.
4	Does the form provide a method of confirmation if the user credential input was successful. Input: N/A	A message box stating to the user that credentials storage has been successful should appear once the enter button event executes.
5	Does the exit method work correctly, exiting the new user form and returning to main. Input: Exit button press event	The form should close down once the exit button is pressed and should show the main window.

MILESTONE 10: DELETE ALL CREDENTIALS

Test Number	What is being tested	Output
1	Does the event provide you with a confirmation message box in case a mis click occurred. Input: N/A	Once the delete all user's dropdown command has been pressed a message box should appear on the top level confirming whether you want to go through with the current action.
2	Is the exit/no button working correctly for cancelation of the function. Input: N/A	The no button press on the confirmation message box will close down the box
3	Is a confirmation message given to the user once the program yes button has been pressed. Input: N/A	Pressing the yes button will close the message prompt for the user and open another stating that all data has been erased

MILESTONE 11: CHANGE USER PASSWORD

Test Number	What is being tested	Output
1	Does the drag function with the toolbar work correctly and smoothly. Input: N/A	Form will move across the window however might slightly drag behind if computer is already running intensive tasks.

2	Is an error message shown if the username or password fields are left blank. Does this error message show when all fields are left blank. Input: One input is correct, master for both	An error message should show if the username/password field is left blank. This same error is shown if all fields are left empty.
3	Is a confirmation message shown when the change button is pressed, and the credentials are changed. Input: N/A	A confirmation message box is shown once the change button is pressed stating to the user that their credential change has been successful

MILESTONE 12: EXPORT FILE

Test Number	What is being tested	Output
1	Does an error message prompt appear if the filename entry is left blank. Input: N/A	A Tkinter message box will appear to the user if the filename entry is left blank with no input
2	Can you select all the different file extensions with seamless change between them. Input: Click Event	Selection of file extensions is smooth and will not cause issues for the user.
3	Does a confirmation message box prompt appear once the file create button has been pressed. Input: N/A	Once an input has been added to the filename entry and the confirm button has been pressed a message box prompt will appear confirming the creation of the export file into the user's system.

MILESTONE 13: PAUSE FEED

Test Number	What is being tested	Output
1	Is feed pausing responsive and able to switch without deforming the window. Input: Click Event	The camera cover widget should appear instantly after pressing the shutter button. The dimension of the shutter will cover the location which the camera container covered perfectly providing a full blackout effect.
2	Does pressing the Shutter button cause a Pause and Continue of the live feed or does it remain paused. Input: Click Event	Pressing the Shutter button for the first time will cause a Pause cycle and pressing it again will cause a continue cycle. This repeats with every button press depending on the current cycle which it is in.
3	Can a window be turned on and off repeatedly without causing the program to break or even crash. Input: Click Event	Pressing the Shutter button repeatedly should not crash the program. The user should be able to see the shutter stages clearly.

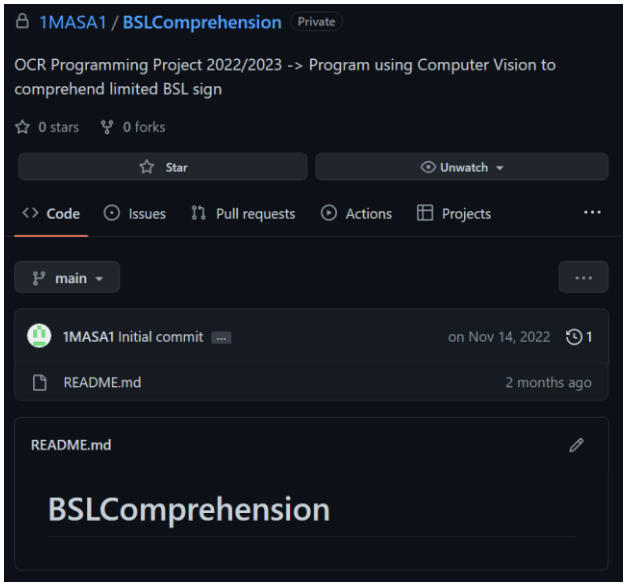
MILESTONE 14: SAVE FILE

Test Number	What is being tested	Output
1	Does the save command have a prevention method for when the input data is empty. Input: N/A	A message box prompt will appear when the translation bar contents are empty stating to the user that saving a file with no contents is not possible.
2	Is a message provided to the user when the save has been executed successfully. Input: N/A	A message prompt will be shown to the user once the save command is executed stating to the user that the contents have been saved successfully.
3	Do the contents of text parameter transfer into the save file. Input: text parameter(contents)	The contents should be visible when opening the save file in the main program directory.
4	Do the contents of the save file show on the translation bar when the program is reloaded. Input: N/A	The contents of the save file should appear in the translation bar when the program is reloaded so that the user can carry on from where they left off.

C. DEVELOPING THE CODED SOLUTION (“THE DEVELOPMENT STORY”)

MILESTONE 0 – MAIN PYTHON

Objective of the milestone: Start a main file to push to git repository


1MASA1 / BSLComprehension Private

OCR Programming Project 2022/2023 -> Program using Computer Vision to comprehend limited BSL sign

0 stars 0 forks

1MASA1 Initial commit on Nov 14, 2022 1

README.md 2 months ago

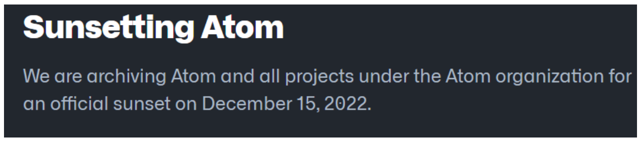
README.md

BSLComprehension

The main file will be used to support the main bulk of the program. Many of the algorithms will be imported as modules so that they can be used by the main window.

Fixes:

While in the process of development Atom announced that they would be ‘Sunsetting’


Sunsetting Atom

We are archiving Atom and all projects under the Atom organization for an official sunset on December 15, 2022.

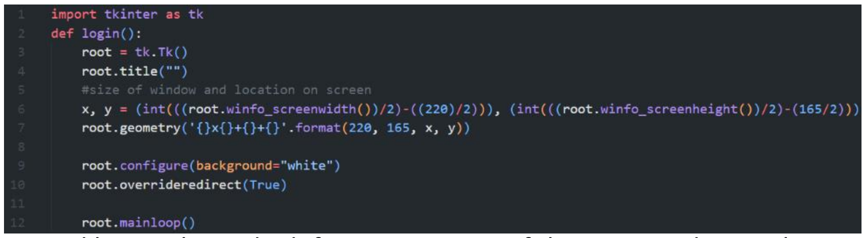
This change had forced me to look for a viable replacement for Atom. My decision came down to Sublime text and VS Code, I finally decided to go with VS code due to its ease of integration with Git (like Atom) and its popularity in the development community. This development platform was more likely to provide me with the support I needed and the IDE-like environment that I was looking for.

Now that the main file is pushed and ready to go, I can begin working on project milestones.

MILESTONE 1 – CREATING THE LOGIN FORM GUI

Objective of the milestone: Create an interactable GUI which allows the users to input valid login information

Creating the login form root



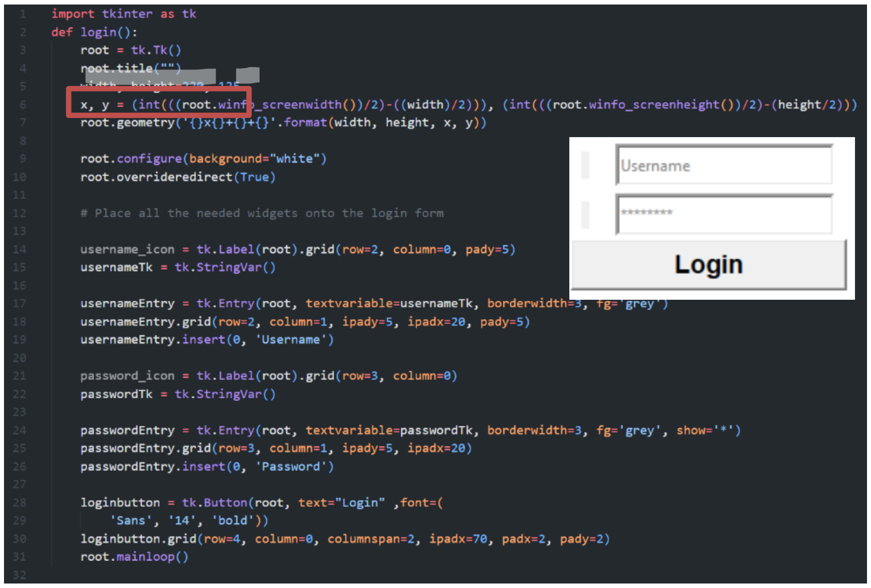
```python
import tkinter as tk
def login():
    root = tk.Tk()
    root.title("")
    #size of window and location on screen
    x, y = (int(((root.winfo_screenwidth())/2)-((220)/2))), (int(((root.winfo_screenheight())/2)-(165/2)))
    root.geometry('{}x{}+{}+{}'.format(220, 165, x, y))

    root.configure(background="white")
    root.overrideredirect(True)

    root.mainloop()
```

Explanation of the code:

I imported the Tkinter module and began by encapsulating the form within a subprogram named login. The code defines properties of the root window such as its size, location and looks. Since Tkinter doesn't have a default method to center windows within a desktop, I had to define the center X, Y location within the desktop where the window would sit. The X and Y location are calculated by taking the screen width/height and dividing by 2 to reach the center ,and since the anchor of the window is the top left corner, you must translate the window by the width and height of the form. Once these values are calculated you must cast them into integers since the Tkinter geometry function doesn't accept any other data type. This gives you a centered root.

The overrideddirect Tkinter command removes the window toolbar so that we can later implement our own better functioning and modern looking toolbar. I did consider not implementing my own toolbar due to excess work required however the toolbar module is a reusable component across all my modules and windows allowing me to reuse in later parts of the project.

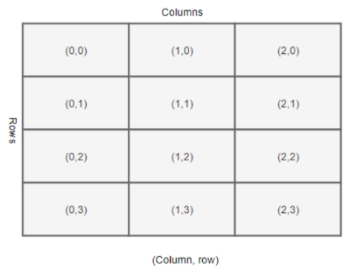
Adding the login form widgets



```python
import tkinter as tk
def login():
    root = tk.Tk()
    root.title("")
    width, height=220, 135
    x, y = (int(((root.winfo_screenwidth())/2)-((width)/2))), (int(((root.winfo_screenheight())/2)-(height/2)))
    root.geometry('{}x{}+{}+{}'.format(width, height, x, y))

    root.configure(background="white")
    root.overrideredirect(True)

    # Place all the needed widgets onto the login form

    username_icon = tk.Label(root).grid(row=2, column=0, pady=5)
    usernameTk = tk.StringVar()

    usernameEntry = tk.Entry(root, textvariable=usernameTk, borderwidth=3, fg='grey')
    usernameEntry.grid(row=2, column=1, ipady=5, ipadx=20, pady=5)
    usernameEntry.insert(0, 'Username')

    password_icon = tk.Label(root).grid(row=3, column=0)
    passwordTk = tk.StringVar()

    passwordEntry = tk.Entry(root, textvariable=passwordTk, borderwidth=3, fg='grey', show='*')
    passwordEntry.grid(row=3, column=1, ipady=5, ipadx=20)
    passwordEntry.insert(0, 'Password')

    loginbutton = tk.Button(root, text="Login" ,font=(
        'Sans', '14', 'bold'))
    loginbutton.grid(row=4, column=0, columnspan=2, ipadx=70, padx=2, pady=2)
    root.mainloop()
```

Explanation of the code:

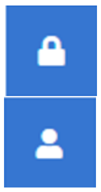
I began the next stage by removing the hardcoded window size values and replacing them with easily

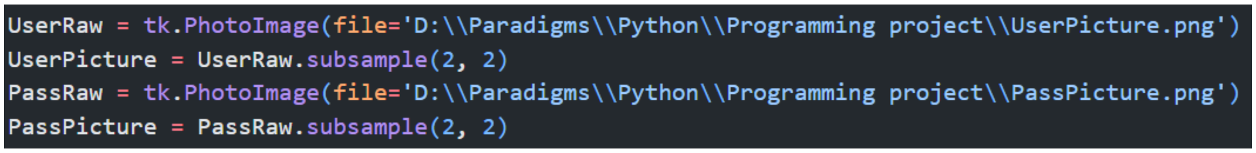


changeable variables. This will save time in the next stage when the toolbar is created.

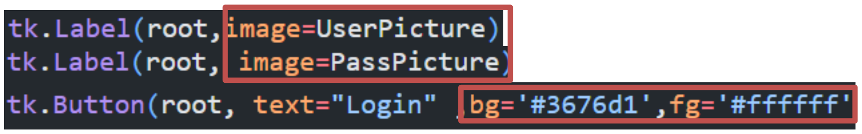
My widget layout was organized using the Tkinter grid method. There are many other methods within Tkinter to place widgets such as Pack and Place however I thought that this layout would provide a layout like the one I previously designed in the project.

The widgets are laid out in a 2x3 grid with an icon widget to be filled and an input box for the user, the button spans across two columns. Each input box has a Tkinter variable attached so that later in the project I can start taking in inputs from a user for authorization. The Password entry shows '*' so that user password are kept safe.

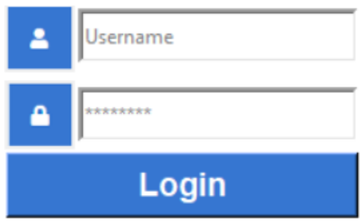




```
UserRaw = tk.PhotoImage(file='D:\\Paradigms\\Python\\Programming project\\UserPicture.png')
UserPicture = UserRaw.subsample(2, 2)
PassRaw = tk.PhotoImage(file='D:\\Paradigms\\Python\\Programming project\\PassPicture.png')
PassPicture = PassRaw.subsample(2, 2)
```

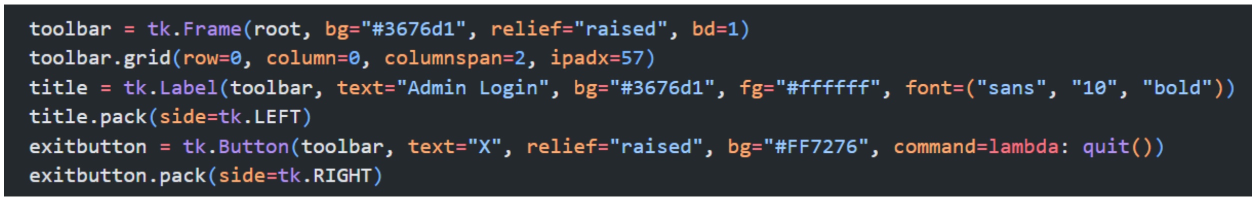


```
tk.Label(root, image=UserPicture)
tk.Label(root,  image=PassPicture)
tk.Button(root, text="Login"  bg='#3676d1',fg='#ffffff'
```



By using the PIL Photo library imported with Tkinter I can import the photos above and sample into the right size for the form. The labels are declared with the image within and the login button colour is changed for a better user experience.

Adding the Toolbar

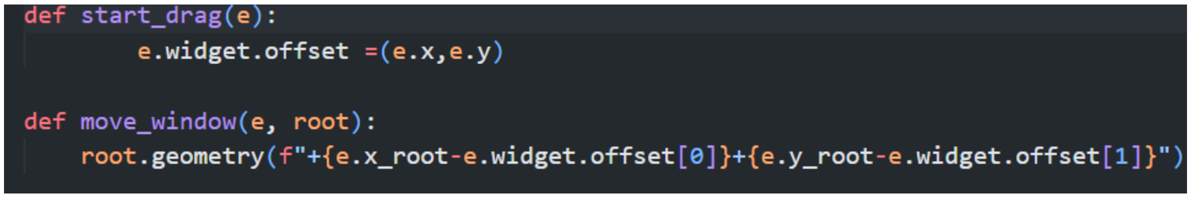


```
toolbar = tk.Frame(root, bg="#3676d1", relief="raised", bd=1)
toolbar.grid(row=0, column=0, columnspan=2, ipadx=57)
title = tk.Label(toolbar, text="Admin Login", bg="#3676d1", fg="#ffffff", font=("sans", "10", "bold"))
title.pack(side=tk.LEFT)
exitbutton = tk.Button(toolbar, text="X", relief="raised", bg="#FF7276", command=lambda: quit())
exitbutton.pack(side=tk.RIGHT)
```

Explanation of code:

Since the toolbar will be like a window within a window, I have opted to make the toolbar widget a Frame. This frame will be gridded in the first row of the window and all the required widgets will be packed within the Frame. My decision to use pack within the frame is due to its Left/Right capabilities which work well when the requirements is widgets both in the top left corner and top right. I have titled my frame and packed it to the left alongside with an exit button packed to the right. The execution of this button runs a lambda quit command, destroying the window. Lambda prevents the execution of the command on initial run, only executing when the button event occurs.

My toolbar now has some functionality however it is missing the window moving capabilities.

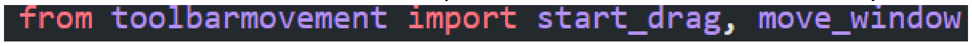


```
def start_drag(e):
        e.widget.offset =(e.x,e.y)

def move_window(e, root):
    root.geometry(f"+{e.x_root-e.widget.offset[0]}+{e.y_root-e.widget.offset[1]}")
```

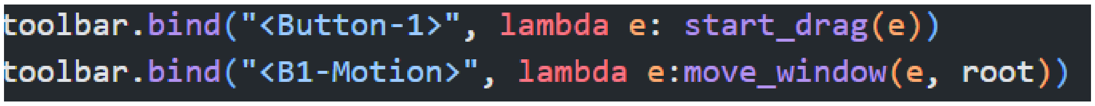
The start drag sub program stores the x and y values of the mouse position relative to the upper left corner of the widget, this function had to be added since without it the toolbars wouldn't be offset correctly and jumping when dragging was common.

The move window function changes the root geometry of the toolbar to the mouse position relative to the top left-hand corner of the screen at the start of the event minus the previous e.x/y mouse event position which are stores in a list; x being list location 0 and y being list location 1. When the location of the screen is taken away from the location of the mouse you got a smooth path of translation movement relative to the position clicked on the toolbar.

These Functions will be stores in a separate file to be reused across all my windows.



```
from toolbarmovement import start_drag, move_window
```



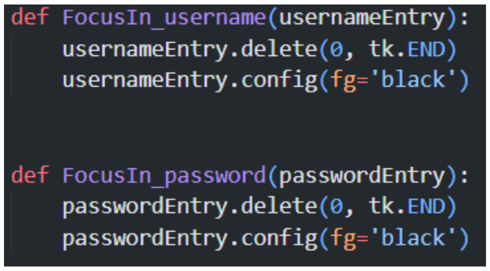
```
toolbar.bind("<Button-1>", lambda e: start_drag(e))
toolbar.bind("<B1-Motion>", lambda e:move_window(e, root))
```

Since the Functions are in a separate file, I need to import them separately at the top of the file

The next line binds the pressing of the left mouse click to the start drag function, lambda e referring to the command only being executed when the toolbar is clicked. The next bind executes the move-window function every time the mouse moves whilst the left-click mouse button is being pressed. The function will be continuously executed until the mouse button is released. I opted for left click mouse button because other programs generally opt for this bind meaning that the user experience will be better.

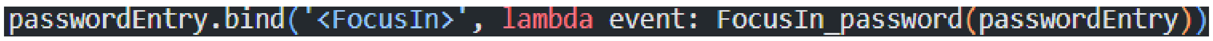
User Experience

The login GUI is essentially finished however there are some small details that I've decided to implement that benefits the way in which the user interacts with the program



```
def FocusIn_username(usernameEntry):
    usernameEntry.delete(0, tk.END)
    usernameEntry.config(fg='black')


def FocusIn_password(passwordEntry):
    passwordEntry.delete(0, tk.END)
    passwordEntry.config(fg='black')
```

These two functions simply delete all the values within the usernameEntry and passwordEntry respectively and changes the font colour from the grey shown when loading to black



```
passwordEntry.bind('<FocusIn>', lambda event: FocusIn_password(passwordEntry))
```

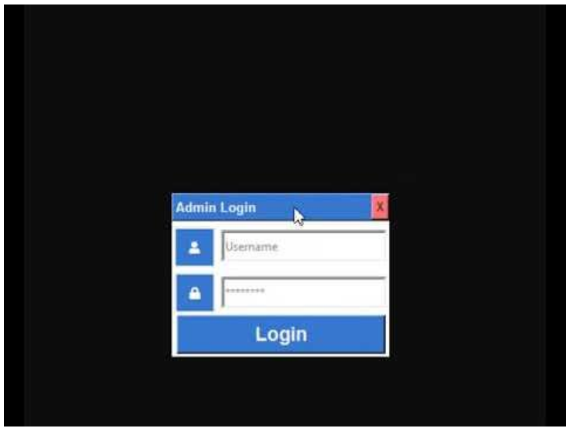
These binds activate the functions when the user presses the respective Entry widget, giving a visual response to show the user that the Entry is ready for an input.

```
usernameEntry.bind('<FocusIn>', lambda event: FocusIn_username(usernameEntry))
```

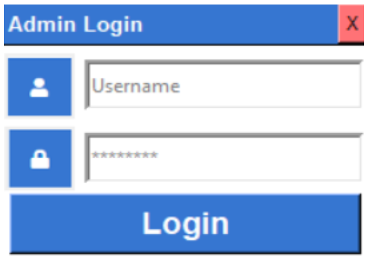
```
root.bind('<Escape>', lambda event: quit())
root.bind('<Down>', lambda event: passwordEntry.focus())
root.bind('<Up>', lambda event: usernameEntry.focus())
```

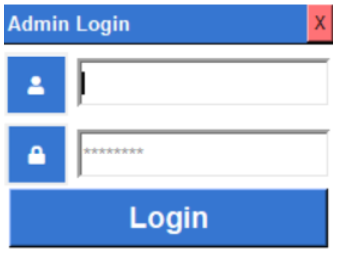
The top bind allows the user to leave the login form by pressing the escape button. The other binds allow the user to navigate between widgets via the arrow keys.

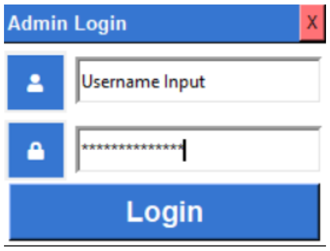
Module Testing



This video shows the smooth dragging of the toolbar along with the changes made from when previously shown. This confirms that the toolbar is working as intended and is appropriate for user use







By clicking on the entry form we can test the viability of the focus in functions, they seem to be functioning correctly removing any previous instruction text that appears by default in the login form. Finally, I check that the entry boxes hold string inputs

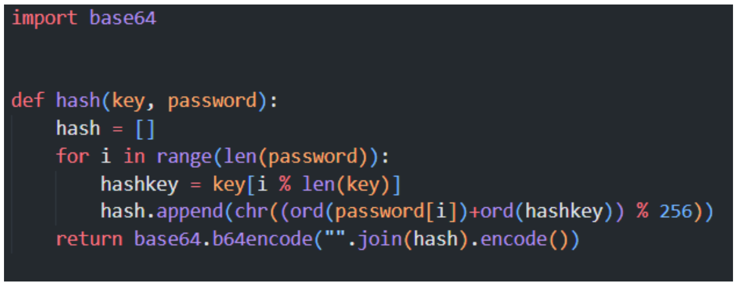
User Feedback

Responsive and intuitive user interface, no complaints given. User liked the implementation of the UX keys since they are comparable to keys used in other mainstream programs

MILESTONE 2 – AUTHORISING USER LOGIN

Objective of the milestone: Authorise user inputs against database stored values to access main program

Hashing Algorithm



```
import base64


def hash(key, password):
    hash = []
    for i in range(len(password)):
        hashkey = key[i % len(key)]
        hash.append(chr((ord(password[i])+ord(hashkey)) % 256))
    return base64.b64encode("".join(hash).encode())
```

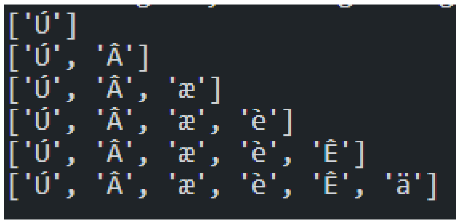
Explanation of code:

The base64 module provides functions for encoding binary data which is exactly what I need to create a hash value. There are many different encoding versions however I decided to opt for the base64.b64encode versions since it is the simple encoding function rather than any other complex and unneeded functions included within the b64 library.

This hashing function is stored separately for reusability therefore the hash is defined as a function with parameter password, this being the value to be encoded, and key, the value which will provide the way in which the password is encoded.

The code loops through based on the length of the password provided, encrypting each letter in each cycle. The hash key is created based on the key itself, each loop has a different hash key which is essentially an iteration of itself. If an input username was username1 the output key would range from u to n with each loop hashing with the letters within the range. The key ranges in size depending on the input length so that hash cracking is incredibly difficult.

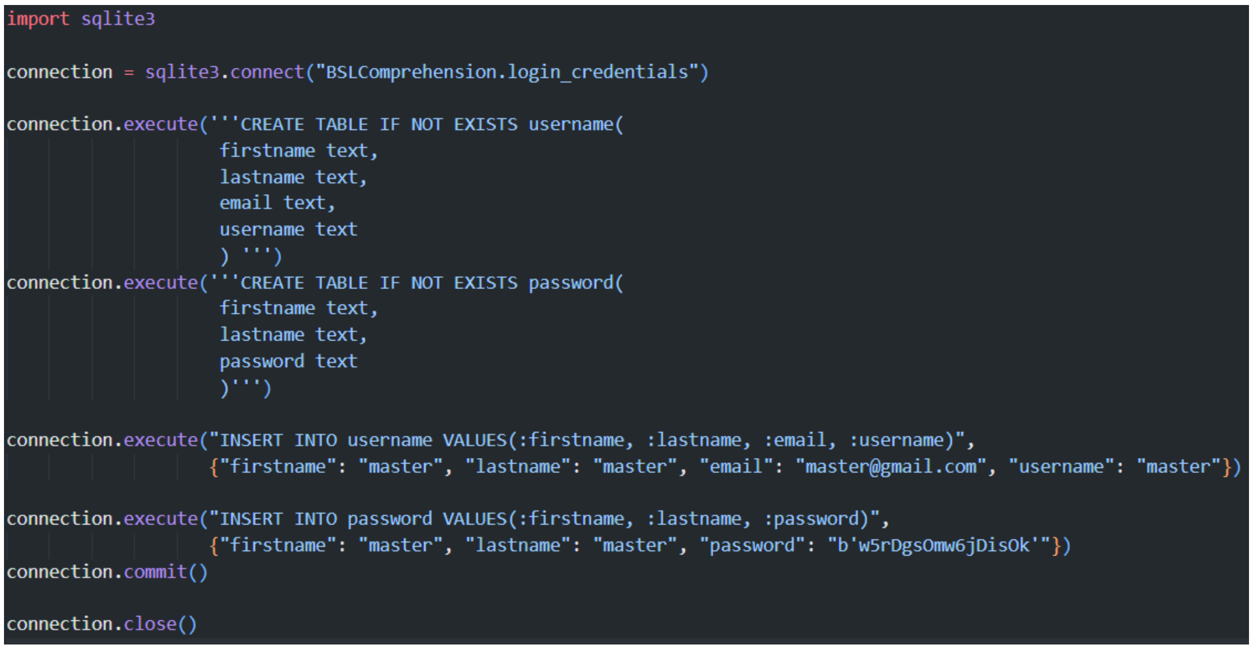
The hash list is then appended with a new character for each character in the password entry, these new characters are different from the original input because they are shifted using the hash key. These values fall within the characters available in ASCII.



```
['Ú']
['Ú', 'Â']
['Ú', 'Â', 'æ']
['Ú', 'Â', 'æ', 'è']
['Ú', 'Â', 'æ', 'è', 'Ê']
['Ú', 'Â', 'æ', 'è', 'Ê', 'ä']
```

When these new values are appended to the hash list I returns and encrypted version of this list using the b64encode library. This return essentially iterates through every new character like I previously did and encodes every character separately, producing a 3 character string from every one character. This produces a larger string which is unrecognizable from the original input.

Creating the SQL tables



```
import sqlite3

connection = sqlite3.connect("BSLComprehension.login_credentials")

connection.execute('''CREATE TABLE IF NOT EXISTS username(
                    firstname text,
                    lastname text,
                    email text,
                    username text
                    ) ''')
connection.execute('''CREATE TABLE IF NOT EXISTS password(
                    firstname text,
                    lastname text,
                    password text
                    )''')

connection.execute("INSERT INTO username VALUES(:firstname, :lastname, :email, :username)",
                   {"firstname": "master", "lastname": "master", "email": "master@gmail.com", "username": "master"})

connection.execute("INSERT INTO password VALUES(:firstname, :lastname, :password)",
                   {"firstname": "master", "lastname": "master", "password": "b'w5rDgsOmw6jDisOk'"})
connection.commit()

connection.close()
```

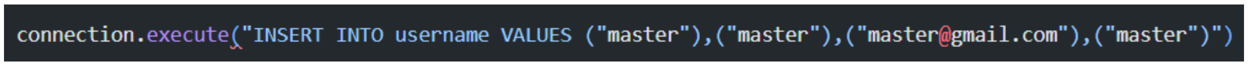
Explanation of code:

I began by implementing the SQLITE3 database library that is made for Python. This library provides a lightweight alternative to cloud SQL, meaning that I won't have to run my own servers to connect to for my databases. Although it runs locally, SQLITE3 uses the same commands for connection and transferring as any other SQL library. If any point I would like to run a cloud server with my databases, this would be an easy change.

I connect to the database file named BSLComprehension.login_credentials. Once this was completed I began to execute some SQL to create the necessary tables for the data I needed to store, the first table named username with the necessary data and its data type. Ditto for the password table.

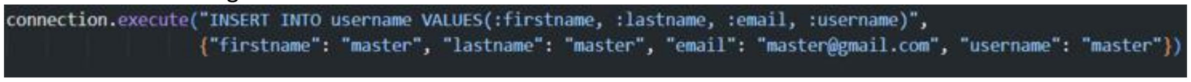
After the tables were created, I needed to insert a default user that I could use to test against once the SQL was integrated with my form. The password insert was already hashed as the form will always hash values before comparing against the SQL table to ensure security for the user.

Fixes:



```
connection.execute("INSERT INTO username VALUES ("master"),("master"),("master@gmail.com"),("master")")
```

In the first iteration of testing and researching I found that the method above of insertion (which I had originally used) was susceptible to SQL injections. SQLITE3 allows named style place holders, the parameters inserted should be an instance of a dictionary. Any extra items are ignored preventing injections with the new method . Although the project is unlikely to face these problems at this stage, future proofing its code structure is never a bad idea.

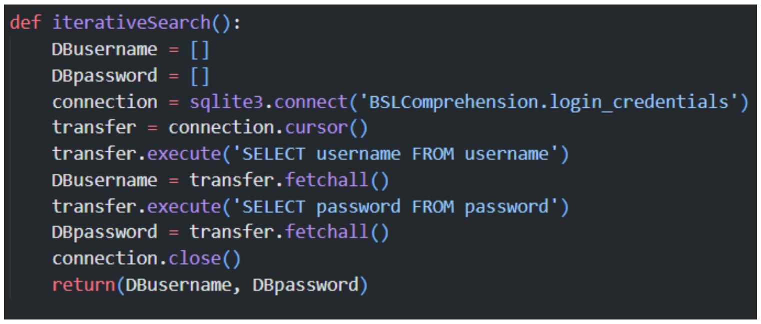


```
connection.execute("INSERT INTO username VALUES(:firstname, :lastname, :email, :username)",
                   {"firstname": "master", "lastname": "master", "email": "master@gmail.com", "username": "master"})
```

Using these named place holder dictionaries before inserting the values is much safer than the method used before. It gives a discrete number of inserts preventing overloading by injection.

I then committed all the changes to the table and closed the connection between the database and the file.

Iteration and Authorisation

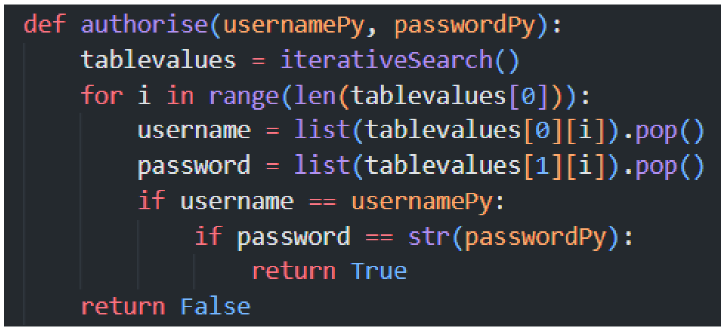


```
def iterativeSearch():
    DBusername = []
    DBpassword = []
    connection = sqlite3.connect('BSLComprehension.login_credentials')
    transfer = connection.cursor()
    transfer.execute('SELECT username FROM username')
    DBusername = transfer.fetchall()
    transfer.execute('SELECT password FROM password')
    DBpassword = transfer.fetchall()
    connection.close()
    return(DBusername, DBpassword)
```

Explanation of code:

I first created two lists in which the values fetched from the databases would be stored, DB username and DB password.

I then proceed to connect to the previously made SQL database and create a cursor. Cursors allow the fetching and transfer of data between the SQL tables and the file itself. Individually for each list, I select all the values from the username field and password field and allocate them in their respective lists. The cursor fetchall() command executes the SQL commands and then fetches all the selected values. The connection is closed, and the lists created from the SQL data is returned from the function.



```
def authorise(usernamePy, passwordPy):
    tablevalues = iterativeSearch()
    for i in range(len(tablevalues[0])):
        username = list(tablevalues[0][i]).pop()
        password = list(tablevalues[1][i]).pop()
        if username == usernamePy:
            if password == str(passwordPy):
                return True
    return False
```

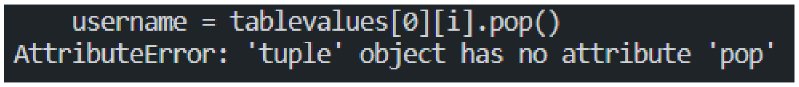
Explanation of code*:*

I decided to separate the Iteration and Authorisation functions for modularity within my program and so that readability was better for those who did not create the program.

First, I define the function with two necessary parameters. usernamePy and passwordPy are the user inputs from the login form which we will use to compare against the values in the database. We call the function iterativeSearch which I just create and the values list values returned are stored in the table values list, essentially creating and matrix of values to be used.

We use a count-controlled loop dependent on the length of row 0 in the tablevalues matrix to iterate through. This means that the loop will never exceed more loops than there are data values stores in the table.

I use the i value which holds the current instruction number to iterate through the first and second row of tablevalues, popping each value as iteration occur. The list casting was required since you cannot pop values from a tuple in python.

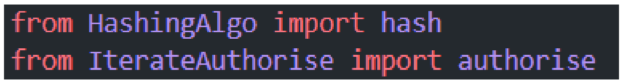


```
    username = tablevalues[0][i].pop()
AttributeError: 'tuple' object has no attribute 'pop'
```

The popped values username and password are compared against the original parameters usernamePy and passswordPy. If both these values are the same the function returns a Boolean True value authorising the login process. If one is incorrect the function returns Boolean False. Every iteration returns a Boolean value.

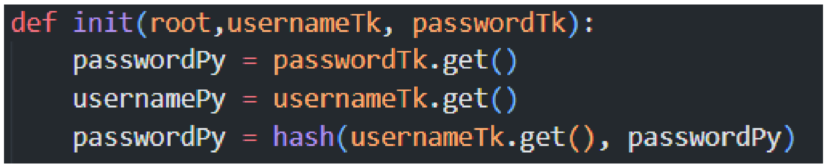
Initializing authorisation in the login form

Explanation of code:



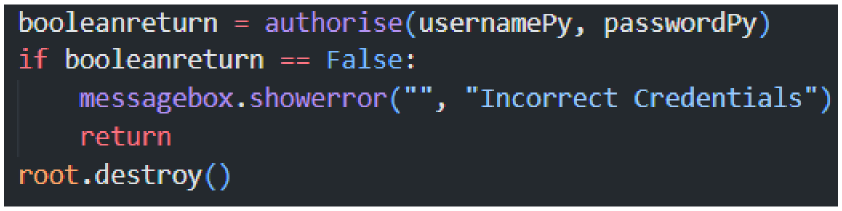
```
from HashingAlgo import hash
from IterateAuthorise import authorise
```

I import the functions I have just been working on to use in the initialization function

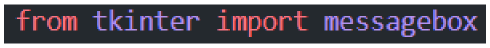


```
def init(root,usernameTk, passwordTk):
    passwordPy = passwordTk.get()
    usernamePy = usernameTk.get()
    passwordPy = hash(usernameTk.get(), passwordPy)
```

Then I create the init function, setting all the necessary parameters. Since Tkinter variables can't be directly accessed by python variables I have to use the .get() method to access the unencrypted data. Of course, the retrieved password is not hashed so I call the hash function I created and pass the username variable as the key and password. This sets the new password to the hashed version of the user input password.



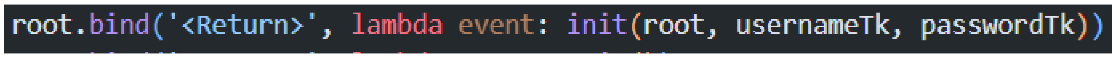
```
booleanreturn = authorise(usernamePy, passwordPy)
if booleanreturn == False:
    messagebox.showerror("", "Incorrect Credentials")
    return
root.destroy()
```



```
from tkinter import messagebox
```

These values are then passed into the authorise function. If the Boolean return variable (the return value from authorise) matches False then, since we imported message box from Tkinter, we can show a pop-up error displaying incorrect credentials for the user. If Boolean True is returned, then the selection never occurs, and the root window is directly destroyed. Since the login form instance occurs within main.py, if it is destroyed then the main.py program continues, and the main root will load.

Implementing into login()



```
root.bind('<Return>', lambda event: init(root, usernameTk, passwordTk))
```

```
loginbutton = tk.Button(root, text="Login", bg='#3676d1', fg='white', font=(
    'Sans', '14', 'bold'), command=lambda: init(root, usernameTk, passwordTk))
```

Explanation of code:

I binded the Enter button on the keyboard to the execution of initialise and the command for the login button was also binded to initialise. If the user performs either of these actions, then the authorisation process will take place.

Module Testing

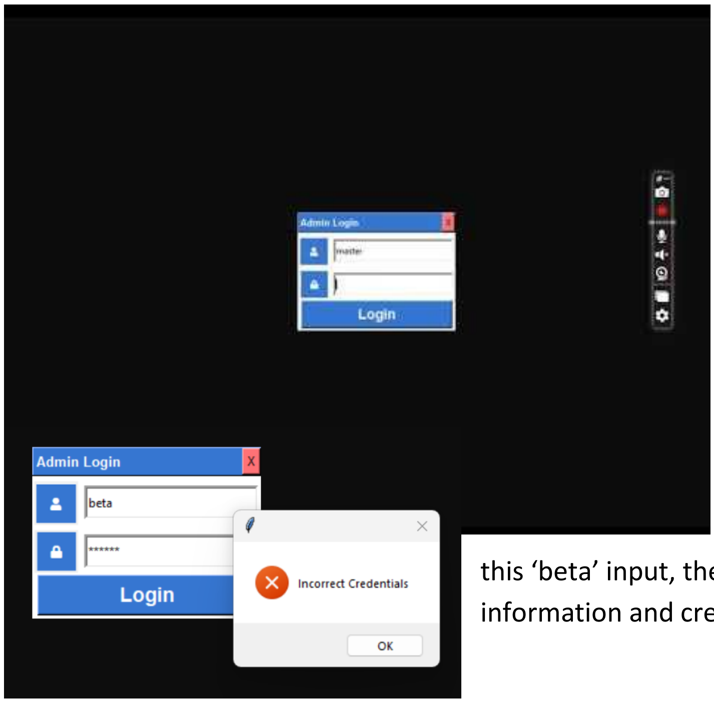
Username table:

firstname	***lastname***	***email***	***username***
firstmaster	*lastmaster*	-------------------------------	*master*
firstmaster2	*lastmaster2*	-------------------------------	*master2*

Password table:

firstname	**lastname**	**password**
firstmaster	lastmaster	master
firstmaster2	lastmaster2	master2

The tables above show the temporary test values used in each table within the database.

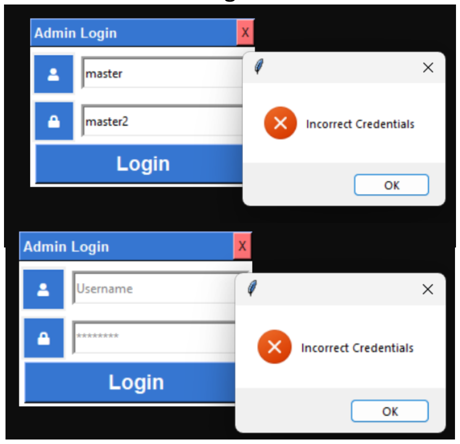


Incorrect Credentials

I begin to test the bind keys with the values now being input. Throughout the video, inputs are entered and submitted without the mouse ever hovering over the form which is exactly one of the goals I set out to accomplish.

In this example, I used the first values stores in the test tables. When the correct values where entered, the login form successfully authorised and closed. The main form then appears.

If the wrong values are entered, such as this 'beta' input, then the form doesn't authorise the information and creates a Tkinter error pop up as expected.



I continue to test using the other value in the tables. To make sure that a user couldn't use another login information, I tested the master username against the master2 password. When the authorisation began an error popped-up confirming that users couldn't use other information as their own within the form. The non-viewable password has been disabled for this example.

Empty entry fields are also not accepted as valid inputs for the database.

User Feedback

The form is very responsive when an error is detected, and they like the decision to show the user a large red cross however they think the error message should be changed to incorrect password/username

MILESTONE 3 – LOADING SCREEN IMPLEMENTATION

Objective of the milestone: Create a loading screen which tells user the remaining time until main opens

For the loading screen I will be using the TQDM.tk library which is a fast. Extensible progress bar for Python. TQDM is usually used within command line application however the handy TQDM.tk library integrates the command line progress bar into a tk application which can be used alongside Tkinter windows and widgets. This library is currently experimental so there is not many features to use however it is my best option of progress bar in python which works in compatibility with Tkinter programs.

```
from tqdm.tk import tqdm
```

I import the tqdm.tk library to begin creating the Tkinter progress bar for my application.

The tqdm progress bar must be connected to a root window however I am loading between windows I don't want the root master to be visible to the user once it is integrated.

```
root.attributes('-alpha', 0)
```

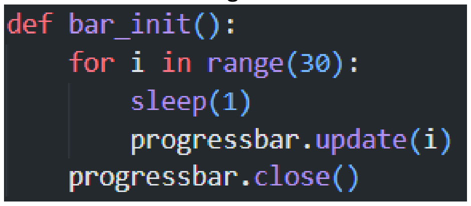
Using the root master attributes method I can change the alpha of the window to zero, this means that the window is now transparent and not visible to the suer when the progress bar is loading.

```
progressbar= tqdm(total=30,desc='Loading Packages' ,tk_parent=root)
```

I define the progress bar widget as a new tqdm bar, this will have 30 increments and a tag stating to the suer that the main program is loading. This progress bar as I mentioned before is attached to the root window.

```
from time import sleep
```

Since this progress bar program will be incremented with time I import the time library, specifically the sleep function which will allow me to create a loading appearance while filling in the tqdm bar.



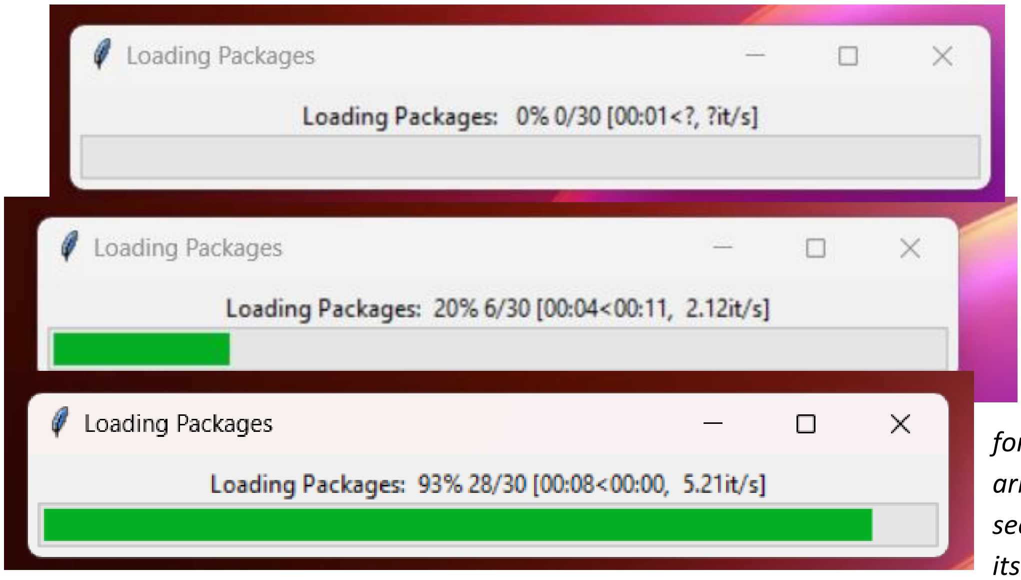
```
def bar_init():
    for i in range(30):
        sleep(1)
        progressbar.update(i)
    progressbar.close()
```

I define the bar filling function.

I iterate through a range equal to the number of slots in the bar, sleeping between each cycle so that the bar takes enough time, meaning that main can load.

At the end of each cycle, I update the progress bar with the cycle number I, this function runs until the progress bar is completely full.

Module Testing



```
Loading Packages
Loading Packages:   0% 0/30 [00:01<?, ?it/s]

Loading Packages
Loading Packages:  20% 6/30 [00:04<00:11,  2.12it/s]

Loading Packages
Loading Packages:  93% 28/30 [00:08<00:00,  5.21it/s]
```

The pictures show the different stages that the loading bar takes, it shows all information of the current process and the time remaining.

Although this is the exact look of a loading screen that I was going for, the loads is completely arbitrary and tqdm doesn't seem to have a way to link its loading stage to any loading times for programs. This is, not tqdm itself, but rather the tqdm.tk experimental library does not include this comaptability. Having known this now, the next cycle of iteration I would like to try to use a different library which I can find which would be compatible which both Tkinter and loading time implementations.

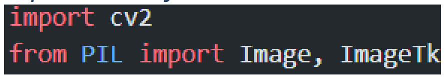
User Feedback

The loading screen appeared straight away and looked very professional however it kept going after it reached maximum an the data above became question marks telling me that the abr doesn't work completely perfectly.

MILESTONE 4 – DISPLAYING CAMERA VIEW

Objective of the milestone: Display a live feed within a Tkinter GUI

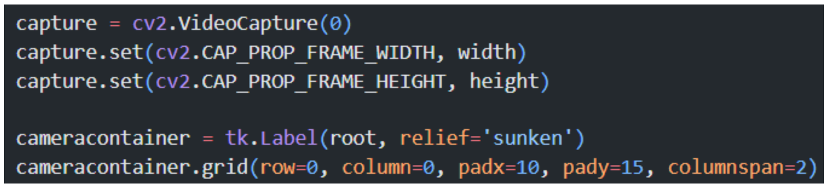
Explanation of code:



```
import cv2
from PIL import Image, ImageTk
```

I began by importing the two necessary libraries for displaying a camera feed, cv2 or pythons open-source computer vision library allows me to display live camera feed on my screen. The PIL import will become helpful later in the program. The video frames aren't supported images within a Tkinter widget so we must convert image types and display them on widgets after using PIL, pythons imaging library.

Accessing the camera

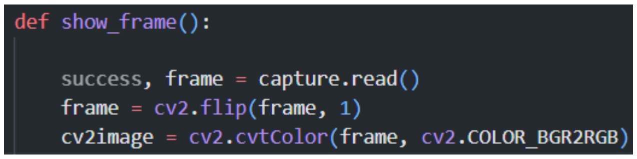


```
capture = cv2.VideoCapture(0)
capture.set(cv2.CAP_PROP_FRAME_WIDTH, width)
capture.set(cv2.CAP_PROP_FRAME_HEIGHT, height)

cameracontainer = tk.Label(root, relief='sunken')
cameracontainer.grid(row=0, column=0, padx=10, pady=15, columnspan=2)
```

I created the capture variable which holds the live video capture. Value 0 referring to the in-built camera inside my device, external devices would use different values as they are not default. This causes multiplatform limitation with my program since some devices may not contain built in cameras and opt for an external device instead. The program would encounter an error when reading and wouldn't display the live feed.

I then created a Tkinter label widget that will eventually store the contents of the camera feed once the show frame function has been called.

Capturing frames

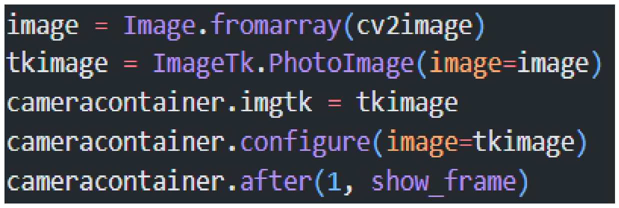


```
def show_frame():

    success, frame = capture.read()
    frame = cv2.flip(frame, 1)
    cv2image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
```

I continued and created the show frame function which made up the bulk of the milestone. Using the previously made variable capture which stores the live feed, I capture a frame and store it within the frame variable. Cv2 read() method will produce two outputs, the first being whether the capture was successful. We will use the results of this later in the program when displaying the pipeline. Since the camera live feed capture is flipped, we flip it back so that the user gets an image which is the correct orientation in the final program.

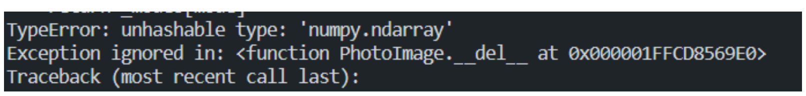
Since cv2 reads the frame BGR I must convert the image colour so that the PIL image library can display the frames correctly in a way that is appealing to users, for this I used the inbuilt BGR2RGB method in cv2.

Displaying the frame



```
image = Image.fromarray(cv2image)
tkimage = ImageTk.PhotoImage(image=image)
cameracontainer.imgtk = tkimage
cameracontainer.configure(image=tkimage)
cameracontainer.after(1, show_frame)
```

Since the cv2image object is captured as a numpy array of data, I must convert it into a PIL image array object that then can be converted into a full image using the information provided by the PIL matrix object. For this I used the fromarray method which is especially made to cast image arrays stored by other libraries into PIL.

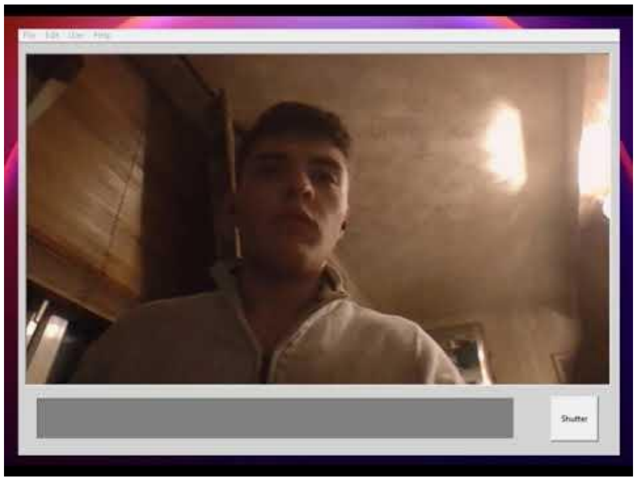


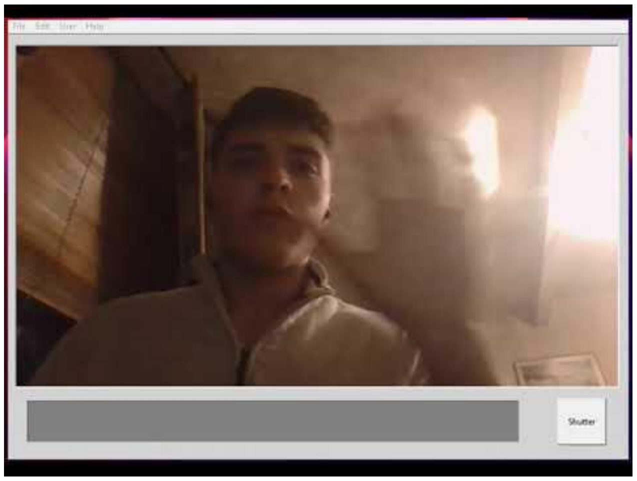
```
TypeError: unhashable type: 'numpy.ndarray'
Exception ignored in: <function PhotoImage.__del__ at 0x000001FFCD8569E0>
Traceback (most recent call last):
```

Not converting produces a numpy hashing error

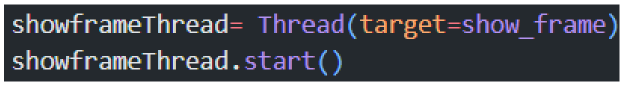
I convert the array of PIL data from the image into a PhotoImage object and then store this PhotoImage object within the camera container widget. I set the current image to the stored Photo image and recursively call the showframe function. The function is called every millisecond as to give the illusion of fluidity to the user. I tried to change the values so that less frames would show to increase performance however that simply slows the program to a level that is not usable.

The videos below show a side-by-side comparison of the difference between a 100ms refresh of each capture vs a 1ms refresh of each capture, although not perfect it is suitable for my needs and is stated in my analysis as one of the limits of my program. →Video was shot with temporary widget placeholders to make test more realistic.



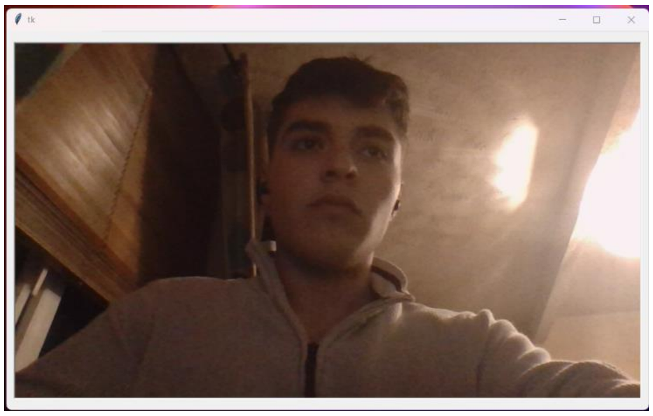


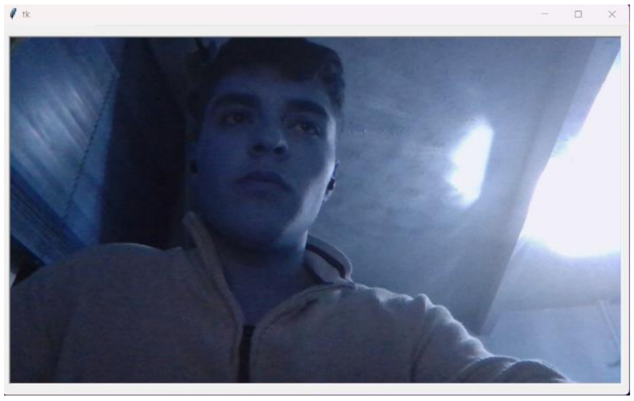
Finally, I wasn't happy with the load speed of the camera feed and thought this might have been due to the sue of one singular thread for all processes. By importing the threading library, I managed to allocate all show frame recursion to a separate thread making the overall program faster. The first line create the thread and assigns the showframe function to it and then I begin the threads processing before the rest of the program loads, using sequencing principles.



```
showframeThread= Thread(target=show_frame)
showframeThread.start()
```

Module Testing





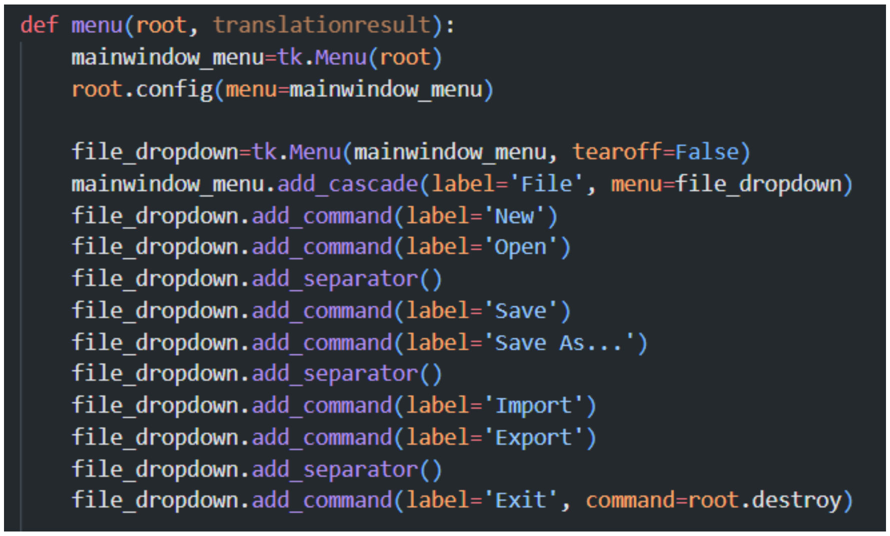
The videos above show the live feed within the Tkinter GUI. These picture show the current state of the project without any temporary widget place holders. The data from the frame will be analyzed in the BGR format shown on the right, as learning models use this to recognize greyscale density rather than knowing what each object is. The left picture shows the user view which will also include the media pipe pipeline overlay for hands.

User Feedback

Very responsive and the latency is good enough for it to be usable.

MILESTONE 5 – NAVIGATION MENU

Objective of milestone: Create a menu to navigate program for user



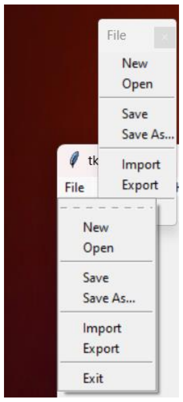
```
def menu(root, translationresult):
    mainwindow_menu=tk.Menu(root)
    root.config(menu=mainwindow_menu)

    file_dropdown=tk.Menu(mainwindow_menu, tearoff=False)
    mainwindow_menu.add_cascade(label='File', menu=file_dropdown)
    file_dropdown.add_command(label='New')
    file_dropdown.add_command(label='Open')
    file_dropdown.add_separator()
    file_dropdown.add_command(label='Save')
    file_dropdown.add_command(label='Save As...')
    file_dropdown.add_separator()
    file_dropdown.add_command(label='Import')
    file_dropdown.add_command(label='Export')
    file_dropdown.add_separator()
    file_dropdown.add_command(label='Exit', command=root.destroy)
```

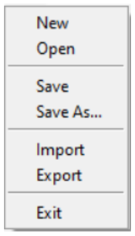
Explanation of code:

Creating the menu widget

I begin by creating a basic menu function which takes in the Tkinter root windows as a parameters so that it can be overlayed onto the main window, the translation result parameter is for later use; it will be useful when creating the export function of the program. I define the main window menu by initialise a Tkinter menu instance and assign this to root. While this menu has been created it is not assigned as the current root menu, so I call the root configuration and set the default menu to the just created main window menu.

Adding menu cascades





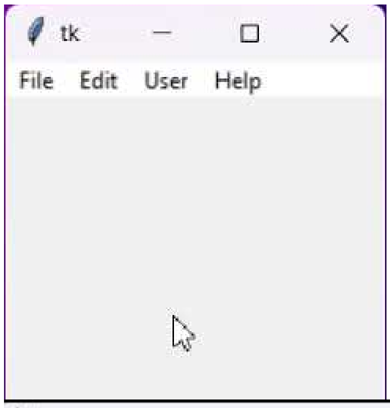
I create the first dropdown by creating another menu named file following the top-down design created in the design section, the images showcase the difference between having a tear off enabled vs disabled. The tear off allows users to view the cascade as a separate window. This was not part of my design, and I did not view it as beneficial, so I removed it.

Since this file dropdown is a separate menu entity from the main menu, I called the add cascade method of the main menu and set the first cascade to the file dropdown, label 'file'.

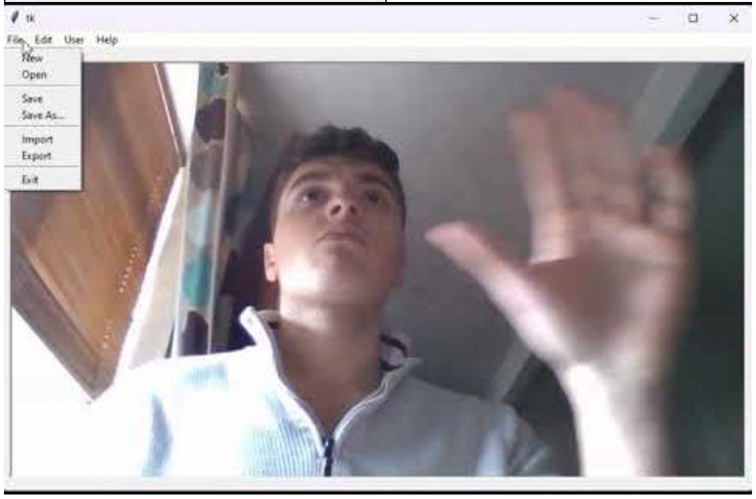
I then proceeded to add all the sub commands within the cascade which I will later build on so that the respective events occur when the user interacts with them. The final command showcases what a finished command within a cascade would look like. The exit command within the file cascade uses the inbuilt destroy method to close the program when the exit label is interacted with.

This process is repeated for all cascades required in the main menu and code is incredibly similar, with changes only occurring when labels are slightly different.

Module Testing



The cascades in the navigation menu have a good response time when clicked and flowing between each cascade is smooth. To test out whether commands will work when added later, I tested out the exit button at the end of the clip. It seems to be functioning correctly. This clip shows the navigation menu implemented in a separate window; the clip below demonstrates the navigation bar when integrated into the main program.



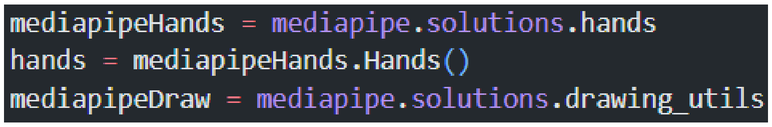
The navigation menu remains responsive when implemented into the main program. The user can view the live feed whilst navigating the menu due to the threading used for the feed function. Not only does the navigation remain smooth but the commands within the cascades function correctly as well.

User Feedback

The navigation menu is a bit difficult to get opened since it has to be clicked twice with a delay however once opened it is easy to navigate and seems like any other program.

MILESTONE 6 – DISPLAYING PIPELINE

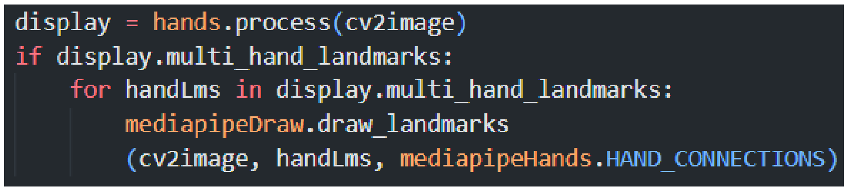
Objective of milestone: Display Hand pipeline for later use in model



```
mediapipeHands = mediapipe.solutions.hands
hands = mediapipeHands.Hands()
mediapipeDraw = mediapipe.solutions.drawing_utils
```

Explanation of code

To begin displaying the pipeline I had to import media pipes trained hand detection model. [Bibliography reference 5] Media Pipe Hands utilizes an ML pipeline consisting of multiple models working together. They have manually annotated around 30 thousand real-world image with 21 node coordinates to train the models. Since media pipe has multiple solutions for different requirements, I must import the hands solution and create the model for detection. The drawing utility solution is the method in media pipe that allows me to plot a visual pipeline overlaying the user's hands.

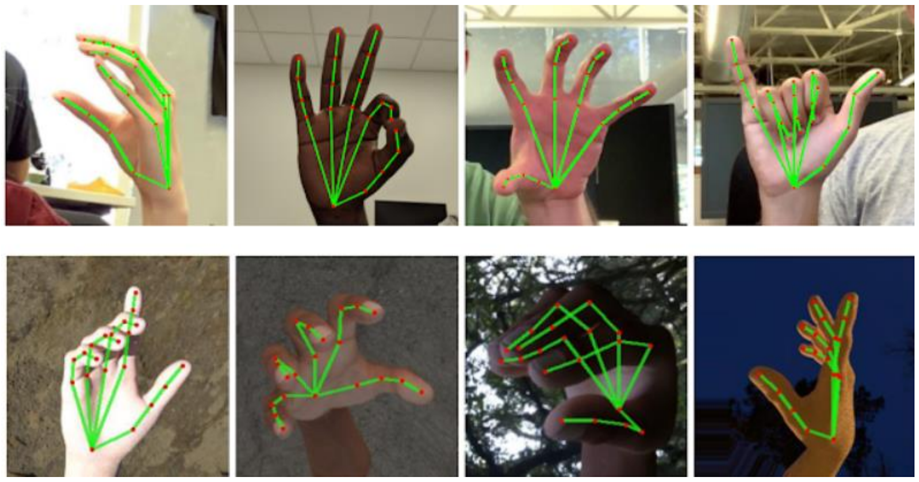


```
display = hands.process(cv2image)
if display.multi_hand_landmarks:
    for handLms in display.multi_hand_landmarks:
        mediapipeDraw.draw_landmarks
        (cv2image, handLms, mediapipeHands.HAND_CONNECTIONS)
```

Pipeline overlay

I take the frame output from the live feed capture used to display in the camera container widget and process it using the newly created media pipe hands model. Once the image has been processed, I use an if statement to check whether the processed image contains any landmarks, this essentially checks to see if the detection has been a success. If not, hands have been detected within the frame, it will check in the next frame capture. If a hand has been detected, I must display the pipeline. The multi_hand_landmarks method for a processed image produces a dictionary of 21 nodes with a 3-size list for each node. This 3-size list stores the x, y and z value of each node which will become very useful later when detecting character outputs.

Using a count-controlled loop, I iterate through every single node detected by the hand model and draw them on top of the frame. When two adjacent nodes are drawn onto the frame, the HAND_CONNECTIONS method will create an edge between them.



Something like this should appear when using the main program.

These images are taken from the mediapipe website [bibliography reference 5]

Differentiating between Left and Right

When the pipeline was displayed, I realized that there was no way of classifying each hand separately. Although this isn't an issue now, when I try to model the signing, I must be able to differentiate between the hands. Media pipe has a hand identification method in the hand model that I just imported, and I will now implement it into my program. This method will return a label with the hand type and also a score estimating the probability that the hand is what is stated on the label.

```
frame = cv2.flip(frame, 1)
```

I began by flipping my frames since this method assumes that the input is flipped. My input camera automatically flips meaning that I will have to flip again so that the processed result is correct. This may become a limitation of my program on separate machines since most camera devices don't automatically flip the feed causing the label identification to be the wrong

```
if result.multi_hand_landmarks:
    for id, handlms in enumerate(result.multi_hand_landmarks):
        lbl = result.multi_handedness[id].classification[0].label
        print(lbl)
        mediapipeDraw.draw_landmarks(cv2image, handlms, mediapipeHands.HAND_CONNECTIONS)
```

way around.

I begin with the same lines of code, checking that a hand is detected and using a count-controlled loop to iterate through nodes. The only difference is the id variable which will classify a hand as either 0 or 1 depending on which appears on the screen first. Since this loop now contains two variables, I had to use the python enumerate function so that an error didn't occur.

I use the multi handedness method within media pipe and the id classification of each hand to produce an identification label and a score of how accurate this is, since I added the label attachment at the end

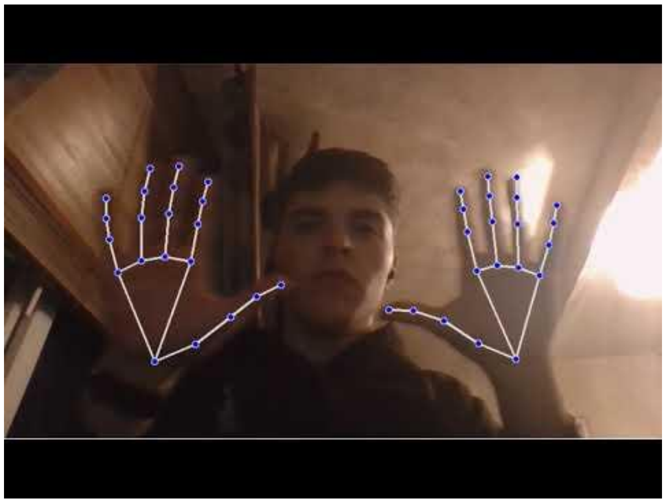
```
TypeError: cannot unpack non-iterable NormalizedLandmarkList object
```

the result within the label variable will only be the detected label without the score.

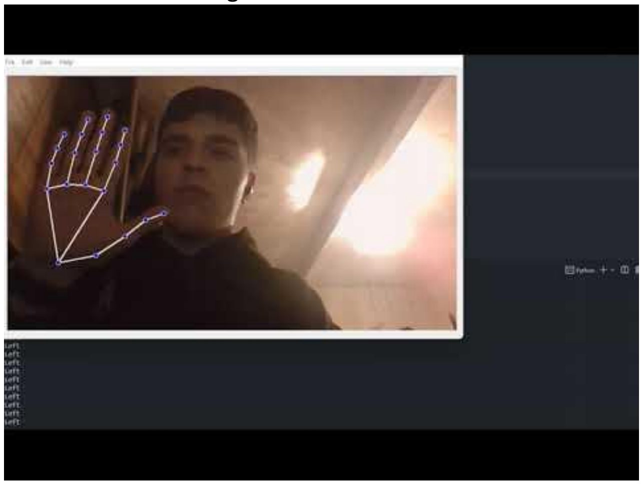
The rest of the process remains the same, plotting each node linearly using the looped landmarks variable.

I've had issues trying to make this pipeline according to my modular design. The hand detection model, and pipeline display do not work in separate file therefore I have opted for keeping the show frame function within the main file. This shouldn't make a difference in performance since the show frame function runs on a separate thread to the main program.

Module Testing



The model is tracking the hands with a decent amount of accuracy, enough for it to be viable for the next stage of the program. However, the model has troubles with sudden fast movements although quickly rectified. The model also produces no output if exposure to light is greater than a certain level. The program will still function outside but hands cannot be directly in front of a light source.



```
Left
Left
Left
Left
Left
Left
Left
Left
Left
Left
```

```
print(lbl)
```

For this test I printer the identification label to check that it was working as required. When the hand first comes into frame you can see that the identification label isn't always accurate however when the hand has remained in frame for a long enough duration the correct label is displayed. This works for both hand independently and also when they are displayed together although his produces an alternating left/right output in the command line.

I'm happy with the quality of tracking in this model and will proceed onto the next stage of the program.

User Feedback

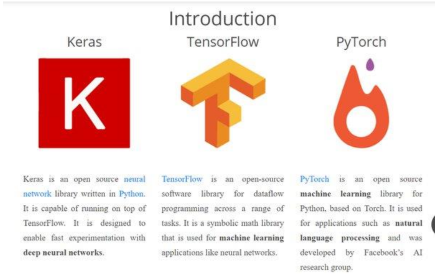
The hand tracking is very cool and new to see. It works when moving around and changing distances but hand detecting doesn't work when hands are over each other or when moving way too fast.

MILESTONE 7 – CHARACTER COMPREHENSION

Objective of milestone: Display BSL characters on screen when correct hand placements are shown on live feed

Deep learning neural networks

I begin this milestone by trying out the most effective and accurate method of identifying the characters, this method difficult and will require some time researching aspects of the deep learning model.



Introduction

Keras

Keras is an open source neural network library written in Python. It is capable of running on top of TensorFlow. It is designed to enable fast experimentation with **deep neural networks**.

TensorFlow

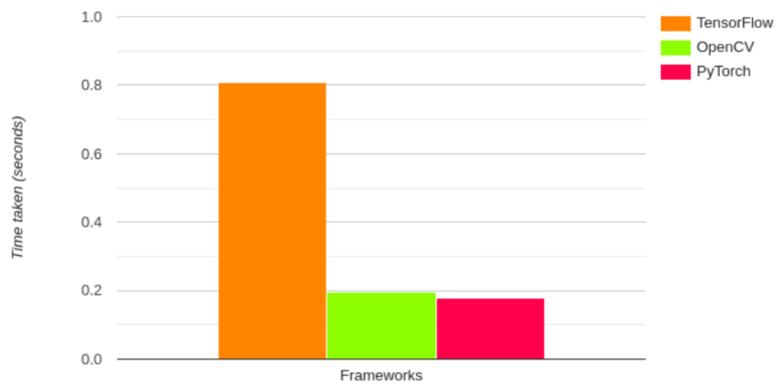
TensorFlow is an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library that is used for **machine learning** applications like neural networks.

PyTorch

PyTorch is an open source **machine learning** library for Python, based on Torch. It is used for applications such as **natural language processing** and was developed by Facebook's AI research group.

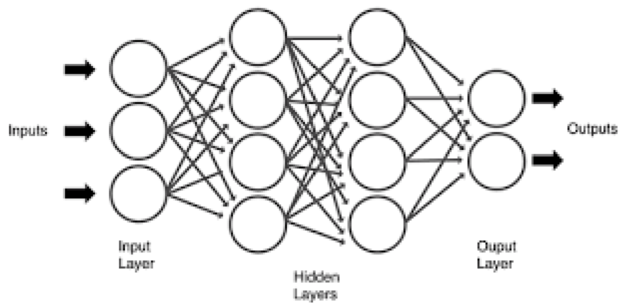
To begin my research, I must decide which framework I was going to use to create my CNN model, these 3 are the most common machine learning frameworks in Python.

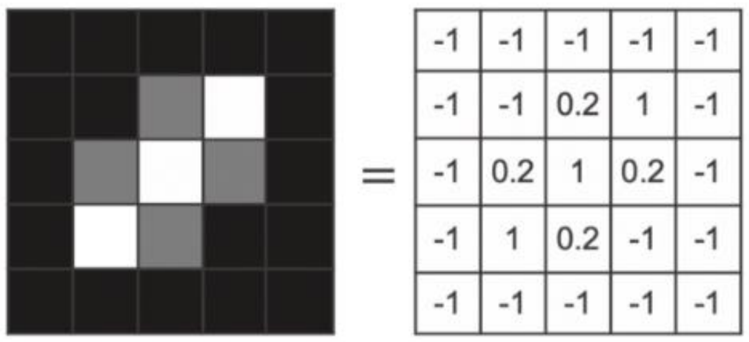
After further thought, I began leaning towards the TensorFlow framework due its compatibility to the media pipe hands pipeline previously added. TensorFlow is known as googles machine learning library however looking at processing times:



TensorFlow is extremely slow as a CNN model making library therefore, since Keras can be used alongside TensorFlow, the model will be built using the Keras library and then integrated into TensorFlow due to compatibility with the media pipe google library.

The deep-learning model will be laid out with a discrete number of inputs with hidden layers between and an output layer in the end with two nodes. The way in which machines learn can be seen as a process of trial and error, but in fact its regression. An image, known as a training image, is input on the left most node of the Convolutional Neural Network.





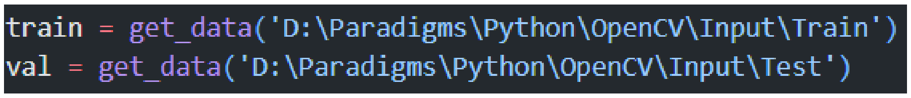
-1	-1	-1	-1	-1
-1	-1	0.2	1	-1
-1	0.2	1	0.2	-1
-1	1	0.2	-1	-1
-1	-1	-1	-1	-1

The training images run through the hidden layers as gridded image objects which get weighted depending on the colour density present within the image. This method is run over and over until an accurate weighting of each labelled training image is created.

Once the CNN model can assign these weighted values to a label, test images are passed through, and the network is rewarded for true values and altered for images it labelled incorrectly.

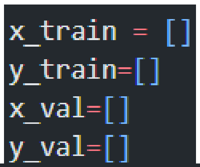
To begin creating my CNN, I must first be able to find an available and premade large dataset of signed characters. [In Bibliography as Dataset] The dataset created by loicmarie for their own translation model will be perfect for creating and testing my own model, although the images in this dataset aren’t in BSL, they will allow me to train a model with a dataset which can later be repeated with my own BSL dataset.

For the CNN test I will be simply creating and testing the A and B datasets.



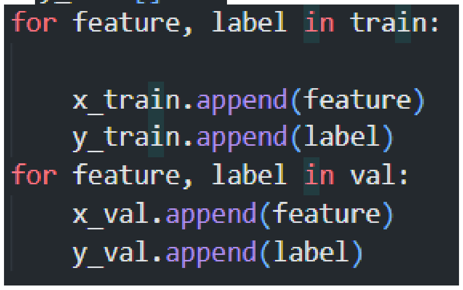
```
train = get_data('D:\Paradigms\Python\OpenCV\Input\Train')
val = get_data('D:\Paradigms\Python\OpenCV\Input\Test')
```

I get all the data from the Train and Test folder and place them in corresponding folders. Each character has 3000 images and they have been split in a 2500:500 ratio of training and testing respectively.



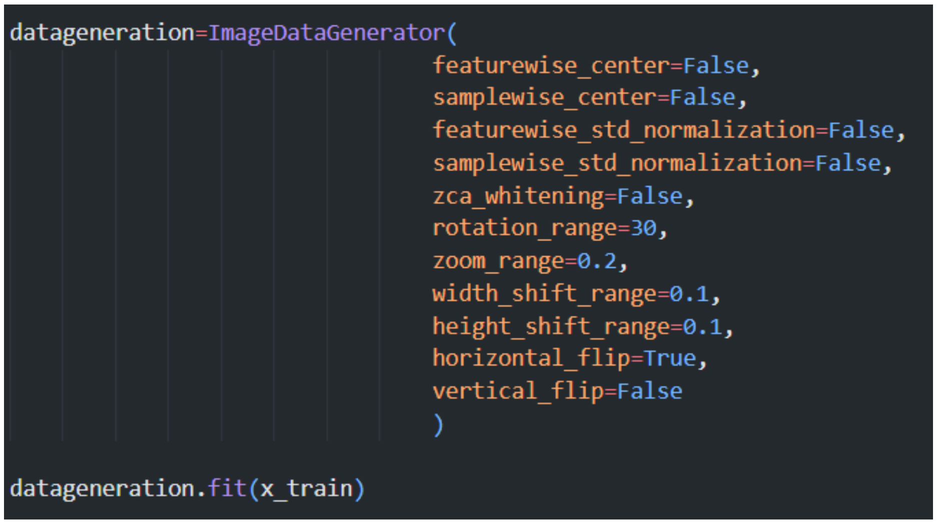
```
x_train = []
y_train=[]
x_val=[]
y_val=[]
```

I create some lists which will store the labels of training and test images and the features which these labels relate two. I labelled them x and y since these can be visualized as a graph.



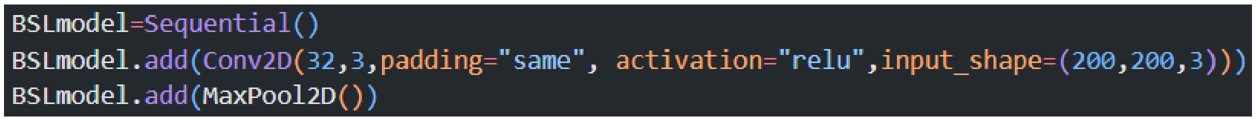
```
for feature, label in train:

    x_train.append(feature)
    y_train.append(label)
for feature, label in val:
    x_val.append(feature)
    y_val.append(label)
```

I proceed to append the necessary features and image labels to the new lists which I have just created. Instead of using a two separate lists for features and labels I could have made a matrix however this provides me when an easier way of visualizing my code.



```
datageneration=ImageDataGenerator(
                                   featurewise_center=False,
                                   samplewise_center=False,
                                   featurewise_std_normalization=False,
                                   samplewise_std_normalization=False,
                                   zca_whitening=False,
                                   rotation_range=30,
                                   zoom_range=0.2,
                                   width_shift_range=0.1,
                                   height_shift_range=0.1,
                                   horizontal_flip=True,
                                   vertical_flip=False
                                   )

datageneration.fit(x_train)
```

To improve the results created by the model image have to be inputted in multiple ways. Kera has an inbuilt Image Data Generator method which allows me to changes aspects of training images randomly. This change can be rotation, flipping or widening. This essentially means that even though the dataset of images are very similar, meaning that the model could possibly adapt to that singular type of skin colour, size and shape, this generator allows for a variety of training. We fit this new data generation variable to the training images.

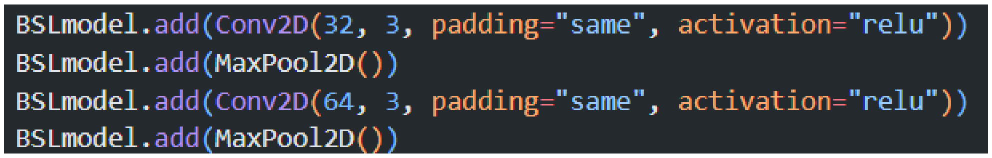


```
BSLmodel=Sequential()
BSLmodel.add(Conv2D(32,3,padding="same", activation="relu",input_shape=(200,200,3)))
BSLmodel.add(MaxPool2D())
```

I begin to create my CNN, I define the BSL model as a new sequential model and create the first layer which will consist of 32 neurons. The input shape refers to the image details such as x,y, colour depth.

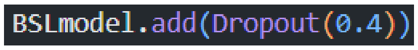
Although this is slightly more in depth, the rectified linear unit activation or relu for short provides positive integer values to the next neuron. The results of the processing of the neurons can provide negative results which are no good for the proceeding neurons. The relu activation will be used in my hidden layers as well.

The max pooling method restricts information to the necessary information required in a 2D plane of an image rather than focusing on unnecessary details. It has been proven to increase the accuracy of CNN models.



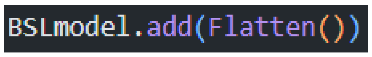
```
BSLmodel.add(Conv2D(32, 3, padding="same", activation="relu"))
BSLmodel.add(MaxPool2D())
BSLmodel.add(Conv2D(64, 3, padding="same", activation="relu"))
BSLmodel.add(MaxPool2D())
```

I create two hidden layers with the same structure as the initial however the first half of the hidden layer contains 32 neurons whereas the second contains 64 neurons instead.



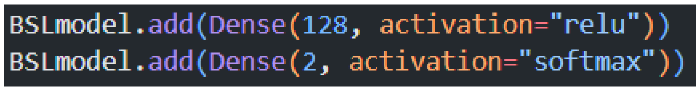
```
BSLmodel.add(Dropout(0.4))
```

The dropout method allows me to control anomalous results passed through the layers before they reach the final layer and get added to the weightings of the model. It essentially restricts the values to 0.6 of a deviation from the mean value, providing a more accurate result.



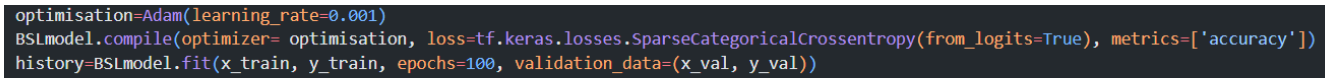
```
BSLmodel.add(Flatten())
```

The flatten method reduces the 2-dimensional plane data that I've been using into a single dimension so that the weightings can be passed on as a weighted image.



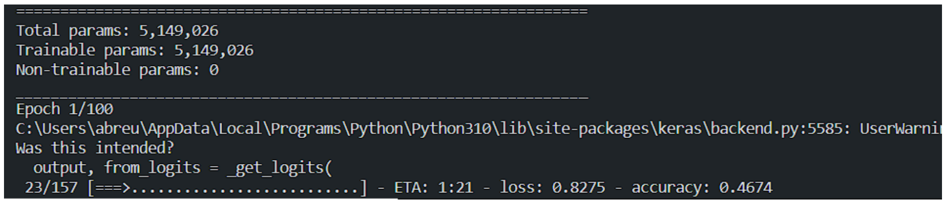
```
BSLmodel.add(Dense(128, activation="relu"))
BSLmodel.add(Dense(2, activation="softmax"))
```

I pass the images through one final layer and output. Depdning on whther the CNN has classified it as A or B will depend on which neuron has been fired. The softmax activation is required in the final layer since this is what converts the outputs from the neutrons into vector probabilities.



```
optimisation=Adam(learning_rate=0.001)
BSLmodel.compile(optimizer= optimisation, loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])
history=BSLmodel.fit(x_train, y_train, epochs=100, validation_data=(x_val, y_val))
```

The Adam optimization is the most used optimizer for CNN models, so I opted for this as it seemed to work for most models. The learning rate is set to very low so that the accuracy is better, since the machine is taking a longer time to process the data in the image through the CNN. I compile the model which begin to train and create weightings from the images. The cross-entropy method will create a quantitative value for lost data and inaccuracies. The last line allows me to view the training data on the command line, unfortunately this is where I decided to switch methods of development. When visualizing the epochs (cycles of learning), a timer stated that training just two folder was going to take up to a few hours at the rate

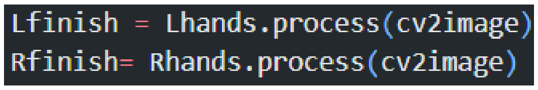


```
=================================================================
Total params: 5,149,026
Trainable params: 5,149,026
Non-trainable params: 0
_________________________________________________________________
Epoch 1/100
C:\Users\abreu\AppData\Local\Programs\Python\Python310\lib\site-packages\keras\backend.py:5585: UserWarni
Was this intended?
  output, from_logits = _get_logits(
 23/157 [===>..........................] - ETA: 1:21 - loss: 0.8275 - accuracy: 0.4674
```

it was going. After waiting for a few hours, it did in fact finish however the model only managed to create an accuracy of 63%. For a viable model I would need upwards of 80%, and I would need to train 26 different folders of character data which would take even longer. This project has time constraints which have made me forcibly switch to the heuristic method so that the project can be finished in time. If I were to have another iteration of development, I would like to make the model work with an accuracy of 80% and above to integrate into the window.

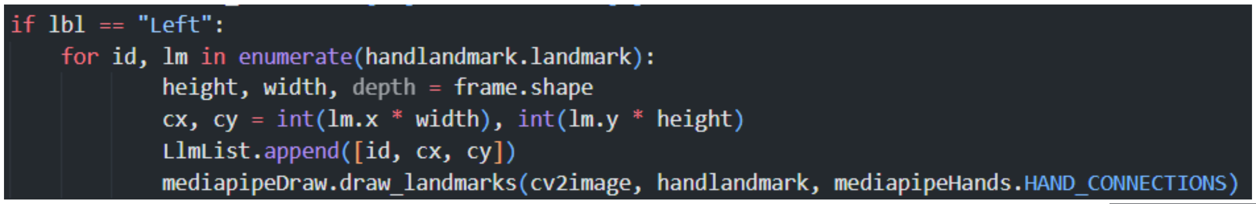
Heuristic Approach

Training of the model was a time intensive task which did not fit inside the time frame set by the project, so I decided to try out the heuristic approach. This method will use the nodes data from the media pipe hands model and calculate distances between these to show the characters.

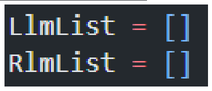


```
Lfinish = Lhands.process(cv2image)
Rfinish= Rhands.process(cv2image)
```

I begin by creating two separate hand detection models, creating one model for the detection for two hands causes the nodes to remain the same on either hand meaning that node locations are only printed for one hand.

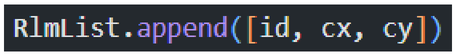


```
if lbl == "Left":
    for id, lm in enumerate(handlandmark.landmark):
            height, width, depth = frame.shape
            cx, cy = int(lm.x * width), int(lm.y * height)
            LlmList.append([id, cx, cy])
            mediapipeDraw.draw_landmarks(cv2image, handlandmark, mediapipeHands.HAND_CONNECTIONS)
```



```
LlmList = []
RlmList = []
```

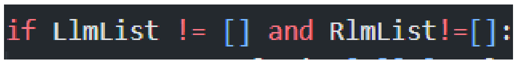
I use the multi handedness classification label that I added when implementing the pipeline to filter out whether the detected hand is left or right. I now extract the node X, Y values [Bibliography reference under hand tracking module] Depending on the side detected, I iterate through the number of landmark nodes detected in the image and get the frame data, this includes the X, Y and depth of colour I the image. Using this data and the X, Y locations of every node I'm iterating through individually, I create a central positions of the identified nodes in the hand detection. When the central X,Y locations are found I append the node ID, central x value and central y value to the Left landmark list. This code is identical for the right label; however, I append the values into a right landmark list instead.



```
RlmList.append([id, cx, cy])
```

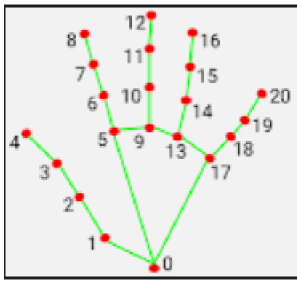
The pipeline display code remains the same, using the media pipe drawing utility solutions to display interconnected nodes for the user on the live feed.

For ease of comparison, I now extract the values for all nodes and place them inside variables with recognizable names. I first must check if both the Left and Right landmark lists contain data, if this is not true then detection via node proximity is not possible.

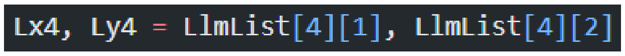


```
if LlmList != [] and RlmList!=[]:
```

Both lists extend to a size of 21 relating to the 21 nodes displayed on a media pipe hand model.

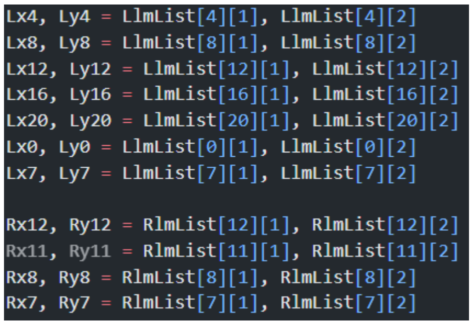


Since the index for the hand model begins at 0, the list will perfectly match the X, Y value that were inputted meaning that no shifting of numbers relative to the node will be required. What I mean by this is that location [4][1] will be the tip of the thumbs X location, if this hand model began at 1 then this location would be node 3.



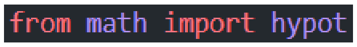
```
Lx4, Ly4 = LlmList[4][1], LlmList[4][2]
```

I created a naming system for these location, first letter referring to the hand label, second referring to whether its either X or Y value and the final its node id. I input the X and Y values for node 4 in their respective variables an repeat this process with all the nodes that are required. To test whether this works I have done this for a hand full of nodes rather than all nodes. To improve the program, I hope to make this process more efficient by creating these variables with a loop rather than hard coded. These hard coded values are repetitive and take up greater amount of line space.



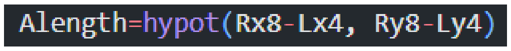
```
Lx4, Ly4 = LlmList[4][1], LlmList[4][2]
Lx8, Ly8 = LlmList[8][1], LlmList[8][2]
Lx12, Ly12 = LlmList[12][1], LlmList[12][2]
Lx16, Ly16 = LlmList[16][1], LlmList[16][2]
Lx20, Ly20 = LlmList[20][1], LlmList[20][2]
Lx0, Ly0 = LlmList[0][1], LlmList[0][2]
Lx7, Ly7 = LlmList[7][1], LlmList[7][2]

Rx12, Ry12 = RlmList[12][1], RlmList[12][2]
Rx11, Ry11 = RlmList[11][1], RlmList[11][2]
Rx8, Ry8 = RlmList[8][1], RlmList[8][2]
Rx7, Ry7 = RlmList[7][1], RlmList[7][2]
```

Using the math module, I import the hypotenuse function to work out the relative distance between nodes.

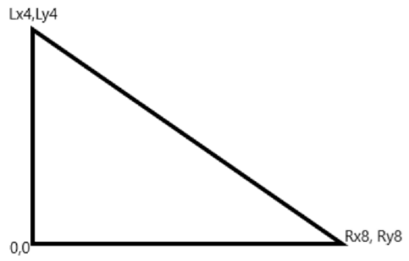


```
from math import hypot
```

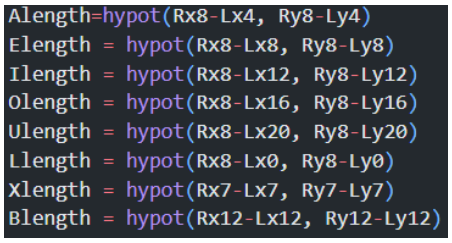
As an example, the sign for A in BSL is the index finger touching the tip of the thumb. In terms of a pipeline, this can be viewed and index number 8 on the right hand being within a certain distance of index number 4 on the left hand



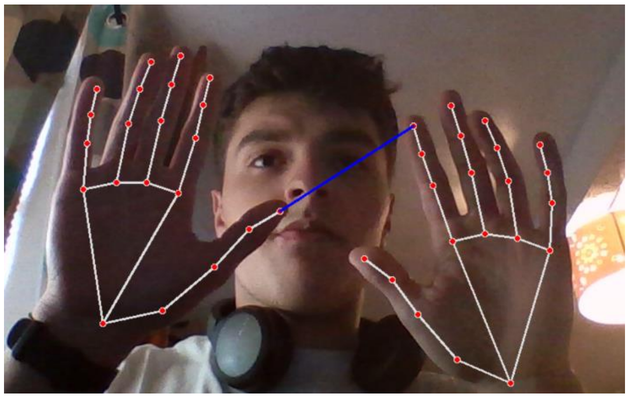
```
Alength=hypot(Rx8-Lx4, Ry8-Ly4)
```



Using this logic, I can use the position and create a triangle of relative distance. The hypotenuse will work out the distance that is useful to me. This value is assigned as the A length value, I repeat this process with some other characters to test the functionality of this calculation within the program.



```
Alength=hypot(Rx8-Lx4, Ry8-Ly4)
Elength = hypot(Rx8-Lx8, Ry8-Ly8)
Ilength = hypot(Rx8-Lx12, Ry8-Ly12)
Olength = hypot(Rx8-Lx16, Ry8-Ly16)
Ulength = hypot(Rx8-Lx20, Ry8-Ly20)
Llength = hypot(Rx8-Lx0, Ry8-Ly0)
Xlength = hypot(Rx7-Lx7, Ry7-Ly7)
Blength = hypot(Rx12-Lx12, Ry12-Ly12)
```



Once this was completed I had to execute events once a certain condition was met, using multiple if statements that contain different weightings depending on the required hand position is what I settled with.

```
if Alength<25:
elif Llength<75:
elif Blength<50:
```

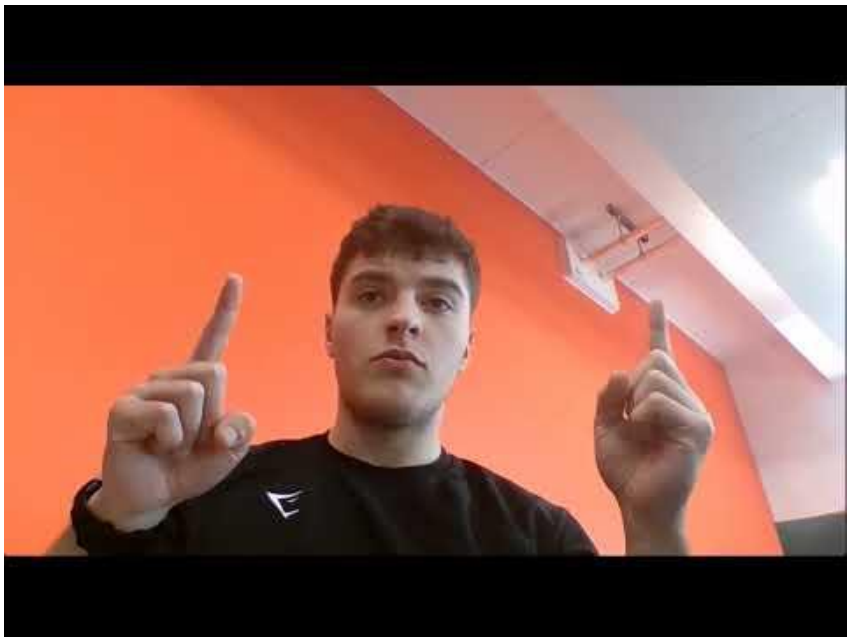
Weightings are dependent on which nodes were selected to represent the character on screen. For example, the L length weighting must be greater than that for the rest of the character since the L characters true signing depends on an index in the center of the palm, the media pipe hand model does not have anode at the center of the palm, so a larger hypotenuse must be taken into account for detection.

Once the condition has been met, we display a character on screen to tell the user that their signing has been detected correctly by the program.

```
cv2.putText(cv2image, "A",(100, 100), cv2.FONT_HERSHEY_SIMPLEX, 3, (0, 0, 0), 4, cv2.LINE_4)
```

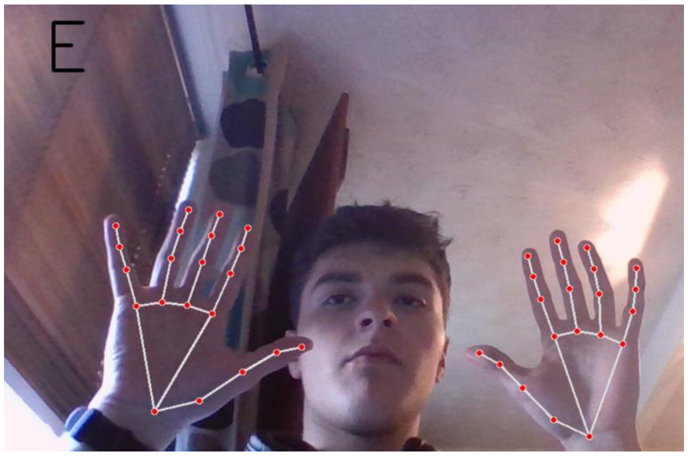
The 100, 100 refers to the x ,y position form the top left corner, the rest simply dictates the look of the text which is not important for the actual display of the character.

Module Testing



I created the module in a separate file so that I could test the BSL module separately. In this case, I didn't implement the pipeline that will be visible to the user however the video still provides an idea of what the program is no capable of.

When the nodes for A become close enough the character is correctly displayed in the Top Left, this works for a multitude of the characters however there are currently some troubles with X. The model has trouble detecting the full hand once they overlap and therefore you get this flashing of detection, this happens when the hand is continuously detected and undetected while the X hand position is maintained.



The weighting values seem to be working correctly, L can be correctly recognized even though the node is slightly South of where the sign is supposed to take place. The biggest problem I'm currently facing with this approach is when the separate hand models detect the same hand. This causes the X,Y positions to be set to the same values. At the beginning you can see that this happens meaning that all characters are displayed at the same time, the only current fix for this problem is to move a hand out of frame and bring it back.

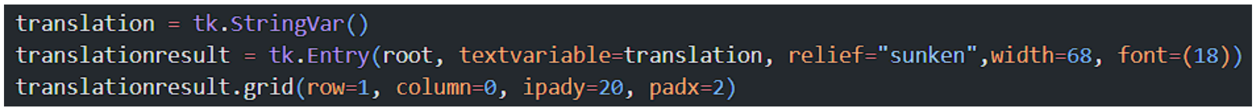
This solution still has a few bugs such as the same hand detection and the sensitivity to light that it experiences. However, that can be improved in later iterations of the program. Overall, it is a good solution to a problem that otherwise wouldn't have a solution without the CNN machine learning approach.

The next steps involve tweaking small settings with media pipe to allow a greater leniency of detection of hands without compromising the validity of the recognition. If an optimal point is found, it will reduce the glare problems that currently occur in the program.

User Feedback

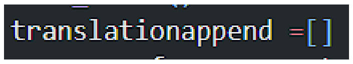
MILESTONE 8 – CHARACTER DISPLAY

Objective of milestone: Find a method to display detected characters in the GUI widgets.

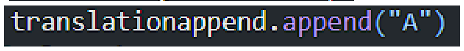


```
translation = tk.StringVar()
translationresult = tk.Entry(root, textvariable=translation, relief="sunken",width=68, font=(18))
translationresult.grid(row=1, column=0, ipady=20, padx=2)
```

The Tkinter translation result widget, which has been created earlier in the program, will be the central point for displaying signed characters to the user. It is visible quite prominently from the main window and will take the values signed and store them inside the translation variable. Any data within the translation variable will be displayed in the translation result widget since it classifies the translate variable as its text variable.

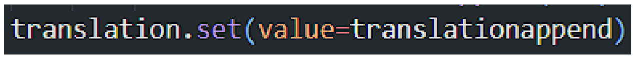


```
translationappend =[]
```



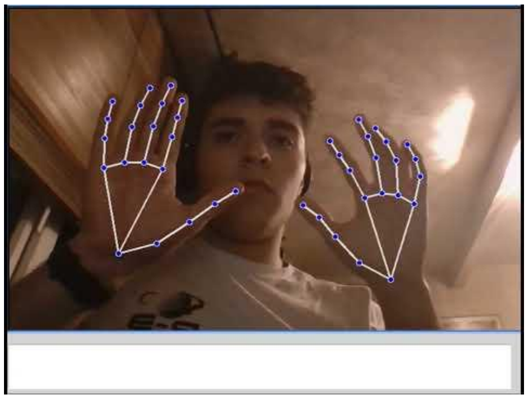
```
translationappend.append("A")
```

I create a list to append to which can then be set as the value within translation. When the specific character condition is met, and the character is displayed don the screen I also append the value to the translation append list



```
translation.set(value=translationappend)
```

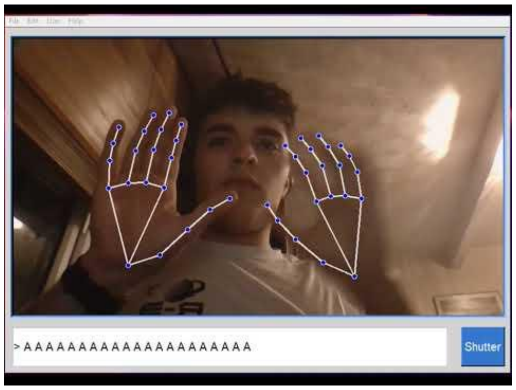
At the end of the cycle, I set the translation value to the contents stores within the append list.



As shown in the video, there is a problem with the appending. Since the show frame functions is in an endless loop within a thread, the append will only last for as long as the condition is met therefore some of the logic has to be changed.

```
def main():
    translationlist=[">"]
```

The new list that is to be appended to is defined directly under the main call to avoid the problem I was having before of the list being emptied every time the loop executed, I also added a better name convention and a preset value so that I can check whether the same issue is occurring when the problem loads.


```
>AAAAAAAAAAAAAAAAAAAAA
```

Although the previous list deletion has been fixed, the detection is now being looped meaning that one detection causes multiple characters to be shown. This is because the refresh is so fast.

As a temporary fix I implement some code which doesn't allow two of the same characters to be adjacent to each other.

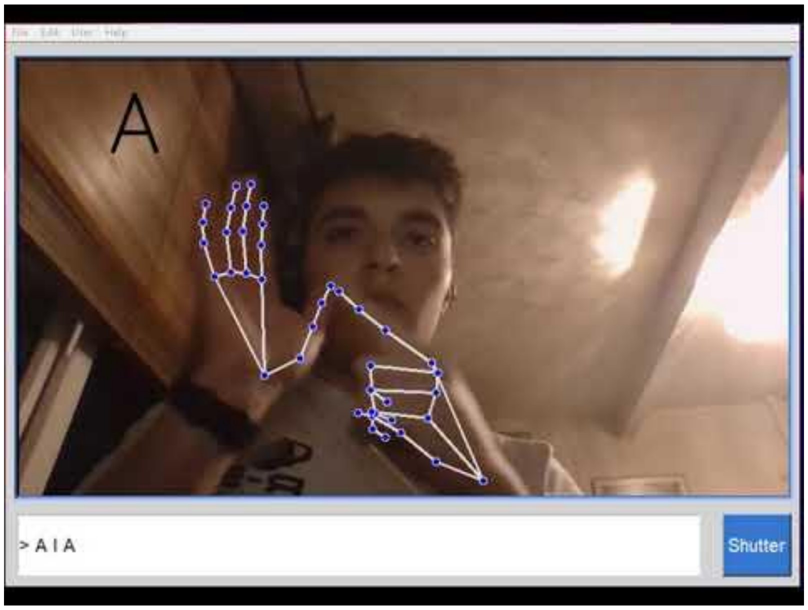
```
if translationlist[-1]!="A":
```

This code is simply a piece of comparison code which checks the final value of the list against the character that is being detected. If the two are the same, then the character will not be appended however if they do not match then the character will be appended.

```
elif Olength<25:
    cv2.putText(cv2image, "O",(100, 100), cv2.FONT_HERSHEY_SIMPLEX, 3, (0, 0, 0), 4, cv2.LINE_4)
    if translationlist[-1]!="O":
        translationlist.append("O")
```

This is some example code for the Character O and what it looks like when it is all put together.

Module Testing



The module has come together well and I encountered less problems than I expected when creating a method for character display. The character all display correctly however there is some mistakes but that is mostly due to inaccuracies in the hand model detection, also due to the amount of light exposure in this particular frame.

The temporary fix for looping characters is not perfect but can be improved in the next iterations of the program.

My next goal is to add more character detections with the model and expand the range of possible characters able to be displayed in the program which will make for a fuller and better program.

User Feedback

It is difficult to get the correct letter sometimes but most of the time it is accurate. The letters that I sign are very visible at the bottom and I can clearly see what I am signing out.

MILESTONE 9 – ADDING CREDENTIALS

Objective of milestone: Create a method for users to add credentials to login with

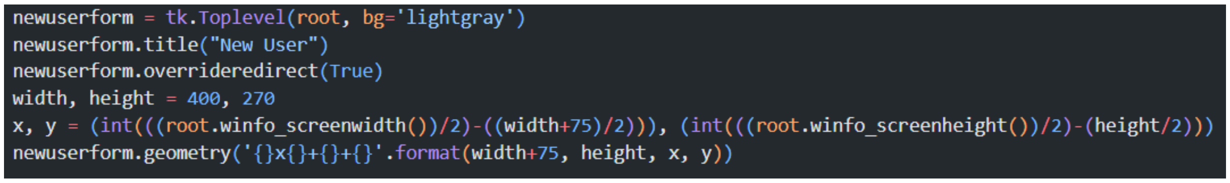
Explanation of code:

```
newuserform = tk.Tk()
```

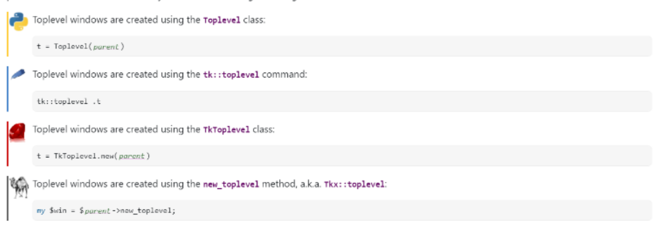
Creating the sister window

When I first began creating this milestone, I defined the new sister window as a Tkinter Tk instance however after gridding widgets onto the new window errors were being created when both Tk windows were being run alongside each other. The widgets weren't being correctly packed because the master root window wasn't specified. Since this was a process which ultimately caused confusion in my program, I decided to research more viable options.

Using online official Tkinter documentation, referenced in the bibliography, I found the top level method which allowed me to create sister windows from the root top level window. Creating this top-level followed a similar pattern to how the rest of the widgets were created.



```
newuserform = tk.Toplevel(root, bg='lightgray')
newuserform.title("New User")
newuserform.overrideredirect(True)
width, height = 400, 270
x, y = (int(((root.winfo_screenwidth())/2)-((width+75)/2))), (int(((root.winfo_screenheight())/2)-(height/2)))
newuserform.geometry('{}x{}+{}+{}'.format(width+75, height, x, y))
```



Toplevel windows are created using the **Toplevel** class:

```
t = Toplevel(parent)
```

Toplevel windows are created using the **tk::toplevel** command:

```
tk::toplevel .t
```

Toplevel windows are created using the **TkToplevel** class:

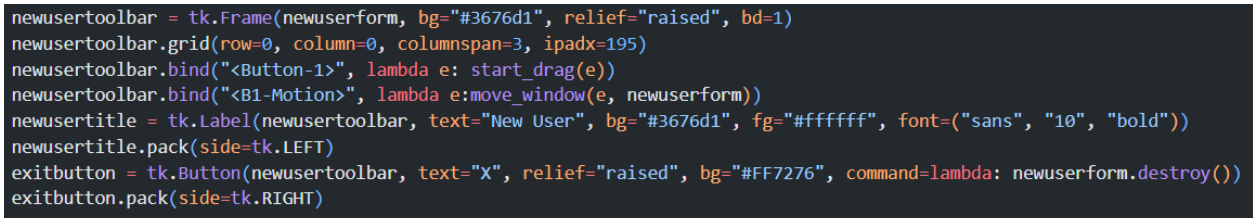
```
t = TkToplevel.new(parent)
```

Toplevel windows are created using the **new_toplevel** method, a.k.a. **Tkx::toplevel**:

```
my $win = $parent->new_toplevel;
```

First, I define the tk top-level window and assign it a window name. This top-level window will be assigned the Tkinter root as its master. Using this top-level method means that widgets will not pop-up errors due to master issues. Multiple instances of windows can also be created and since all will fall under the root if a user happens to have multiple child windows open you can close all of them from a single central point. The overridereddirect method allows me to get rid of the windows default toolbar created on all windows, this will be replaced by the toolbar we created earlier in the project for the login form.

Next, I defined the arbitrary width and height. It was not calculated to a certain measurement but rather to a size that I thought was reasonable. Using the same centering method as for my other windows, I take the width of the screen divide it by 2 and take away the width of the window divided by 2. Using this for both height and width centers windows onto a screen perfectly regardless of the resolution.

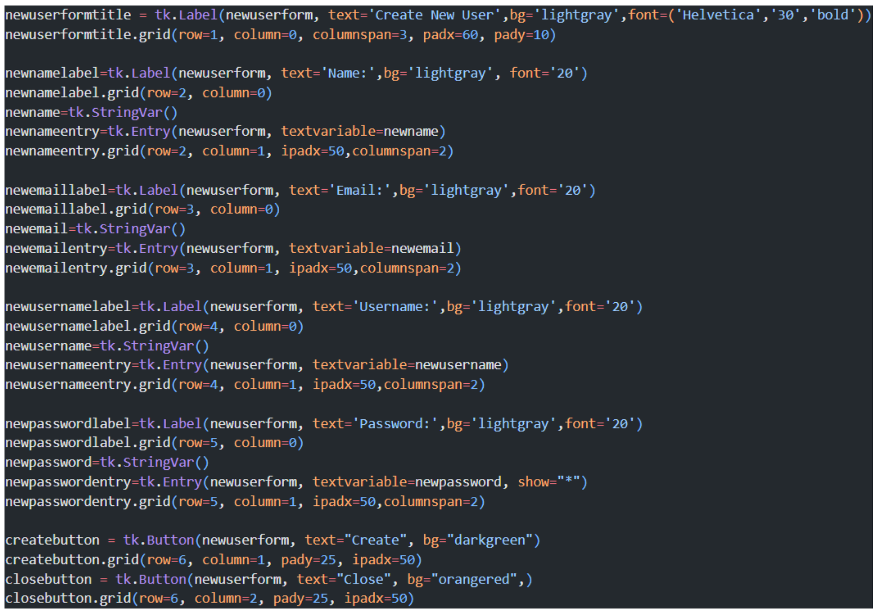


```
newusertoolbar = tk.Frame(newuserform, bg="#3676d1", relief="raised", bd=1)
newusertoolbar.grid(row=0, column=0, columnspan=3, ipadx=195)
newusertoolbar.bind("<Button-1>", lambda e: start_drag(e))
newusertoolbar.bind("<B1-Motion>", lambda e:move_window(e, newuserform))
newusertitle = tk.Label(newusertoolbar, text="New User", bg="#3676d1", fg="#ffffff", font=("sans", "10", "bold"))
newusertitle.pack(side=tk.LEFT)
exitbutton = tk.Button(newusertoolbar, text="X", relief="raised", bg="#FF7276", command=lambda: newuserform.destroy())
exitbutton.pack(side=tk.RIGHT)
```

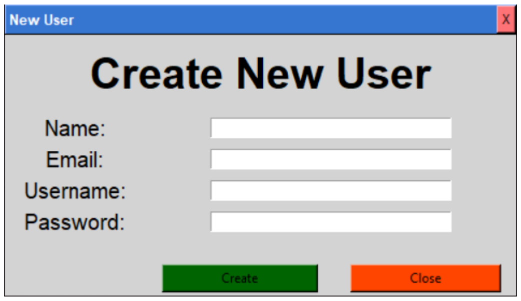
This is the code for the new user toolbar, explanation of how it functions can be found under the creation of the login form GUI

New User Window Widgets

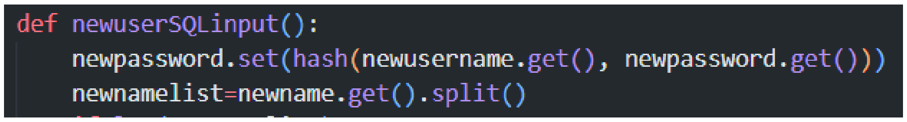
This makes up the bulk of the code, it simply places widgets onto the sister window following the layout plans specified in the design stage. The text variables will be passed on to the module which will pass all this data into the SQL tables.



```
newuserformtitle = tk.Label(newuserform, text='Create New User',bg='lightgray',font=('Helvetica','30','bold'))
newuserformtitle.grid(row=1, column=0, columnspan=3, padx=60, pady=10)

newnamelabel=tk.Label(newuserform, text='Name:',bg='lightgray', font='20')
newnamelabel.grid(row=2, column=0)
newname=tk.StringVar()
newnameentry=tk.Entry(newuserform, textvariable=newname)
newnameentry.grid(row=2, column=1, ipadx=50,columnspan=2)

newemaillabel=tk.Label(newuserform, text='Email:',bg='lightgray',font='20')
newemaillabel.grid(row=3, column=0)
newemail=tk.StringVar()
newemailentry=tk.Entry(newuserform, textvariable=newemail)
newemailentry.grid(row=3, column=1, ipadx=50,columnspan=2)

newusernamelabel=tk.Label(newuserform, text='Username:',bg='lightgray',font='20')
newusernamelabel.grid(row=4, column=0)
newusername=tk.StringVar()
newusernameentry=tk.Entry(newuserform, textvariable=newusername)
newusernameentry.grid(row=4, column=1, ipadx=50,columnspan=2)

newpasswordlabel=tk.Label(newuserform, text='Password:',bg='lightgray',font='20')
newpasswordlabel.grid(row=5, column=0)
newpassword=tk.StringVar()
newpasswordentry=tk.Entry(newuserform, textvariable=newpassword, show="*")
newpasswordentry.grid(row=5, column=1, ipadx=50,columnspan=2)

createbutton = tk.Button(newuserform, text="Create", bg="darkgreen")
createbutton.grid(row=6, column=1, pady=25, ipadx=50)
closebutton = tk.Button(newuserform, text="Close", bg="orangered",)
closebutton.grid(row=6, column=2, pady=25, ipadx=50)
```



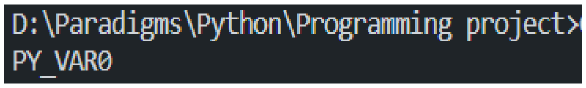
New User SQL Input



```
def newuserSQLinput():
    newpassword.set(hash(newusername.get(), newpassword.get()))
    newnamelist=newname.get().split()
```

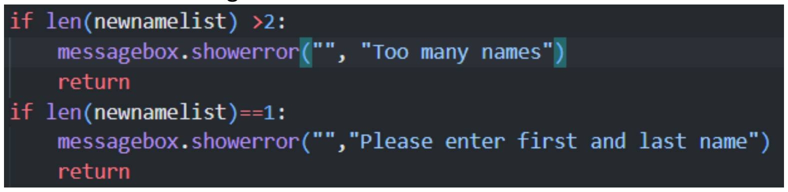
I begin by hashing the new password entered into the new-user form by using the hashing created earlier in the project for the login form. This hashing algorithm uses the username entered by the user as the key for encryption.

Since Tkinter variables are encapsulated I must use the set and get methods to access their plain values, if I do not value to not resemble what is inputted by the user.



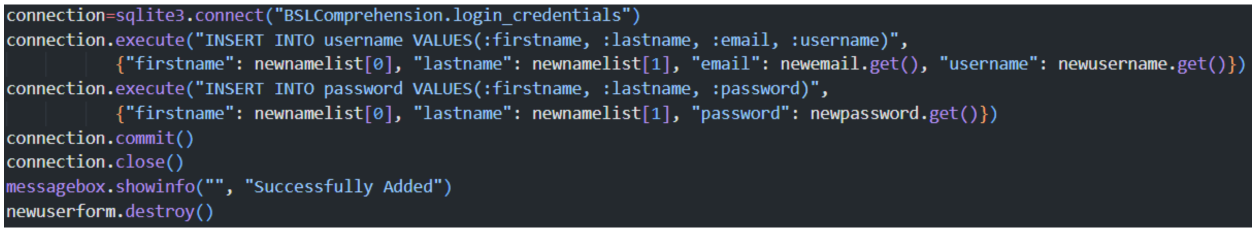
```
D:\Paradigms\Python\Programming project>
PY_VAR0
```

Since the newname variable may contain more than one two names and a minimum of two, I use the split method which separates each word at every break to create a list that I can iterate through to input the data into the SQL tables. New name list location 0 will hold the users firstname and location 1 will hold the surname of the user.



```
if len(newnamelist) >2:
    messagebox.showerror("", "Too many names")
    return
if len(newnamelist)==1:
    messagebox.showerror("","Please enter first and last name")
    return
```

As a method of input sanitization, I check that the user has input at least two names, surname and firstname as the program requires this information. If the user inputs a single name a message box error will appear stating to the user that a first and last name must be entered. By adding return at the end of these statements I make sure that the function doesn't proceed to add misinformation to the SQL table.

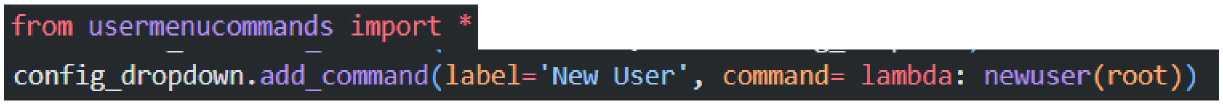


```
connection=sqlite3.connect("BSLComprehension.login_credentials")
connection.execute("INSERT INTO username VALUES(:firstname, :lastname, :email, :username)",
          {"firstname": newnamelist[0], "lastname": newnamelist[1], "email": newemail.get(), "username": newusername.get()})
connection.execute("INSERT INTO password VALUES(:firstname, :lastname, :password)",
          {"firstname": newnamelist[0], "lastname": newnamelist[1], "password": newpassword.get()})
connection.commit()
connection.close()
messagebox.showinfo("", "Successfully Added")
newuserform.destroy()
```

I establish a connection with the credentials database by using the sqlite3 connect method, once a connection is established, I can proceed to input the new information. Since I will not be transferring any information from the table into the program, I will not require a cursor object like previously needed and can instead perform command directly using the connection.

Using the execute method I write some SQL to input the Tkinter variable values into the designated tables. As previously mentioned, new name list location 0 refers to the firstname of the user and location 1 to the lastname. For the encapsulated variables, I use the get method and repeat the process for the password table.

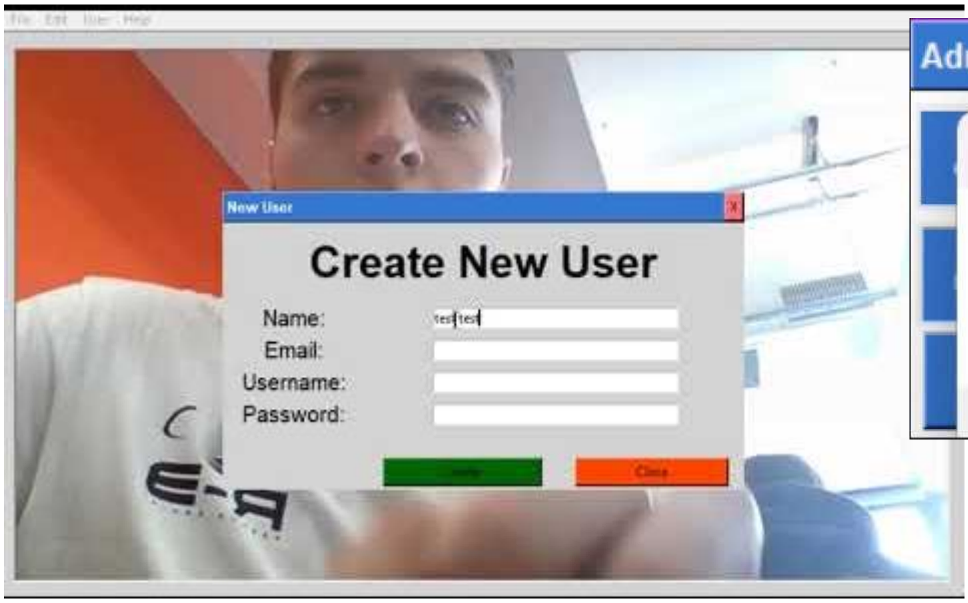
All changes are saved into the table by the commit method and the connection between program and database is closed. Once the information is successfully added a message box appears to the user to confirm that the addition of credentials has been successful, and the new user form is destroyed. Once the new user form is destroyed the main program will once again be visible again.

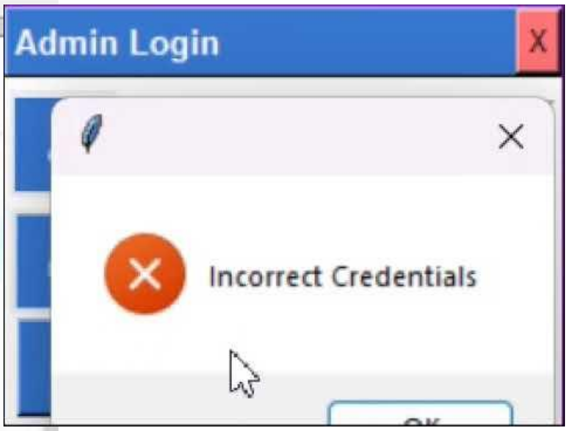


```
from usermenucommands import *
config_dropdown.add_command(label='New User', command= lambda: newuser(root))
```

Since the new user module is now complete, I return to the menu file and import all functions from the user menu commands file. Although the program currently only has one function, it will save me some time once I add more commands to the program. I search for the dropdown in which the new user will fall under and add a new command. I simply call the new user module we just created and pass along the root window so that the GUI can be created. The lambda command is an in-built method in python, it means that the command, connected to this menu which essentially acts as a Tkinter button, will only execute once the button press event occurs. Not using the command would mean that once the program is loaded all the command would be executed. This is the standard way of handling events in python.

Module Testing





These videos cover testing of the module. The menu command for opening the new user form is working as intended and is responsive from the user end. The form loads as a top-level window; however, the camera live feed is still visible.

I fill the entry boxes with test credentials, making sure to leave the name entry with a single name to test the sanitization. As expected, the program returns a message box error and closes the form. I proceed to input the correct information in the correct form and press the create button at the bottom of the form. The program returns the successful input message and closes, displaying the main program as intended.

The right clip showcases that the information has in fact been committed to the database and now works with the login form. To test the login forms input sanitization with this new data, I input the incorrect username and try to authorise the process. An error message box appears. Finally, I input the new credentials in the correct form and as expected the login form authorizes the new data and closes down to load the main program.

I'm very pleased with the results of this criteria and will continue to use this method to create the other commands in the navigation menu. The forms work well alongside the main window and don't reduce performance of the live feed.

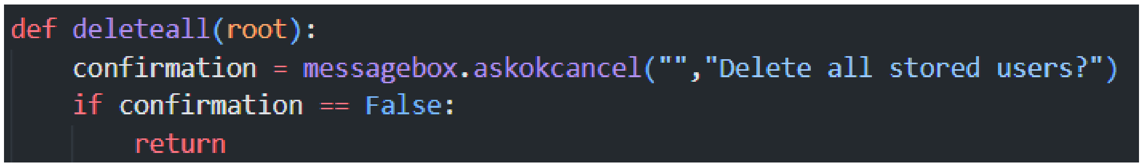
User feedback

I would like if the program didn't close once the new user was made, it makes for a slow way of doing things. The window though is user friendly and intuitive to use, login form updated straight away with the new user information.

MILESTONE 10 – DELETE ALL CREDENTIALS

Objective of milestone: Allow user to delete all user credentials from database

Explanation of code:

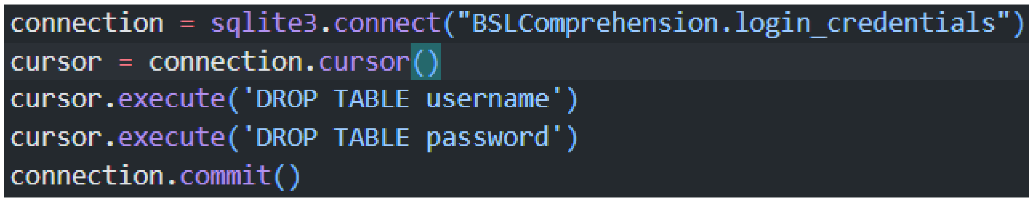


```
def deleteall(root):
    confirmation = messagebox.askokcancel("","Delete all stored users?")
    if confirmation == False:
        return
```

I create the delete all module that will take in the root as a parameter, this is so that once the deletion occurs the program will be exited and the user will have to login once again.

Since this module can cause problems for the user, I add a confirmation message box once the module is run. It will take in a Boolean value which will either confirm or deny the action, if False is returned the action will be immediately canceled by returning out of the function.

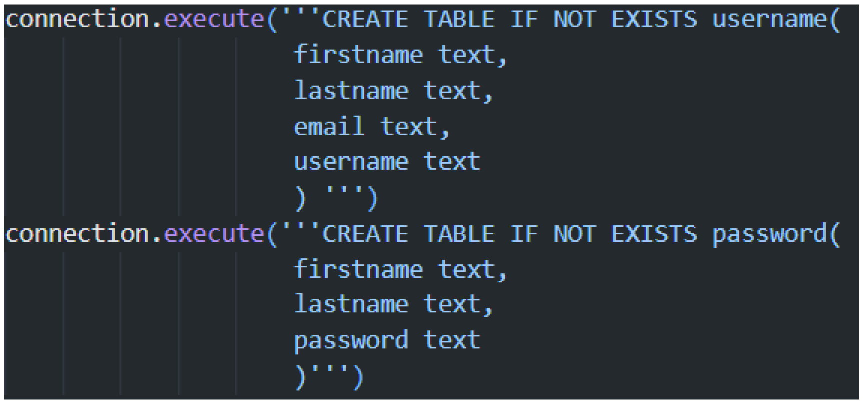
Connection to database



```
connection = sqlite3.connect("BSLComprehension.login_credentials")
cursor = connection.cursor()
cursor.execute('DROP TABLE username')
cursor.execute('DROP TABLE password')
connection.commit()
```

Again, when using sqlite3 I must establish a connection between the program and the database to be able to make any kind of change. Since we are directly accessing the tables I create a cursor for executing my SQL. I then execute two drop table commands, for username and password respectively, which deleted every single piece of data in each one of the tables. I proceed to commit these changes meaning that the database now contains no credentials.

Recreating the tables



```
connection.execute('''CREATE TABLE IF NOT EXISTS username(
                    firstname text,
                    lastname text,
                    email text,
                    username text
                    ) ''')
connection.execute('''CREATE TABLE IF NOT EXISTS password(
                    firstname text,
                    lastname text,
                    password text
                    )''')
```

Now that all the information has been removed, I must create the tables again using the SQL command that I originally used to create the tables. The code is identical, and the tables will include all the fields that they did before.

At this stage the tables have been successfully replaced in the database meaning that the suer data wipe has been successful. However, there are currently no available credentials that the user may use to login again after the wipe therefore I create default login credentials which the user will use to login after every successful wipe. These default credentials will simply be 'master'.

Adding default credentials

```
connection.execute("INSERT INTO username VALUES(:firstname, :lastname, :email, :username)",
                   {"firstname": "master", "lastname": "master", "email": "master@gmail.com", "username": "master"})

connection.execute("INSERT INTO password VALUES(:firstname, :lastname, :password)",
                   {"firstname": "master", "lastname": "master", "password": "b'w5rDgsOmw6jDisOk'"})
```

I insert the master values into the tables using SQL that has been previously seen before in the creation of the table. The password is inputted hashed.

```
connection.commit()
connection.close()
root.destroy()
```

I commit these changes to database and close the connection between the program and the tables. I proceed to destroy the root so that the user must login in with the new credentials. The program will have to be reopened

```
config_dropdown.add_command(label='Erase all Users', command=lambda: deleteall(root))
```

I then added the command to the corresponding dropdown in the menu file

Module Testing

I begin by testing to see whether the message box cancel is functioning as intended, it does indeed cancel the delete all function returning the user back to the main program.

Next, I delete all the current stored information which includes the previous test data that I inputted. Once the confirmation button has been pressed the program closes and has to be reloaded. To check that the credentials have been successfully deleted by the module I enter the test credentials into the login form. This produces an error message, meaning that the data is no longer available to authenticate the login. Finally, I input the master default credentials which authorise the login. At the end of the clip the program can be seen loading up.

I tried to reopen the program via the code file so that the suer wouldn't have to manually restart the program however this caused issues with circular calling. Since the menu file is called in main, main cannot be called in the menu file. This method is enough for the end user and meets the needs of the milestone.

User Feedback

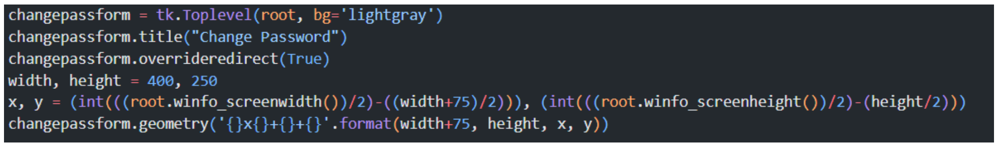
The process is very smooth and what I'd expect from a commercial application however I would like to see the login page open back up once the program has wiped users.

MILESTONE 11 – CHANGE USER PASSWORD

Objective of milestone: Create a method for user to be able to change password

Explanation of code

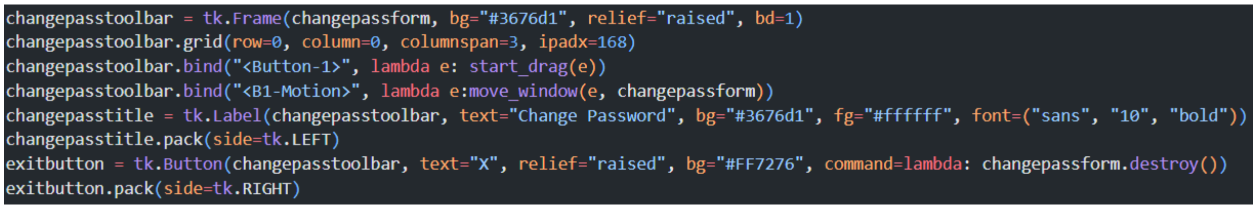
Centering the form



```
changepassform = tk.Toplevel(root, bg='lightgray')
changepassform.title("Change Password")
changepassform.overrideredirect(True)
width, height = 400, 250
x, y = (int(((root.winfo_screenwidth())/2)-((width+75)/2))), (int(((root.winfo_screenheight())/2)-(height/2)))
changepassform.geometry('{}x{}+{}+{}'.format(width+75, height, x, y))
```

Using reusable program components, I use the same centering template as for my other menu GUIs changing the width and height values. This window is also created as a top-level sister window of the root.

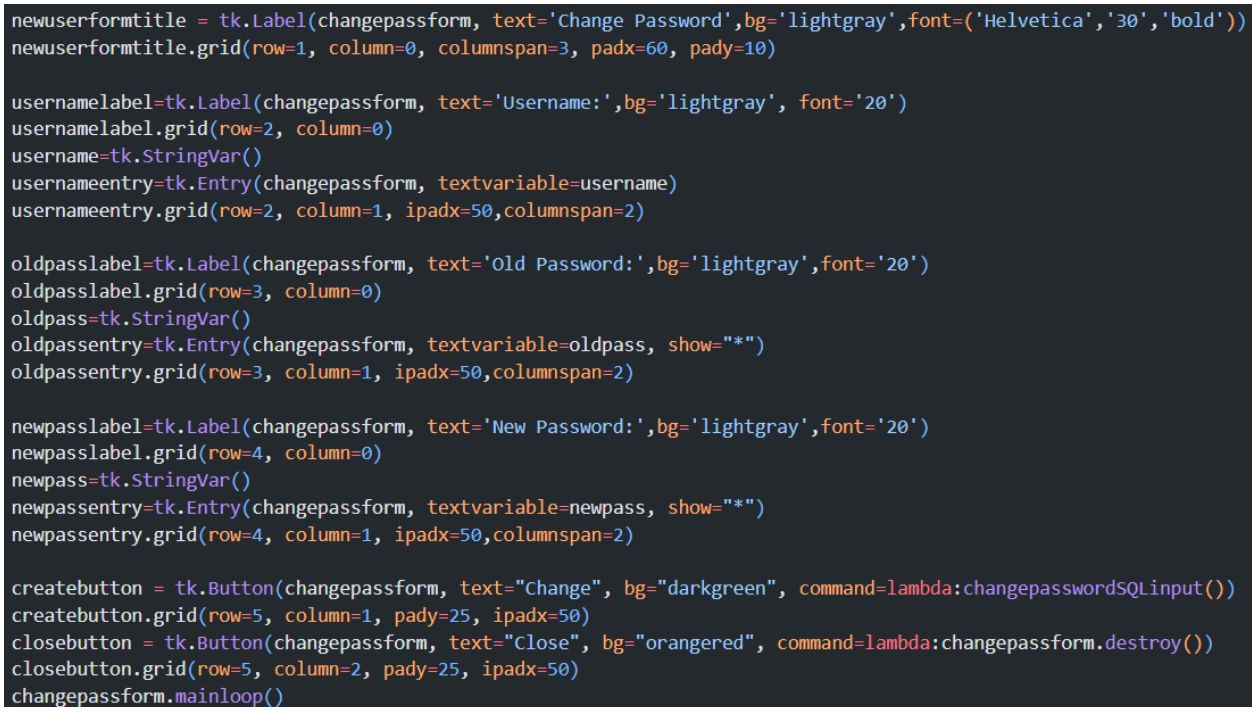
Creating the toolbar



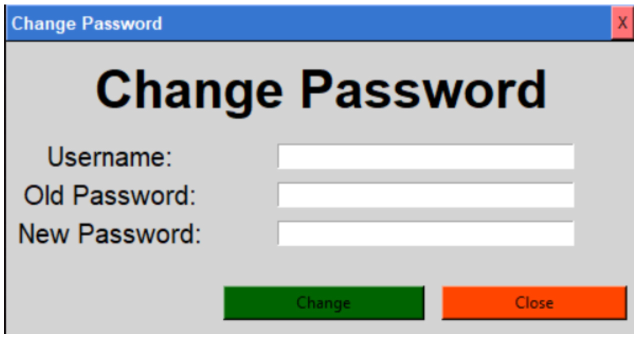
```
changepasstoolbar = tk.Frame(changepassform, bg="#3676d1", relief="raised", bd=1)
changepasstoolbar.grid(row=0, column=0, columnspan=3, ipadx=168)
changepasstoolbar.bind("<Button-1>", lambda e: start_drag(e))
changepasstoolbar.bind("<B1-Motion>", lambda e:move_window(e, changepassform))
changepasstitle = tk.Label(changepasstoolbar, text="Change Password", bg="#3676d1", fg="#ffffff", font=("sans", "10", "bold"))
changepasstitle.pack(side=tk.LEFT)
exitbutton = tk.Button(changepasstoolbar, text="X", relief="raised", bg="#FF7276", command=lambda: changepassform.destroy())
exitbutton.pack(side=tk.RIGHT)
```

Once again using the reusable toolbar code in this GUI, importing the start_drag and move_window modules that are in a separate file to allow for the movement of the window with the newly created toolbar.

GUI creation

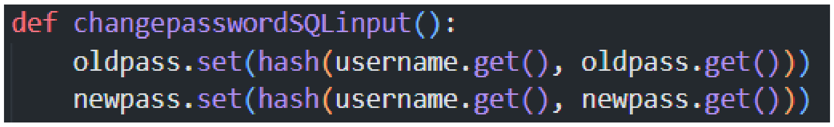


```
newuserformtitle = tk.Label(changepassform, text='Change Password',bg='lightgray',font=('Helvetica','30','bold'))
newuserformtitle.grid(row=1, column=0, columnspan=3, padx=60, pady=10)

usernamelabel=tk.Label(changepassform, text='Username:',bg='lightgray', font='20')
usernamelabel.grid(row=2, column=0)
username=tk.StringVar()
usernameentry=tk.Entry(changepassform, textvariable=username)
usernameentry.grid(row=2, column=1, ipadx=50,columnspan=2)

oldpasslabel=tk.Label(changepassform, text='Old Password:',bg='lightgray',font='20')
oldpasslabel.grid(row=3, column=0)
oldpass=tk.StringVar()
oldpassentry=tk.Entry(changepassform, textvariable=oldpass, show="*")
oldpassentry.grid(row=3, column=1, ipadx=50,columnspan=2)

newpasslabel=tk.Label(changepassform, text='New Password:',bg='lightgray',font='20')
newpasslabel.grid(row=4, column=0)
newpass=tk.StringVar()
newpassentry=tk.Entry(changepassform, textvariable=newpass, show="*")
newpassentry.grid(row=4, column=1, ipadx=50,columnspan=2)

createbutton = tk.Button(changepassform, text="Change", bg="darkgreen", command=lambda:changepasswordSQLinput())
createbutton.grid(row=5, column=1, pady=25, ipadx=50)
closebutton = tk.Button(changepassform, text="Close", bg="orangered", command=lambda:changepassform.destroy())
closebutton.grid(row=5, column=2, pady=25, ipadx=50)
changepassform.mainloop()
```



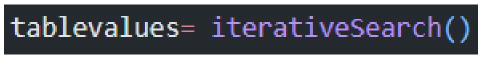
I follow the design sections layout to create a GUI for the change password form. It resembles the new user form to maintain a theme throughout the program. To save time I have added the change password module as a command assigned to the create button press event. The change pass form contains an old pass variable that can be compared to the stored SQL value. This will allow me to search for the user record and specifically edit the password with the new password entered.

Creating the change password module



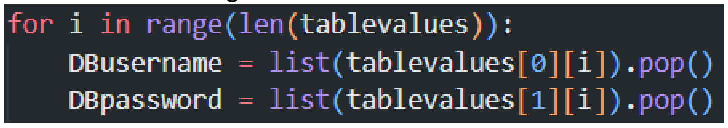
```
def changepasswordSQLinput():
    oldpass.set(hash(username.get(), oldpass.get()))
    newpass.set(hash(username.get(), newpass.get()))
```

First, I create the module and set the values of each password variable to the hashed version of itself, as previously mentioned the Tkinter variables are encapsulated because they are an object therefore I must use the set and get method to access the values.



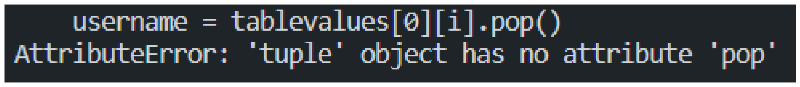
```
tablevalues= iterativeSearch()
```

I then use the module created earlier in the project which allows me to search through all the records in a database and assign them to locations in a tuple. The tuple will allow me to identify the record value which has to be edited.

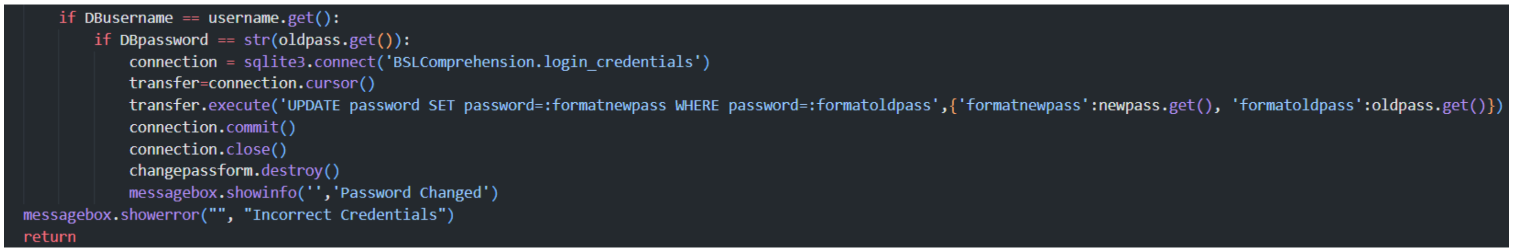


```
for i in range(len(tablevalues)):
    DBusername = list(tablevalues[0][i]).pop()
    DBpassword = list(tablevalues[1][i]).pop()
```

I proceed to iterate through the tuple, using the python length function to get the required iteration length. I separate slot 0 and 1 into their respective variables to make the search easier. Since tuple top values cannot be removed, I cast the tuple into a list and pop the current iteration value.



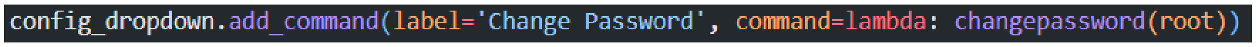
```
    username = tablevalues[0][i].pop()
AttributeError: 'tuple' object has no attribute 'pop'
```



```
    if DBusername == username.get():
        if DBpassword == str(oldpass.get()):
            connection = sqlite3.connect('BSLComprehension.login_credentials')
            transfer=connection.cursor()
            transfer.execute('UPDATE password SET password=:formatnewpass WHERE password=:formatoldpass',{'formatnewpass':newpass.get(), 'formatoldpass':oldpass.get()})
            connection.commit()
            connection.close()
            changepassform.destroy()
            messagebox.showinfo('','Password Changed')
messagebox.showerror("", "Incorrect Credentials")
return
```

While iterating through I check to see if the Database Username which has just been extracted from the tuple matches the Tkinter variable value that the user input, if the username does not match iteration will continue until it does or if it does not a message box error stating incorrect credentials will appear to the user. If this username matches, I must compare that the old password matches as well. This part of the code caused me many issues, comparisons would pop up errors. Looking back at the hashing algorithm, the value returned for the old pass was a byte data type therefore comparison with these values was not possible since data type values were different. To solve this issue, I cast the byte data type into a string value which can be compared to the database password value imported.

Again, I connect to the SQL database using the Sqlite3 python module. I proceed to create a cursor and execute the SQL which will allow me to input the user data into the database. The database cannot retain the old values which were used so I update the password table. I set the password as the new password where the password is currently the old password variable, because these values are stores in variables, I must format these values into the string. This means that I can integrate variables values into a string without having to hardcode a value, which obviously wouldn't be possible since there will be different values from different users.

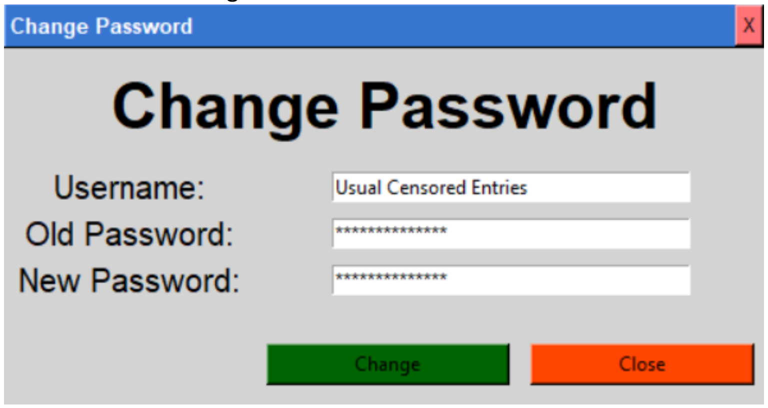


```
config_dropdown.add_command(label='Change Password', command=lambda: changepassword(root))
```

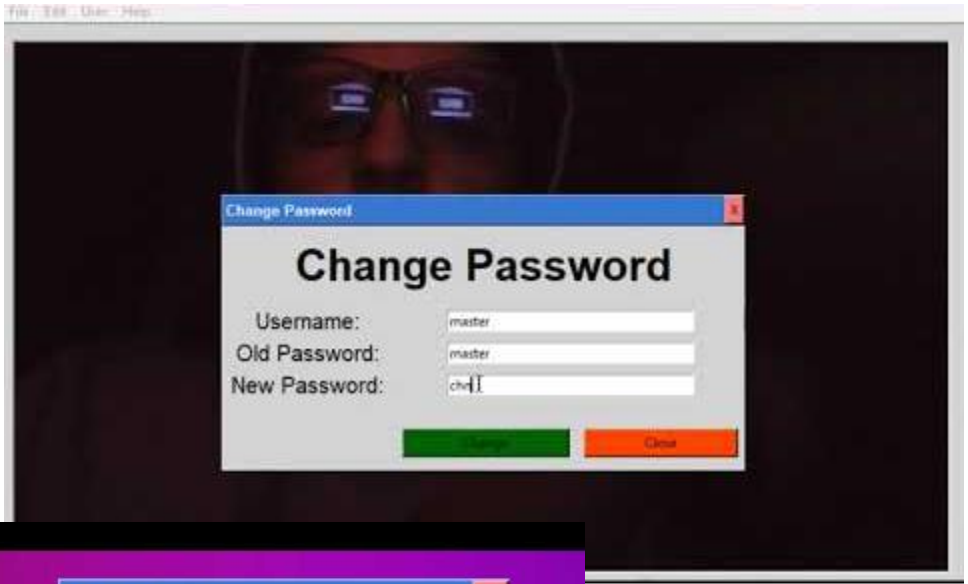
Finally, I add the new change password module into my config dropdown in the menu file so that the module can be accessed by the user once the change password menu button press event occurs.

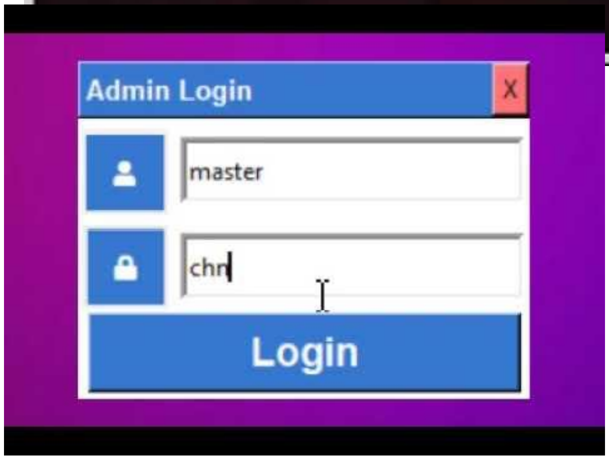
Module Testing

For the tests password censoring has been turned off so that there is visible proof that the password credentials are in fact altering after the program is a success. To the user the window will look something like this:



Both password entries are not visible and therefore a level of security remains for the user, I'm happy with the layout of this window as it has followed the GUI plan and resembles other elements of the project such as the login form and the new user window.





The window is still integrated into the main navigation menu, the load could be improved with some minor glitches which go away once the user interacts with the window. For this test I input the default mater values for this program stated in the previous milestone section. I change the default password and will proceed to test this on the login form. Once the change button is pressed there is some processing time. The message box pop-up confirming to the user that the password has been changes is working perfectly.

Using the old default master values causes an error message box, entering the new password and pressing login closes the window meaning that the window authorisation has been successful and that the program is working as intended. The Censoring of password entries has also been turned off on the login form for the testing of the module.

This milestone has worked better than expected and aside from small bugs occurring when two windows are layered, it behaves exactly as the milestone description desired. The complications were minimal.

User Feedback

I like the look and feel of the window and the way in which it is laid out. The design was intuitive and it was clear what the window was for.

MILESTONE 12 – EXPORT FILE

Objective of milestone: Allow users to export the contents of the character display into a variety of files.

Creating the new file

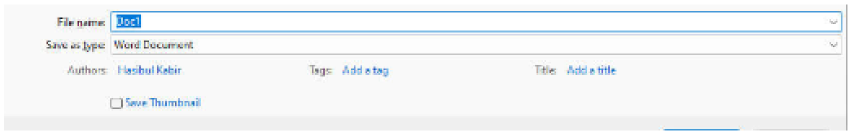
filemenucommands.py

```python
def export(root, text):
```

```python
file_dropdown.add_command(label='Export',command=lambda: export(root, translationresult))
```

I begin by creating the file dropdown commands file which will include export and save commands. I create the export function and pass the root window as a parameter, so that the top level window creation is possible, and also the text that will be visible in the translation result bar. I proceed to add this command to the file command menu as export. Again, using the lambda function in python so that the command only activates once the button press event happens.

Root and Toolbar



File name: Doc1
Save as type: Word Document
Authors: Hasibul Kabir Tags: Add a tag Title: Add a title
Save Thumbnail

For this window I tried to follow the Save As look Word but with my own theme.

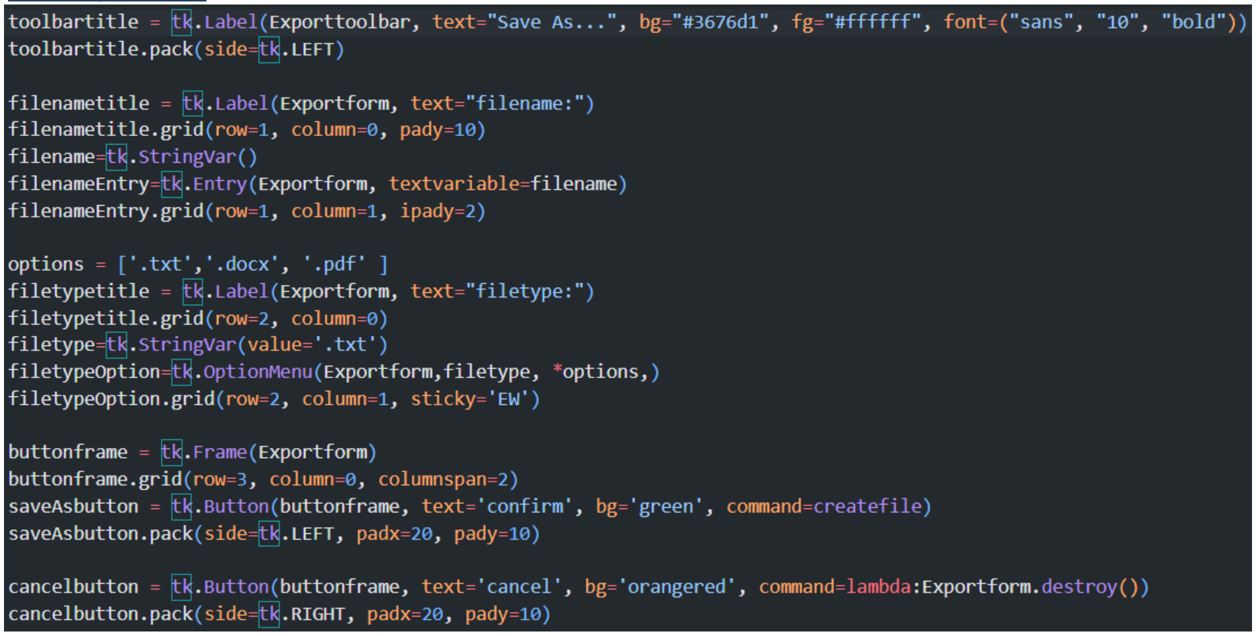
```python
Exportform = tk.Toplevel(root)
Exportform.overrideredirect(True)
x, y = (int(((Exportform.winfo_screenwidth())/2)-((180)/2))), (int(((Exportform.winfo_screenheight())/2)-(100/2)))
Exportform.geometry('{}x{}+{}+{}'.format(190, 135, x, y))
```

I create the top-level window and center the window by reusing the x, y screen centering code which considers the screen size and the top-level size. Since the window design I am trying to replicate is smaller I change the x, y values of calculation and width, height values to so that they fit with the design.

```python
Exporttoolbar = tk.Frame(Exportform, bg="#3676d1", relief="raised", bd=1)
Exporttoolbar.grid(row=0, column=0, columnspan=2, ipadx=60)
Exporttoolbar.bind("<Button-1>", lambda e:start_drag(e))
Exporttoolbar.bind("<B1-Motion>", lambda e:move_window(e, Exportform))
```

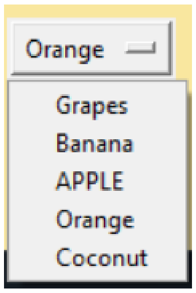
Creating using reusable program components, the difference here happens on the ipadx value, ipadx stretches the widgets in the x plane by the specified factor. In this case since the form window is so much smaller the stretch variable must be significantly decreased. These values were approaches heuristically.

Creating the GUI



```
toolbartitle = tk.Label(Exporttoolbar, text="Save As...", bg="#3676d1", fg="#ffffff", font=("sans", "10", "bold"))
toolbartitle.pack(side=tk.LEFT)

filenametitle = tk.Label(Exportform, text="filename:")
filenametitle.grid(row=1, column=0, pady=10)
filename=tk.StringVar()
filenameEntry=tk.Entry(Exportform, textvariable=filename)
filenameEntry.grid(row=1, column=1, ipady=2)

options = ['.txt','.docx', '.pdf' ]
filetypetitle = tk.Label(Exportform, text="filetype:")
filetypetitle.grid(row=2, column=0)
filetype=tk.StringVar(value='.txt')
filetypeOption=tk.OptionMenu(Exportform,filetype, *options,)
filetypeOption.grid(row=2, column=1, sticky='EW')

buttonframe = tk.Frame(Exportform)
buttonframe.grid(row=3, column=0, columnspan=2)
saveAsbutton = tk.Button(buttonframe, text='confirm', bg='green', command=createfile)
saveAsbutton.pack(side=tk.LEFT, padx=20, pady=10)

cancelbutton = tk.Button(buttonframe, text='cancel', bg='orangered', command=lambda:Exportform.destroy())
cancelbutton.pack(side=tk.RIGHT, padx=20, pady=10)
```

The Gui for the export window follows uses all the same Tkinter widgets that have been previously seen and explored, however there is one widget in this milestone that has not been explored yet



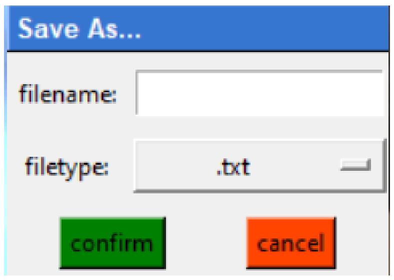
The Tkinter Option Menu.

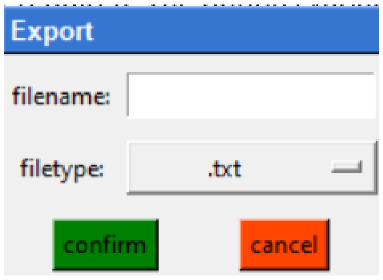
`filetypeOption=tk.OptionMenu(Exportform,filetype, *options,)`

It is defined as any other widget with the tk.Method and the master window that it is assigned. The other initialization variables, filetype and options, are necessary for the option menu to function. The filetype is simply a Tkinter variable like I've used before with a default file value of .txt. `filetype=tk.StringVar(value='.txt')`

Since the variable is a requirement in the Option Menu though, the text command keyword that I'm used to is not required. The options variable is a list of file ending tags, which will be used as the options for the user. `options = ['.txt','.docx', '.pdf' ]`

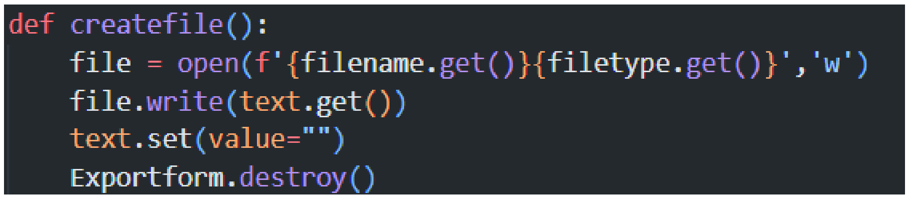
The asterisk is a method of unpacking a list. Since it is being used in its raw format, using the asterisk removes any brackets and commas present and shows the raw data.





Originally the function was going to be used for the Save As command however when in development I realized that the module performed more like an export function. The code was slightly tweaked.

Create file function

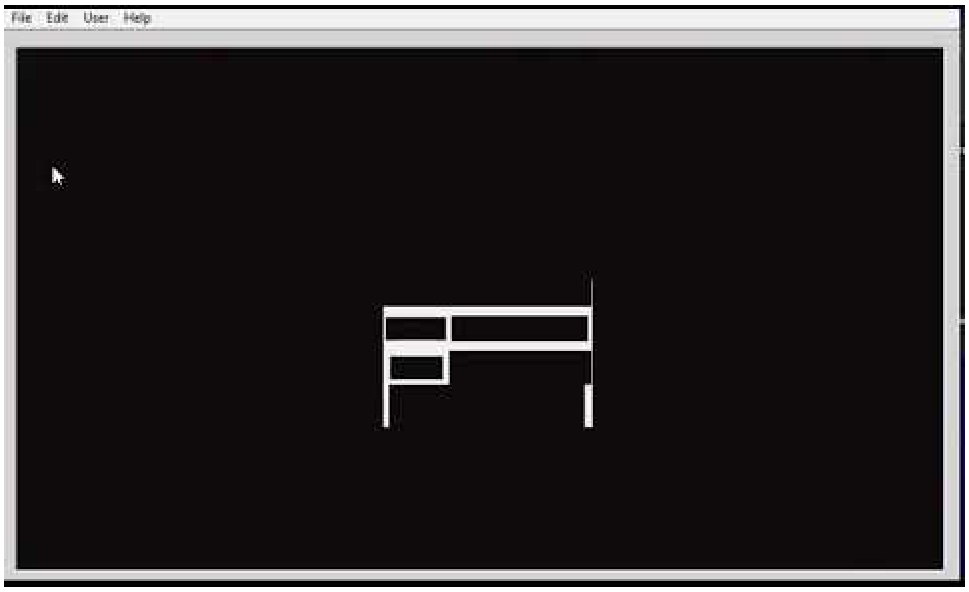


```
def createfile():
    file = open(f'{filename.get()}{filetype.get()}','w')
    file.write(text.get())
    text.set(value="")
    Exportform.destroy()
```

The function will use pythons file handling module which include the open() method to create new files. I begin by defining the file variable as the file I am about to create. I use the open method and format the text, again formatting allows me to put user changing variables into strings and use the get method to put the chosen filename and extension as the file name. The w defines what I wish to do with the file, in this case we simply want to write to the file, so I open it in write mode. The open method also means that two files with the same name will not be created.

I proceed to write the contents of the Translation Result into the new file that has just been created, this text is encapsulated by Tkinter so I must once again use the get() method. After the text has been written to the file, I reset the value of the Tkinter variable which empties the contents visible to the user. Now that the file has been created and the contents have been exported to the chosen file type I can destroy the window and return to main.

Module Testing



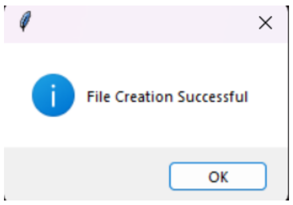
The window is working well alongside main however similar to the change password form, there is a bug where the camera feed colors pass through to the entries. This shouldn't be a huge issue since they disappear after a short period of time.

To test the module I have the folder for exports open on the right, currently there is only the git file, some caches from program running and a BSL reference picture.

I create my firs file as PDF with filename begin with A so that it would jump to the top. Once the create button has been pressed and the folder for exports refreshed, a new pdf file can be seen which has been created by this module. I repeat the process for the docx file and once again a file appears, one as a pdf and one as a word document with the same file name.

To test whether the files won't be replicated, I open the from once more and create a file with the same name as the recently created pdf file and press enter. By viewing the folder on the left, you can see that a new duplicated file has not been created meaning that the open function is working correctly. One issue that I have forgotten about is the confirmation of file creation, this is beneficial for the user to make sure that the module has in fact completed.

```
messagebox.showinfo("", "File Creation Successful")
```


File Creation Successful

This was a simple solution and has made the module better for it.

The export function is working perfectly, and no issues are present. However, I would like to improve some of the capabilities by adding file location choice and a greater span of file choices for the users.

User Feedback

I think that I should be able to choose the file location for my export, but I like the word-like layout followed for the window

MILESTONE 13 – PAUSE FEED

Objective of milestone: Allow users to pause and continue the video feed displayed in the main program.

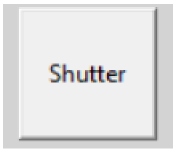
Shutter Cover

```
booleanshutter = tk.BooleanVar(value=True)
shutterbutton = tk.Button(root, text="Shutter", command=lambda:shutter())
shutterbutton.grid(row=1, column=1, ipady=20, ipadx=10)
```

I begin by creating a Boolean Tkinter variable which will have a default value of True. Since the Shutter is binary state of Pause or Continue, I can assign a Boolean variable for the handling. True for camera feed off/ Pause is pressed and False for camera feed on/ Continue is pressed. This allows me to reduce the number of button widgets in main to just one. I then create the Shutter Button which is already connected to the lambda shutter command

```
def shutter():
```

This button is gridded at the bottom right corner of the program on the same row as the translation result bar. It is a basic Tkinter button template.



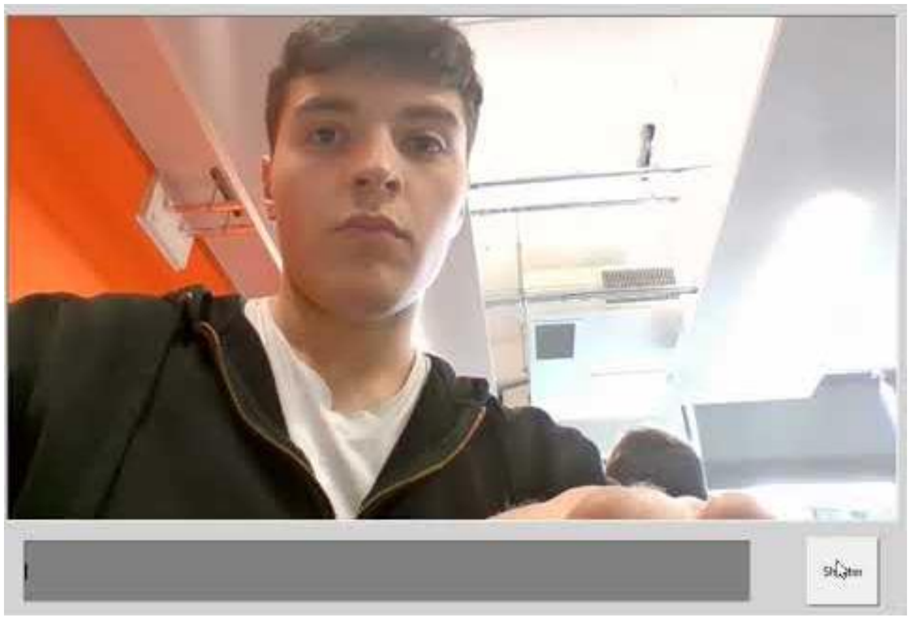
Shutter Function

```
if booleanshutter.get() == True:
    cameracontainer.grid_forget()
    cameracover.grid(row=0, column=0, padx=10, pady=15, columnspan=2)
    booleanshutter.set(value=False)
else:
    cameracover.grid_forget()
    cameracontainer.grid(row=0, column=0, padx=10, pady=15, columnspan=2)
    booleanshutter.set(value=True)
```

I check the Boolean values stored within the Boolean Shutter variable. If True, I unpack the camera container removing the lived feed view for the user and replace the location with the camera cover widget. This camera cover widget will state whether the camera is paused, so that the user id notified of whether there is an issue or if it is being used for its intended purpose. When this loop is completed I set the values to False so that once it is pressed again the continue loop will be run instead.

The false loop simply does the opposite of the true loop. The camera cover widget is unpacked and replaced with the camera container live feed that will now be visible to the user again. Again, at the end of this loop I replace the Boolean value so that the other will be run once the user presses the button again.

Module Testing



This video shows the Shutter Button both working correctly for the pause function and for the continue.

Dimensions of the Shutter Cover align perfectly with the camera container so replacement bewteen button presses remains smooth, the Paused text in the center is obvious to the user accomplishing the goal of notifying the user when pausing has occurred.

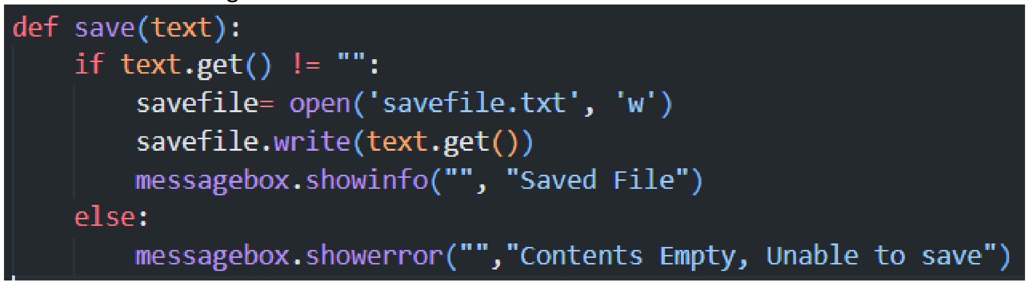
The milestoen set out to create a useful Shutter function for the program which was obvious to use by the end user. This module has accomplished this and in a rather effective way, since code is not overly complex.

User Feedback

I think that the camera cover is very well done and visible on the screen. After playing with it for a bit though I think that the camera feed should actually cut off rather than covered since the program activity continues even when the Paused is displayed.

MILESTONE 14 – SAVE FILE

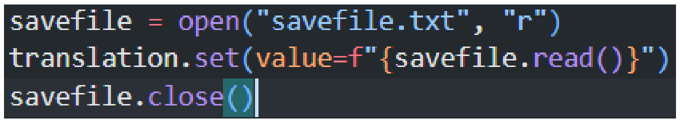
Objective of milestone: Allow users to save contents of translation result bar and load it up in next instance of program.



```
def save(text):
    if text.get() != "":
        savefile= open('savefile.txt', 'w')
        savefile.write(text.get())
        messagebox.showinfo("", "Saved File")
    else:
        messagebox.showerror("","Contents Empty, Unable to save")
```

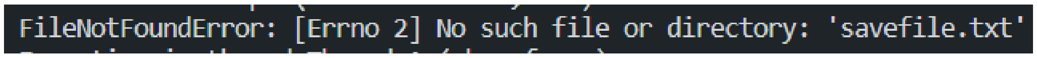
I begin by creating the save function which will take in the translation result as a parameter since this is the text that I wish to save and reload. I check to see whether the translation box is empty, if it is then an error will pop up stating that you are unable to save an empty file. If the result bar does contain text, then a text file named saved file is opened in write mode and all the contents of the widgets are entered into the file. This is very similar to the export code used in the previous milestone since this is the best method of handling files in python. Once the transfer has been complete then the user receives a message box stating that the task has been a success. Any changes made can be saved again since the files will simply be overwritten with the new text.

This takes care of the saving within the program however maintain the save state and loading it when the program starts up is slightly more complicated



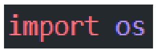
```
savefile = open("savefile.txt", "r")
translation.set(value=f"{savefile.read()}")
savefile.close()
```

I proceed to go into the main file where the save state load will be handled and open the save file in read mode instead of writing this time. I proceed to set the value of the translation variable to the save file contents, using formatting to place a variable into the string. I then close the file.



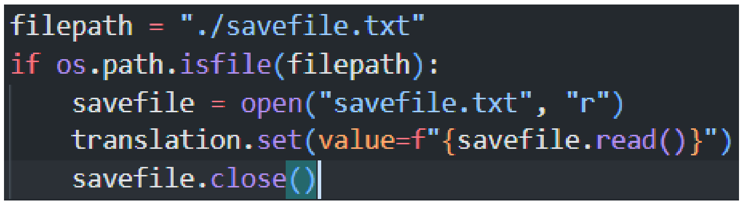
```
FileNotFoundError: [Errno 2] No such file or directory: 'savefile.txt'
```

The code will not work if a save file does not currently exist, which tells me that I will need to point towards the directory of the file and check whether it is there.



```
import os
```

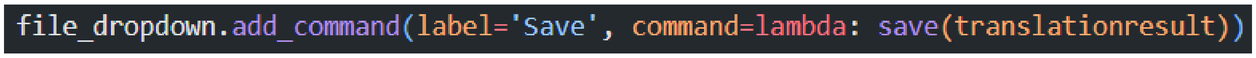
The best library for handling this is the OS itself, the OS is already adapted for handling these files and the code is the same and command prompt text.



```
filepath = "./savefile.txt"
if os.path.isfile(filepath):
    savefile = open("savefile.txt", "r")
    translation.set(value=f"{savefile.read()}")
    savefile.close()
```

I first set the file path, the dot stands for the current directory where it will always be stored and the filename which will always be the same as well. Using the OS library, I check to see if the file currently exists by using the is file method and passing the file location as a parameter. Localization of a file returns True and allows the if statement to proceed which solves the problem that was happening above.

Finally, I need to add the save module as a command under the menu file drop down.

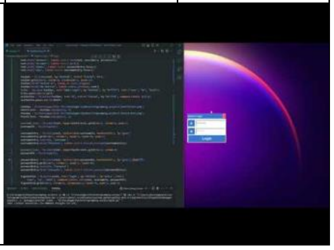


```
file_dropdown.add_command(label='Save', command=lambda: save(translationresult))
```

D. EVALUATION

POST DEVELOPMENT TESTING

MILESTONE 1/2: LOGIN FORM

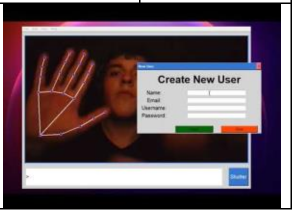
Test No	Testing and Inputs	Expected Output	Timestamp	Pass-Fail
1	Verify if use can login with the correct credentials, including usernames and password that are supposed to match. Inputs: (master, master) (user1, user1password)	The credentials should be accepted. Mismatched usernames and password should not authorise login. Login Form should close down and main should begin to load	0:00 – 0:40	Pass
2	Check what happens when fields are left blank with no inputs. Inputs: Not Applicable	Leaving either of the fields blank should return an error message box to the user stating that the field cannot be left blank.	0:45 – 1:00	Pass
3	Confirm error message provided when user enters incorrect credentials. Inputs: (errortest, errortest)	Entering the incorrect credentials should return a message box to the users stating that one or more of their inputs are incorrect.	1:00 – 1:10	Pass
4	Verify shortcut key binds. This includes the functionality of the arrow keys, enter key and escape button. Inputs: Key Presses from Up arrow, Down arrow, Enter and Escape keys	Pressing the up and down arrow keys should make the focus switch between entry widgets. The enter key should begin the authorisation function once clicks and the escape key should halt the program.	1:20 – 1:39	Pass
	Test Video for Milestones 1/2:			

MILESTONE 3 – LOADING SCREEN

Test No	Testing and Inputs	Expected Output	Timestamp	Pass-Fail
1	Check to see if loading screen appears after login form authorisation. Inputs: Not applicable	The Login Form should close down, and the login screen should appear almost immediately following the user authorisation displaying a loading bar and time remaining	0:00 – 0:13	Fail
2	Does loading screen fill space between the closure of the login form and the initialisation of main. Inputs: Not applicable	The loading screen should remain visible to the user from the instant that the login form closes down to the moment that the main file opens	N/A	Fail
3	Displays the remaining loading time until main opens. Inputs: Not Applicable	Users should be able to see an estimate for time remaining on the loading screen bar	N/A	Fail
Test Video for Milestone 3:				

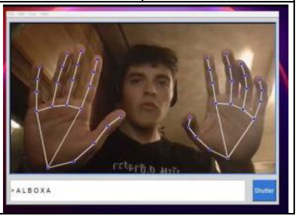
MILESTONE 4/6 – CAMERA VIEW AND PIPELINE

Test No	Testing and Inputs	Expected Output	Timestamp	Pass-Fail
1	Does The camera view loads instantly after main window starts up using a separate thread. Inputs: Camera Live Feed	Camera feed should load with a small delay, once it has loaded live feed will take up the top half of the program	0:10 - 0:15	Pass
2	Does the live feed produce minimal motion blur when a user moves suddenly. Inputs: Camera Live Feed	Live feed should only have a small amount of motion blur and the camera frame should not begin to stutter as movement speed increases	0:20 - 0:25	Pass
3	Testing whether the Video feed continues while other tasks are being executed within the window. This provides testing of whether the threading is working correctly. Inputs: Create a new user with (user2, user2password), Camera Live Feed	The live feed should continue being outputted while the separate thread is on, when creating a new user, I should be able to type in the new credentials and see the camera feed at the same time with no stutter.	0:40 – 1:10	Pass

4	Does the media pipe pipeline only begin to display to the user when both hands appear on the camera feed. Inputs: Camera Live Feed, Hand Frames	The pipeline will not display when one hand is in the feed but once both hands are in the feed, visible nodes and edges will appear on the users' hands remaining in a fixed position on the hands even when movement occurs	0:15-0:20	Fail
5	Does the pipeline remain on the live feed even when user hands are overlapped and moves suddenly. Does exposure to light effect the display. Inputs: Camera Live Feed, Hand Frames	I expect that fast movement will not cause many problems with hand detection however very fast and sudden movement will stop the tracking for a small amount of time until it refocuses. Overlapping hands will cause some problems but should be fixed by slow movement and light will heavily effect the display capabilities.	1:20 - 1:40	Fail
	Test Video for Milestone 4/6:			

MILESTONE 7/8 – CHARACTER COMPREHENSION/DISPLAY

Test No	Testing and Inputs	Expected Output	Timestamp	Pass-Fail
1	Does the model correctly translate the signed character of the user	The model should be able to comprehend a limit quantity of characters in BSL with a varying accuracy depending on the character	0:00 – 0:05	Pass
2	Does model respond correctly from varying distances from the camera feed	The model should not have any trouble detecting from close distances or from medium distance however will struggle with detection from long ranges	0:20 – 0:40	Pass
3	Do illuminated spaces slow/Halt the comprehension program	The program should be able to detect even when higher levels of light exposure are present, this doesn't include exposure levels which change the visible screen such as directly towards a light source.	0:45 – 0:55	Fail

4	Can the model respond to fast hand movements by the user	The model should be able to handle movement which fall within the normal range of speed such as moving across the screen however it will not be able to handle detection once the live feed begins to create motion blur on the hands.	0:58 – 1:08	Fail
5	Are the characters displayed instantaneously after model detection	Characters should be shown in the translation bar as soon as the model detects the characters from the hand model.	0:00 - 0:05	Pass
6	Can the user manually write within the translation bar, which they should not be able to do. Input: Random characters	The user should not be able to manually type into the translation bar and must only have access to it when signing characters. Any attempt at typing within the bar will wipe the types characters.	1:10 – 1:20	Pass
7	Does the character detection model display the detected character on the live feed and on the translation bar	The program should display the current characters being signed on the live feed on the top left corner of the screen and the character should be added to the translation bar as well.	0:00 – 0:05	Pass
Test Video for Milestones 7/8:				

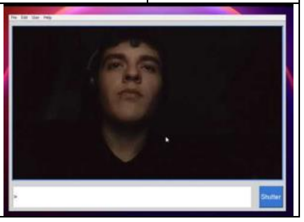
MILESTONE 9 – ADDING CREDENTIALS

Test No	Testing and Inputs	Expected Output	Timestamp	Pass-Fail
1	Should be able to open program from the navigation menu and load immediately. Inputs: Event clicks on navigation menu.	Program window should appear almost instantly after the command from the New User menu dropdown executes. It will appear at the topmost level.	0:00 – 0:03	Pass
2	Does the form move across the window when the toolbar is dragged with minimal error. Input: Click event on toolbar	Form will move across the window however might slightly drag behind if computer is already running intensive tasks.	0:05 – 0:10	Fail

		Sharp movements should not affect its reliability.		
3	Is an error message provided when one of the entry field holds the wrong format of data. Input: Name entry (John, JohnSmith, John Smith Junior, John Smith)	An error message should be provided if only first name is entered, however no method of prevention has been added for a name with no spaces. It will likely count as a singular name. Any name that contains a first and last name part will be accepted.	0:20 – 1:15	Pass
4	Does the program provide an error if all the input field are left empty. Inputs: N/A	The program should provide an error message that doesn't allow a user to enter information while field are empty. Username and password fields may also not be empty since this is a security issue.	0:45 – 1:15	Fail
5	Does the form provide a method of confirmation if the user credential input was successful. Input: N/A	A message box stating to the user that credentials storage has been successful should appear once the enter button event executes.	1:15 – 1:16	Pass
6	Does the exit method work correctly, exiting the new user form and returning to main. Input: Exit button press event	The form should close down once the exit button is pressed and should show the main window.	1:35 – 2:25	Pass
Test Video for Milestone 9:				

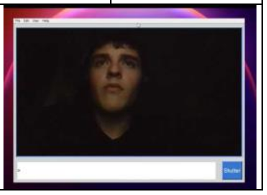
MILESTONE 10 – DELETE ALL CREDENTIALS

Test No	Testing and Inputs	Expected Output	Timestamp	Pass-Fail
1	Does the event provide you with a confirmation message box in case a mis click occurred. Input: N/A	Once the delete all user's dropdown command has been pressed a message box should appear on the top level confirming whether you want to go through with the current action.	0:05 – 0:12	Pass
2	Is the exit/no button working correctly for cancelation of the function.	The no button press on the confirmation message box will	0:12 – 0:15	Pass

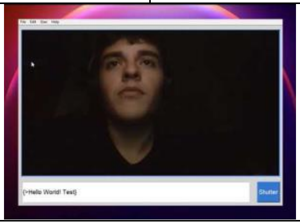
	Input: N/A	close down the box and return the user to the main program		
3	Is a confirmation message given to the user once the program yes button has been pressed. Input: N/A	Pressing the yes button will close the message prompt for the user and open another stating that all data has been erased. After this message close the program will close for information resubmission.	0:20 – 0:25	Fail
4	Does the module successfully wipe details once confirmation has been given to the user. Input: login form(master, master), (user1, user1password)	Once the data has been wiped, another login form will be opened and tested using previously added credentials. The user1 credentials will no longer authorise access since they are not stored in the database however the master default credentials will allow access to the main program.	0:30 – 0:49	Pass
	Video Test for Milestone 10:			

MILESTONE 11 – CHANGE USER PASSWORD

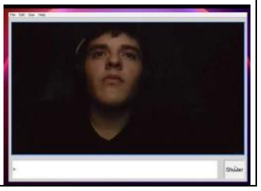
Test No	Testing and Inputs	Expected Output	Timestamp	Pass-Fail
1	Does the drag function with the toolbar work correctly and smoothly. Input: N/A	Form will move across the window however might slightly drag behind if computer is already running intensive tasks. Sharp movements should not affect its reliability.	0:03 – 0:09	Pass
2	Is an error message shown if the username or password fields are left blank. Does this error message show when all fields are left blank. Input: One input is correct, master for both	An error message should show if the username/password field is left blank. This same error is shown if all fields are left empty since it is dependent on username and password fields as well.	0:10 – 0:18	Fail
3	Is a confirmation message shown when the change button is pressed, and the credentials are changed. Input: N/A	A confirmation message box is shown once the change button is pressed stating to the user that their credential change has been successful	0:40 – 0:45	Pass

4	Have the previous credentials changed, including an inability to use the old password/username with the new password/username. Input: old credentials(master, master), new credentials(master, changedpassword)	Mismatched credentials should not work, this includes using old and new credentials together since it will produce an authorisation error. Using the new changed credentials will allow authorisation into the main program.	0:55 – 1:05	Pass
Video Test for Milestone 11:				

MILESTONE 12 – EXPORT FILE

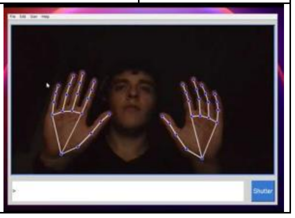
Test No	Testing and Inputs	Expected Output	Timestamp	Pass-Fail
1	Does an error message prompt appear if the filename entry is left blank. Input: N/A	A Tkinter message box will appear to the user if the filename entry is left blank with no input	0:10 – 0:18	Fail
2	Can you select all the different file extensions with seamless change between them. Input: Click Event	Selection of file extensions is smooth and will not cause issues for the user.	0:38 – 0:45	Pass
3	Does a confirmation message box prompt appear once the file create button has been pressed. Input: N/A	Once an input has been added to the filename entry and the confirm button has been pressed a message box prompt will appear confirming the creation of the export file into the user's system.	0:45 – 0:47	Pass
4	Does the module create a visible and readable file in the user's system which contains the contents of the translation entry. Input: A small sentence which will be in the translation entry (Hello World! Test)	Once the module has been executed a file with chosen filename and extension will appear in the same directory in which the main program is stored which contains the contents of the translation bar	0:50 – 0:57	Pass
Video Test for Milestone 12:				

MILESTONE 13 – PAUSE FEED

Test No	Testing and Inputs	Expected Output	Timestamp	Pass-Fail
1	Is feed pausing responsive and able to switch without deforming the window. Input: Click Event	The camera cover widget should appear instantly after pressing the shutter button. The dimension of the shutter will cover the location which the camera container covered perfectly providing a full blackout effect.	0:00 – 0:02	Pass
2	Does pressing the Shutter button cause a Pause and Continue of the live feed or does it remain paused. Input: Click Event	Pressing the Shutter button for the first time will cause a Pause cycle and pressing it again will cause a continue cycle. This repeats with every button press depending on the current cycle which it is in.	0:02 – 0:05	Pass
3	Can a window be turned on and off repeatedly without causing the program to break or even crash. Input: Click Event	Pressing the Shutter button repeatedly should not crash the program. The user should be able to see the shutter stages clearly.	0:05 – 0:25	Fail
Video Test Milestone 13: 				

MILESTONE 14 – SAVE FILE

Test No	Testing and Inputs	Expected Output	Timestamp	Pass-Fail
1	Does the save command have a prevention method for when the translation bar is empty of data. Input: N/A	A message box prompt will appear when the translation bar contents are empty stating to the use er that saving a file with no contents is not possible.	0:10 – 0:15	Pass
2	Is a message provided to the user when the save has been executed successfully. Input: N/A	A message prompt will be shown to the user once the save command is executed stating to the user that the contents have been saved successfully.	0:30 – 0:32	Pass
3	Do the contents of the translation bar transfer into the save file.	The contents should be visible when opening the save file in the main program directory.	0:40 – 0:45	Pass

	Input: translation bar (contents)			
4	Do the contents of the save file show on the translation bar when the program is reloaded. Input: N/A	The contents of the save file should appear in the translation bar when the program is reloaded so that the user can carry on from where they left off.	1:00 – 1:05	Fail
5	Do the contents of a save file remain when a new save file is produced. Input: translation bar (new contents)	The contents of the current save file should not overwrite the contents of the previous save file but rather it should be displayed on a separate line in the file.	1:20 – 1:47	Fail
Video Test for Milestone 14:				

EVALUATING THE PROGRAM

LIMITATIONS

Version limitations: Due to the amount of time that my project has taken me to complete I have not managed to produce as many cycles of development that I had wished. With more time I would improve certain bugs in the system that will better the overall user experience. This would include broadening the quantity of characters that the model can detect and improving the quality of user credential data handling.

Model Limitations: My restricted knowledge of deep learning models and experience with coding has limited the way in which I have handles the comprehension program. I have opted for a method which takes approximations rather than learning via Machine Learning. This has limited the accuracy of the project and means that if I proceed using this method, more errors will appear than if I were to use a Convolutional Neural Network.

GUI Quality: When I began my project, I opted for the Tkinter GUI library because I was already familiar with how the code structure worked, trying to limit the amount of new content that I would have to learn and reducing the tight time constraints that were already guaranteed. If I were to complete the project again, I would opt for Pyqt5, this library includes a greater range of widgets and integration with further libraries which could have modernised my program and improved the performance. This modernised GUI is more efficient and could probably run on computers with lower specifications than my current machine.

Language restraints: My program has been written in Python. This is a language that I'm familiar with and therefore I opted for when creating the project however if I was trying to improve the program and didn't have any time constraint, I would use a more efficient medium level language such as C++. Using a language like C++ would allow me to produce a program that could run on any kind of machine. Pythons' high-level nature limits how efficient I can make the final program since there are many aspects to the final product.

FINAL EVALUATION

Success Criteria		Success	Evaluation
1	Create the login form GUI	Fully Met	The forms interface shows all the necessary widgets for the next part of the development. The login GUI is fast, loading up immediately after the executable is run. Key bindings allocated to this GUI also make the form able to be navigated without the use of a mouse, simply your keyboard. My goal was to create an appealing GUI which attracted the user at first sight since it would be the first part of the program that they would interact with, I would say that this has been fully met and successful beyond what I thought was possible with the Tkinter GUI library.
2	Authorising user login	Fully Met	The user authorisation algorithm has been an enormous success in my project. The algorithm works for authorisation of multiple users, not allowing incorrect credentials and informing the user instantaneously when an error like this happens. Not only does this algorithm perform this task well, but it also performs this task with a level of security for user information. All password that are handled with a connection to the SQL database run through a hashing algorithm beforehand that makes the password incredibly difficult to decrypt. A small addition of starred passwords on the entry widget also adds a level of security to keep the user's data safe. The algorithm has been through thorough testing and has proved that the only way to access the system is via the correct, matched credentials of the user.
3	Loading Screen Implementation	Not Met	The loading screen implementation has been more difficult to implement than I originally thought when beginning the program. The original purpose of the loading screen was to create a buffer between the two files opening however I quickly found that the library that I chose to use was not adapted to handling time

			tasks like this. Although I did manage to create a load screen which implemented well with Tkinter and looked like a professional loading bar you would see on a commercial program, it was not aiding the loading times between the programs. In fact, the loading screen acted as a third separate module which instead of filling the buffering time actually slotted to create a larger wait time for the user. This was counterintuitive and made the program less suited for the user, so I removed the solution. In the next iteration of my program, I would like to find a loading screen which fits in with the buffer time so that the user doesn't have to be looking at a blank screen wondering whether the authorisation process of the login form has worked correctly or whether the program has ceased from working.
4	Displaying Camera View	Fully Met	The implementation of a live feed into a Tkinter GUI was something completely new, it required a lot of research and understanding of how both Tkinter and OpenCv function so that I could integrate the two and understand how to debug an issues that arose. The original criterion was to create a live feed within the program for later use, and I would say that this solution has been successful in achieving that. Integrating originally was slightly difficult since the OpenCv image type and Tkinter image type are completely different however once that problem was solved all that was needed was small tweaks in refreshing and image size to achieve a frame which lacked stutter and motion blur. Movements that happen within the camera happen identically in the camera container and users cannot tell the difference between the actual camera feed and the updating of frames within the program. Although this criterion was met, I would like to improve the efficiency of this particular module so that it is possible for me to display a larger camera live feed. As of right now, any larger feeds cause the images to stutter due to limitations in computational power.
5	Navigation Menu	Fully Met	The Tkinter navigation menu that I've incorporated is well suited to the requirements of the program. The original criteria was to simply create a method of navigation that

			allows access to features that will come later in development such as credential handling and file handling. The simplicity of the menu, which follows the default OS theme, means that it is easy to navigate for the use for the user, since it resembles other navigation menus in popular mainstream products, but it also means that the navigation menu can sit gridded at the top of the program with little graphical and process requirements allowing the main bulk of the program, the BSL comprehension, to take a larger share of the processing. Although it is simple, it falls perfectly in line with I criteria and is therefore a full success.
6	Displaying Pipeline	Fully Met	Displaying media pipes pipeline was identified in the criteria as a method to detect the hands and ultimately inform the user of where the machine is locating their hands on the live feed. I've handled media pipe hand tracking in previous projects before however since this project required the independent tracking of two hands for BSL recognition it was a new challenge to face. There are methods of support for integrating media pipe pipeline into OpenCV Tkinter however multihandedness (left- and right-hand tracking) is not often done. This lead to me having to research different methods in which I could possibly do this and found the solution to be to simply create to models, one to for the left hand and one for the right. Any other method which used only one model would detect the nodes on either hand as the same and this wouldn't work since I needed independent hands. Id say that this module has been successful and fully met the original criteria it was intended for, although users experience small mishaps in pipeline display when inputting sudden movements, the overall tracking for users is accurate and more than enough to work within the projects scope.
7	Character Comprehension	Partially Met	Creating the character comprehension model was by far the most intensive and time-consuming process of the entire project. It took hours of research into CNN models and heuristic approaches to understand the processes which I had to go through to create an interpretation program. The criteria stated that the comprehension program would ideally

			be built using a Machine Learning model but as I knew that this was certainly going to be a demanding task, I set the goal of approaching it heuristically as well. The model creation was successful after hours of looking into code listings and theory however training the model took much longer than expected and the degree of accuracy of which is came out with was not sufficient. Using just two-character training images, I managed to train the model to 60% after around 7 hours of processing time. This was not realistic with my time constraints, so I had to opt for the heuristic approach. Although the node location comparison approach will start to degrade in accuracy once expanded, it will work for the current process that I need it for. It has been able to detect the characters which have been added accurately and with minimal error. The weightings approach, similar to what you would see in a path finding algorithm, has worked well for my project and leads me to believe that until the next iteration of the project this solution provides a partial success.
8	Character Display	Partially Met	The current program correctly displays the signed characters detected by the model with no problem, they are visible to the suer via the translation bar and cannot be manually edited however since the model can sometimes detect incorrectly for a split second the character the user gets in the translation bar may not be what they originally planned on signing. Originally, I wanted to add a time delay. The model must have detected the character to be displayed on the bar for at least 3 seconds before it is added, and I researched into the time library in python to do this however due to time constraints and how long the solution to this was taking I had to move on. Similarly, I'm yet to add a delete method for signed characters due to time constraints. In the next iteration of the program, I would like to add these methods to make the way that the user experiences the program better than what it is currently. Overall, the module has met the main goals of the criteria which are display the models' detected characters but it could use a lot of improvement.

9	Adding Credentials	Partially Met	The new user window is functioning well within the integrated window and does in fact allow the user to add multiple credentials successfully. It provides functionality and an easy-to-use GUI however there is one bug that make this module only partly successful. Although the user is required to provide names, the username and password fields can be left completely empty. This is a security risk in the program since these new credentials will not require any input from the user, allowing for instant authorisation. The module covers the goal of adding new users correctly and is also extremely fast to load however this bug must be fixed in the next iteration and until then this module can only be classified as partially completed.
10	Delete All Credentials	Fully Met	The module for erasing all credentials provides a swift method for deleting outdated credentials bloating the program. The module presents a choice to the user, it asks the user to confirm the action or return back to the main program. The criteria simply stated that a method for deleting old credentials should exist which this module covers. Testing the module shows that this in fact does work and returns the credentials back to the default values. This leads me to believe that this criterion has been completed and that the module has fully met its needs.
11	Change User Password	Fully Met	The original criteria for this module outlined that a method for changing the password for a specific user must be available to the user. This form provides this method along with a responsive GUI that can be used by the user. All changes are applied once the program is restarted and trying to log in with previously changed passwords is not possible. This goal has been fully met since it covers all the requirements of the criterion.
12	Export File	Fully Met	The export function for the program allows users to save the contents of the translation bar into a file of their choice. Currently there are three available file extensions which all work within the GUI. Opening and handling the GUI for the user is simple, and creating an export file is a fast process. The export file is produces in the directory meaning that the criteria has

			been completely met. If I were to iterate through another cycle I would like to be bale to add a greater number of file extensions and hopefully a file location chooser.
13	Pause Feed	Partially Met	The camera cover widget correctly covers the camera container live feed and responsively switches between the paused display and the continue live feed. This provides the necessary requirements to successfully pass the criteria however due to computational restraints the cover simply covers the feed rather than stopping it and restarting it again. Stopping the capture means that it has to be loaded every time the button press event happens, and this loading of the capture takes more time than it ideally should. As a result, I have opted for covering the camera meaning that it can still capture the characters being detected accidentally while the user moves with the camera cover on. This means that I have only partially met the criterion and hope to improve this in the next iteration of the program.
14	Save File	Not Met	The save file function works exceptionally well when saving the translation bar contents into a text save file, even having a prevention method for empty saving. However, when loading main back up the save file contents appears for only a fraction of time until the translation bar refreshes due to the show frame function in a separate thread. The criteria stated that a files should be saved and be able to be loaded again once the program is opened. I've tried to solve the issue by restricting the way in which it is set however I'm having sequencing issues. The only way to currently fix this issue is to restructure the way in which my code is written which is not possible with the time that I have left to complete the rest of the program therefore this modules targets are not met.

PROJECT APPENDIXES

MAIN.PY

```
import tkinter as tk
from loginform import login
from menubar import menu
import cv2
import mediapipe as mediapipe
from PIL import Image, ImageTk
from threading import Thread
import os
from math import hypot
from toolbarmovement import start_drag, move_window

def main():
    translationlist=[">"]
    mediapipeHands = mediapipe.solutions.hands
    Lhands = mediapipeHands.Hands()
    Rhands = mediapipeHands.Hands()
    mediapipeDraw = mediapipe.solutions.drawing_utils

    def shutter():
        if booleanshutter.get() == True:
            cameracontainer.grid_forget()
            cameracover.grid(row=0, column=0, padx=10, pady=15, columnspan=2)
            booleanshutter.set(value=False)
        else:
            cameracover.grid_forget()
            cameracontainer.grid(row=0, column=0, padx=10, pady=15, columnspan=2)
            booleanshutter.set(value=True)

    def show_frame():
        success, frame = capture.read()
        if not success:
             print("Ignoring empty camera frame.")
        frame = cv2.flip(frame, 1)
        cv2image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        Lfinish = Lhands.process(cv2image)
        Rfinish= Rhands.process(cv2image)
        LlmList = []
        RlmList = []

        if Lfinish.multi_hand_landmarks:
            for handlandmark in Lfinish.multi_hand_landmarks:
                for id, lms  in enumerate(Lfinish.multi_hand_landmarks):
                    lbl=Lfinish.multi_handedness[id].classification[0].label
                    if lbl == "Left":
                        for id, lm in enumerate(handlandmark.landmark):
                                height, width, depth = frame.shape
                                cx, cy = int(lm.x * width), int(lm.y * height)
                                LlmList.append([id, cx, cy])
                                mediapipeDraw.draw_landmarks(cv2image,
handlandmark, mediapipeHands.HAND_CONNECTIONS)
                    if lbl =="Right":
                        for handlandmark in Rfinish.multi_hand_landmarks:
                            for id, lm in enumerate(handlandmark.landmark):
                                    height, width, depth = frame.shape
                                    cx, cy = int(lm.x * width), int(lm.y *
height)
                                    RlmList.append([id, cx, cy])
                                    mediapipeDraw.draw_landmarks(cv2image,
handlandmark, mediapipeHands.HAND_CONNECTIONS)

        if LlmList != [] and RlmList!=[]:
            Lx4, Ly4 = LlmList[4][1], LlmList[4][2]
            Lx8, Ly8 = LlmList[8][1], LlmList[8][2]
            Lx12, Ly12 = LlmList[12][1], LlmList[12][2]
            Lx16, Ly16 = LlmList[16][1], LlmList[16][2]
            Lx20, Ly20 = LlmList[20][1], LlmList[20][2]
            Lx0, Ly0 = LlmList[0][1], LlmList[0][2]
            Lx7, Ly7 = LlmList[7][1], LlmList[7][2]

            Rx12, Ry12 = RlmList[12][1], RlmList[12][2]
            Rx11, Ry11 = RlmList[11][1], RlmList[11][2]
            Rx8, Ry8 = RlmList[8][1], RlmList[8][2]
            Rx7, Ry7 = RlmList[7][1], RlmList[7][2]

            Alength=hypot(Rx8-Lx4, Ry8-Ly4)
            Elength = hypot(Rx8-Lx8, Ry8-Ly8)
            Ilength = hypot(Rx8-Lx12, Ry8-Ly12)
            Olength = hypot(Rx8-Lx16, Ry8-Ly16)
            Ulength = hypot(Rx8-Lx20, Ry8-Ly20)
            Llength = hypot(Rx8-Lx0, Ry8-Ly0)
            Xlength = hypot(Rx7-Lx7, Ry7-Ly7)
            Blength = hypot(Rx12-Lx12, Ry12-Ly12)

            if Alength<25:
                cv2.putText(cv2image, "A",(100, 100), cv2.FONT_HERSHEY_SIMPLEX,
3, (0, 0, 0), 4, cv2.LINE_4)
                if translationlist[-1]!="A":
                    translationlist.append("A")
            elif Elength<25:
                cv2.putText(cv2image, "E",(100, 100), cv2.FONT_HERSHEY_SIMPLEX,
3, (0, 0, 0), 4, cv2.LINE_4)
            elif Ilength<25:
                cv2.putText(cv2image, "I",(100, 100), cv2.FONT_HERSHEY_SIMPLEX,
3, (0, 0, 0), 4, cv2.LINE_4)
                if translationlist[-1]!="I":
                    translationlist.append("I")
            elif Llength<75:
                cv2.putText(cv2image, "L",(100, 100), cv2.FONT_HERSHEY_SIMPLEX,
3, (0, 0, 0), 4, cv2.LINE_4)
                if translationlist[-1]!="L":
                    translationlist.append("L")
            elif Olength<25:
                cv2.putText(cv2image, "O",(100, 100), cv2.FONT_HERSHEY_SIMPLEX,
3, (0, 0, 0), 4, cv2.LINE_4)
                if translationlist[-1]!="O":
                    translationlist.append("O")
            elif Ulength<25:
                cv2.putText(cv2image, "U",(100, 100), cv2.FONT_HERSHEY_SIMPLEX,
3, (0, 0, 0), 4, cv2.LINE_4)
                if translationlist[-1]!="U":
                    translationlist.append("U")
            elif Xlength<50:
                cv2.putText(cv2image, "X",(100, 100), cv2.FONT_HERSHEY_SIMPLEX,
3, (0, 0, 0), 4, cv2.LINE_4)
                if translationlist[-1]!="X":
                    translationlist.append("X")
            elif Blength<50:
                cv2.putText(cv2image, "B",(100, 100), cv2.FONT_HERSHEY_SIMPLEX,
3, (0, 0, 0), 4, cv2.LINE_4)
                if translationlist[-1]!="B":
                    translationlist.append("B")
        translation.set(value=translationlist)
        cv2image = Image.fromarray(cv2image)
        tkimage = ImageTk.PhotoImage(image=cv2image)
        cameracontainer.imgtk = tkimage
        cameracontainer.configure(image=tkimage)
        cameracontainer.after(1, show_frame)

    root = tk.Tk()
    width, height = 800, 600
    x, y = (int(((root.winfo_screenwidth())/2)-((width+75)/2))),
(int(((root.winfo_screenheight())/2)-(height/2)))

    root.geometry('{}x{}+{}+{}'.format(width+75, height, x, y))
    root.title("BSL Comprehension")
    root.resizable(False, False)
    root.overrideredirect(True)
    root.configure(background="lightgray")

    root.bind("<Escape>", lambda e: root.quit())
    root.bind("<Button-1>", lambda e: start_drag(e))
    root.bind("<B1-Motion>", lambda e:move_window(e, root))

    capture = cv2.VideoCapture(0)
    capture.set(cv2.CAP_PROP_FRAME_WIDTH, width)
    capture.set(cv2.CAP_PROP_FRAME_HEIGHT, height)

    cameracontainer = tk.Label(root, relief='sunken',bg="#3676d1", bd=5)
    cameracontainer.grid(row=0, column=0, padx=10, pady=15, columnspan=2)
    cameracover = tk.Label(root, bg="#000000",fg="#ffffff", text="PAUSED",
height=32, width=121)

    translation = tk.StringVar()
    translationresult = tk.Entry(root, textvariable=translation,
relief="sunken",width=68, font=(18))
    translationresult.grid(row=1, column=0, ipady=20, padx=2)

    booleanshutter = tk.BooleanVar(value=True)
    shutterbutton = tk.Button(root,
text="Shutter",bg="#3676d1",fg="#ffffff",font=("bold"),height=2,width=6
,command=lambda:shutter())
    shutterbutton.grid(row=1, column=1, ipady=3)
    filepath = "./savefile.txt"
    if os.path.isfile(filepath):
        savefile = open("savefile.txt", "r")
        translation.set(value=f"{savefile.read()}")
        savefile.close()
    showframeThread= Thread(target=show_frame)
    showframeThread.start()
    menu(root, translation)
    root.mainloop()

login()
main()
```

LOGINFORM.PY

```
import tkinter as tk
from tkinter import messagebox
from HashingAlgo import hash
from IterateAuthorise import authorise
from toolbarmovement import start_drag, move_window
```

```
def init(root,usernameTk, passwordTk):
    passwordPy = passwordTk.get()
    usernamePy = usernameTk.get()
    if usernamePy=="":
        messagebox.showerror("", "Username field can't be left blank")
        return
    if passwordPy=="":
        messagebox.showerror("", "Password field can't be left blank")
        return
    passwordPy = hash(usernameTk.get(), passwordPy)
    booleanreturn = authorise(usernamePy, passwordPy)
    if booleanreturn == False:
        messagebox.showerror("", "Incorrect Credentials")
        return
    root.destroy()



def FocusIn_username(usernameEntry):
    usernameEntry.delete(0, tk.END)
    usernameEntry.config(fg='black')


def FocusIn_password(passwordEntry):
    passwordEntry.delete(0, tk.END)
    passwordEntry.config(fg='black')

def login():
    root = tk.Tk()
    root.title("")
    x, y = (int(((root.winfo_screenwidth())/2)-((220)/2))),
(int(((root.winfo_screenheight())/2)-(165/2)))
    root.geometry('{}x{}+{}+{}'.format(220, 165, x, y))
    root.configure(background="white")
    root.overrideredirect(True)
    root.bind('<Return>', lambda event: init(root, usernameTk, passwordTk))
    root.bind('<Escape>', lambda event: quit())
    root.bind('<Down>', lambda event: passwordEntry.focus())
    root.bind('<Up>', lambda event: usernameEntry.focus())

    toolbar = tk.Frame(root, bg="#3676d1", relief="raised", bd=1)
    toolbar.grid(row=0, column=0, columnspan=2, ipadx=57)
    toolbar.bind("<Button-1>", lambda e: start_drag(e))
    toolbar.bind("<B1-Motion>", lambda e:move_window(e, root))
    title = tk.Label(toolbar, text="Admin Login", bg="#3676d1", fg="#ffffff",
font=("sans", "10", "bold"))
    title.pack(side=tk.LEFT)
    exitbutton = tk.Button(toolbar, text="X", relief="raised", bg="#FF7276",
command=lambda: quit())
    exitbutton.pack(side=tk.RIGHT)

    UserRaw = tk.PhotoImage(file='D:\\Paradigms\\Python\\Programming
project\\UserPicture.png')
    UserPicture = UserRaw.subsample(2, 2)
    PassRaw = tk.PhotoImage(file='D:\\Paradigms\\Python\\Programming
project\\PassPicture.png')
    PassPicture = PassRaw.subsample(2, 2)

    username_icon = tk.Label(root, image=UserPicture).grid(row=2, column=0,
pady=5)
    usernameTk = tk.StringVar()

    usernameEntry = tk.Entry(root, textvariable=usernameTk, borderwidth=3,
fg='grey')
    usernameEntry.grid(row=2, column=1, ipady=5, ipadx=20, pady=5)
    usernameEntry.insert(0, 'Username')
    usernameEntry.bind('<FocusIn>', lambda event:
FocusIn_username(usernameEntry))

    password_icon = tk.Label(root, image=PassPicture).grid(row=3, column=0)
    passwordTk = tk.StringVar()

    passwordEntry = tk.Entry(root, textvariable=passwordTk, borderwidth=3,
fg='grey')
    passwordEntry.grid(row=3, column=1, ipady=5, ipadx=20)
    passwordEntry.insert(0, 'Password')
    passwordEntry.bind('<FocusIn>', lambda event:
FocusIn_password(passwordEntry))

    loginbutton = tk.Button(root, text="Login", bg='#3676d1', fg='white', font=(
        'Sans', '14', 'bold'), command=lambda: init(root, usernameTk,
passwordTk))
    loginbutton.grid(row=4, column=0, columnspan=2, ipadx=70, padx=2, pady=2)
    root.mainloop()
```

LOADSCREEN.PY

```
import tkinter as tk
from time import sleep
from tqdm.tk import tqdm
```

```
def load(root):
    root.attributes('-alpha', 0)
    progressbar= tqdm(total=30,desc='Loading Packages' ,tk_parent=root)
    def bar_init():
        for i in range(30):
            sleep(1)
            progressbar.update(i)
        progressbar.close()

    bar_init()

root=tk.Tk()
load(root)
root.mainloop()
```

HASHINGALGORITHM.PY

```
import base64


def hash(key, password):
    hash = []
    for i in range(len(password)):
        hashkey = key[i % len(key)]
        hash.append(chr((ord(password[i])+ord(hashkey)) % 256))
    return base64.b64encode("".join(hash).encode())
```

ITERATEAUTHORISE.PY

```
import sqlite3


def iterativeSearch():
    DBusername = []
    DBpassword = []
    connection = sqlite3.connect('BSLComprehension.login_credentials')
    transfer = connection.cursor()
    transfer.execute('SELECT username FROM username')
    DBusername = transfer.fetchall()
    transfer.execute('SELECT password FROM password')
    DBpassword = transfer.fetchall()
    connection.close()
    return(DBusername, DBpassword)


def authorise(usernamePy, passwordPy):
    tablevalues = iterativeSearch()
    for i in range(len(tablevalues[0])):
        username = list(tablevalues[0][i]).pop()
        password = list(tablevalues[1][i]).pop()
        if username == usernamePy:
            if password == str(passwordPy):
                return True
    return False
```

MENUBAR.PY

```
import tkinter as tk
from usermenucommands import *
from filemenucommands import *

def menu(root, translationresult):
    mainwindow_menu=tk.Menu(root)
    root.config(menu=mainwindow_menu)

    file_dropdown=tk.Menu(mainwindow_menu, tearoff=False)
    mainwindow_menu.add_cascade(label='File', menu=file_dropdown)
    file_dropdown.add_command(label='New')
    file_dropdown.add_command(label='Open')
    file_dropdown.add_separator()
    file_dropdown.add_command(label='Save', command=lambda:
save(translationresult))
    file_dropdown.add_command(label='Save As...')
    file_dropdown.add_separator()
    file_dropdown.add_command(label='Import')
    file_dropdown.add_command(label='Export',command=lambda: export(root,
translationresult))
    file_dropdown.add_separator()
    file_dropdown.add_command(label='Exit', command=root.destroy)

    edit_dropdown = tk.Menu(root, tearoff=False)
    root.config(menu=mainwindow_menu)
    mainwindow_menu.add_cascade(label='Edit', menu=edit_dropdown)
    edit_dropdown.add_command(label='Undo')
    edit_dropdown.add_command(label='Redo')
    edit_dropdown.add_separator()
    edit_dropdown.add_command(label='Cut')
    edit_dropdown.add_command(label='Copy')
    edit_dropdown.add_command(label='Paste')

    config_dropdown = tk.Menu(mainwindow_menu, tearoff=False)
    mainwindow_menu.add_cascade(label='User', menu=config_dropdown)
    config_dropdown.add_command(label='New User', command= lambda: newuser(root))
    config_dropdown.add_separator()
    config_dropdown.add_command(label='View Password')
    config_dropdown.add_command(label='Change Password', command=lambda:
changepassword(root))
    config_dropdown.add_separator()
    config_dropdown.add_command(label='Erase all Users', command=lambda:
deleteall(root))

    help_dropdown = tk.Menu(mainwindow_menu, tearoff=False)
    mainwindow_menu.add_cascade(label='Help', menu=help_dropdown)
```

FILEMENUCOMMANDS.PY

```
import tkinter as tk
from tkinter import messagebox
from toolbarmovement import start_drag, move_window

def export(root, text):
    def createfile():
        file = open(f'{filename.get()}{filetype.get()}','w')
        file.write(text.get())
        text.set(value="")
        messagebox.showinfo("", "File Creation Successful")
        Exportform.destroy()
    Exportform = tk.Toplevel(root)
    Exportform.overrideredirect(True)
    x, y = (int(((Exportform.winfo_screenwidth())/2)-((180)/2))),
(int(((Exportform.winfo_screenheight())/2)-(100/2)))
    Exportform.geometry('{}x{}+{}+{}'.format(190, 135, x, y))

    Exporttoolbar = tk.Frame(Exportform, bg="#3676d1", relief="raised", bd=1)
    Exporttoolbar.grid(row=0, column=0, columnspan=2, ipadx=60)
    Exporttoolbar.bind("<Button-1>", lambda e:start_drag(e))
    Exporttoolbar.bind("<B1-Motion>", lambda e:move_window(e, Exportform))

    toolbartitle = tk.Label(Exporttoolbar, text="Export     ", bg="#3676d1",
fg="#ffffff", font=("sans", "10", "bold"))
    toolbartitle.pack(side=tk.LEFT)

    filenametitle = tk.Label(Exportform, text="filename:")
    filenametitle.grid(row=1, column=0, pady=10)
    filename=tk.StringVar()
    filenameEntry=tk.Entry(Exportform, textvariable=filename)
    filenameEntry.grid(row=1, column=1, ipady=2)

    options = ['.txt','.docx', '.pdf' ]
    filetypetitle = tk.Label(Exportform, text="filetype:")
    filetypetitle.grid(row=2, column=0)
    filetype=tk.StringVar(value='.txt')
    filetypeOption=tk.OptionMenu(Exportform,filetype, *options,)
    filetypeOption.grid(row=2, column=1, sticky='EW')

    buttonframe = tk.Frame(Exportform)
    buttonframe.grid(row=3, column=0, columnspan=2)
    saveAsbutton = tk.Button(buttonframe, text='confirm', bg='green',
command=createfile)
    saveAsbutton.pack(side=tk.LEFT, padx=20, pady=10)

    cancelbutton = tk.Button(buttonframe, text='cancel', bg='orangered',
command=lambda:Exportform.destroy())
    cancelbutton.pack(side=tk.RIGHT, padx=20, pady=10)
    Exportform.mainloop()

def save(text):
    if text.get() != "":
        savefile= open('savefile.txt', 'w')
        savefile.write(text.get())
        messagebox.showinfo("", "Saved File")
    else:
        messagebox.showerror("","Contents Empty, Unable to save")
```

USERMENUCOMMANDS.PY

```
import tkinter as tk
from tkinter import messagebox
import sqlite3
from HashingAlgo import hash
from IterateAuthorise import iterativeSearch
from toolbarmovement import start_drag, move_window
from loginform import login

def newuser(root):
    def newuserSQLinput():
        newnamelist=newname.get().split()

        if len(newnamelist) >2:
            messagebox.showerror("", "Too many names")
            return
        if len(newnamelist)==1:
            messagebox.showerror("","Please enter first and last name")
            return
        newpassword.set(hash(newusername.get(), newpassword.get()))
        connection=sqlite3.connect("BSLComprehension.login_credentials")
        connection.execute("INSERT INTO username VALUES(:firstname, :lastname, 
:email, :username)",
                   {"firstname": newnamelist[0], "lastname": newnamelist[1], 
"email": newemail.get(), "username": newusername.get()})
        connection.execute("INSERT INTO password VALUES(:firstname, :lastname, 
:password)",
                   {"firstname": newnamelist[0], "lastname": newnamelist[1], 
"password": newpassword.get()})
        connection.commit()
        connection.close()
        messagebox.showinfo("", "Successfully Added")
        newuserform.destroy()



    newuserform = tk.Toplevel(root, bg='lightgray')
    newuserform.title("New User")
    newuserform.overrideredirect(True)
    width, height = 400, 270
    x, y = (int(((root.winfo_screenwidth())/2)-((width+75)/2))), 
(int(((root.winfo_screenheight())/2)-(height/2)))
    newuserform.geometry('{}x{}+{}+{}'.format(width+75, height, x, y))

    newusertoolbar = tk.Frame(newuserform, bg="#3676d1", relief="raised", bd=1)
    newusertoolbar.grid(row=0, column=0, columnspan=3, ipadx=195)
    newusertoolbar.bind("<Button-1>", lambda e: start_drag(e))
    newusertoolbar.bind("<B1-Motion>", lambda e:move_window(e, newuserform))
    newusertitle = tk.Label(newusertoolbar, text="New User", bg="#3676d1", 
fg="#ffffff", font=("sans", "10", "bold"))
    newusertitle.pack(side=tk.LEFT)
    exitbutton = tk.Button(newusertoolbar, text="X", relief="raised", 
bg="#FF7276", command=lambda: newuserform.destroy())
    exitbutton.pack(side=tk.RIGHT)

    newuserformtitle = tk.Label(newuserform, text='Create New 
User',bg='lightgray',font=('Helvetica','30','bold'))
    newuserformtitle.grid(row=1, column=0, columnspan=3, padx=60, pady=10)

    newnamelabel=tk.Label(newuserform, text='Name:',bg='lightgray', font='20')
    newnamelabel.grid(row=2, column=0)
    newname=tk.StringVar()
    newnameentry=tk.Entry(newuserform, textvariable=newname)
    newnameentry.grid(row=2, column=1, ipadx=50,columnspan=2)

    newemaillabel=tk.Label(newuserform, text='Email:',bg='lightgray',font='20')
    newemaillabel.grid(row=3, column=0)
    newemail=tk.StringVar()
    newemailentry=tk.Entry(newuserform, textvariable=newemail)
    newemailentry.grid(row=3, column=1, ipadx=50,columnspan=2)

    newusernamelabel=tk.Label(newuserform,
text='Username:',bg='lightgray',font='20')
    newusernamelabel.grid(row=4, column=0)
    newusername=tk.StringVar()
    newusernameentry=tk.Entry(newuserform, textvariable=newusername)
    newusernameentry.grid(row=4, column=1, ipadx=50,columnspan=2)

    newpasswordlabel=tk.Label(newuserform,
text='Password:',bg='lightgray',font='20')
    newpasswordlabel.grid(row=5, column=0)
    newpassword=tk.StringVar()
    newpasswordentry=tk.Entry(newuserform, textvariable=newpassword, show="*")
    newpasswordentry.grid(row=5, column=1, ipadx=50,columnspan=2)

    createbutton = tk.Button(newuserform, text="Create", bg="darkgreen",
command=lambda:newuserSQLinput())
    createbutton.grid(row=6, column=1, pady=25, ipadx=50)
    closebutton = tk.Button(newuserform, text="Close", bg="orangered",
command=lambda:newuserform.destroy())
    closebutton.grid(row=6, column=2, pady=25, ipadx=50)

    newuserform.mainloop()

def changepassword(root):
    def changepasswordSQLinput():
        oldpass.set(hash(username.get(), oldpass.get()))
        newpass.set(hash(username.get(), newpass.get()))
        tablevalues= iterativeSearch()
        for i in range(len(tablevalues)):
            DBusername = list(tablevalues[0][i]).pop()
            DBpassword = list(tablevalues[1][i]).pop()
            if DBusername == username.get():
                if DBpassword == str(oldpass.get()):
                    connection =
sqlite3.connect('BSLComprehension.login_credentials')
```

```
                    transfer=connection.cursor()
                    transfer.execute('UPDATE password SET password=:formatnewpass 
WHERE password=:formatoldpass',{'formatnewpass':newpass.get(),
'formatoldpass':oldpass.get()})
                    connection.commit()
                    connection.close()
                    changepassform.destroy()
                    messagebox.showinfo('','Password Changed')
        messagebox.showerror("", "Incorrect Credentials")
        return
    changepassform = tk.Toplevel(root, bg='lightgray')
    changepassform.title("Change Password")
    changepassform.overrideredirect(True)
    width, height = 400, 250
    x, y = (int(((root.winfo_screenwidth())/2)-((width+75)/2))),
(int(((root.winfo_screenheight())/2)-(height/2)))
    changepassform.geometry('{}x{}+{}+{}'.format(width+75, height, x, y))
    changepasstoolbar = tk.Frame(changepassform, bg="#3676d1", relief="raised",
bd=1)
    changepasstoolbar.grid(row=0, column=0, columnspan=3, ipadx=168)
    changepasstoolbar.bind("<Button-1>", lambda e: start_drag(e))
    changepasstoolbar.bind("<B1-Motion>", lambda e:move_window(e,
changepassform))
    changepasstitle = tk.Label(changepasstoolbar, text="Change Password",
bg="#3676d1", fg="#ffffff", font=("sans", "10", "bold"))
    changepasstitle.pack(side=tk.LEFT)
    exitbutton = tk.Button(changepasstoolbar, text="X", relief="raised",
bg="#FF7276", command=lambda: changepassform.destroy())
    exitbutton.pack(side=tk.RIGHT)

    newuserformtitle = tk.Label(changepassform, text='Change
Password',bg='lightgray',font=('Helvetica','30','bold'))
    newuserformtitle.grid(row=1, column=0, columnspan=3, padx=60, pady=10)

    usernamelabel=tk.Label(changepassform, text='Username:',bg='lightgray',
font='20')
    usernamelabel.grid(row=2, column=0)
    username=tk.StringVar()
    usernameentry=tk.Entry(changepassform, textvariable=username)
    usernameentry.grid(row=2, column=1, ipadx=50,columnspan=2)

    oldpasslabel=tk.Label(changepassform, text='Old
Password:',bg='lightgray',font='20')
    oldpasslabel.grid(row=3, column=0)
    oldpass=tk.StringVar()
    oldpassentry=tk.Entry(changepassform, textvariable=oldpass, show="*")
    oldpassentry.grid(row=3, column=1, ipadx=50,columnspan=2)

    newpasslabel=tk.Label(changepassform, text='New 
Password:',bg='lightgray',font='20')
    newpasslabel.grid(row=4, column=0)
    newpass=tk.StringVar()
    newpassentry=tk.Entry(changepassform, textvariable=newpass, show="*")
    newpassentry.grid(row=4, column=1, ipadx=50,columnspan=2)

    createbutton = tk.Button(changepassform, text="Change", bg="darkgreen", 
command=lambda:changepasswordSQLinput())
    createbutton.grid(row=5, column=1, pady=25, ipadx=50)
    closebutton = tk.Button(changepassform, text="Close", bg="orangered", 
command=lambda:changepassform.destroy())
    closebutton.grid(row=5, column=2, pady=25, ipadx=50)
    changepassform.mainloop()

def deleteall(root):
    confirmation = messagebox.askokcancel("","Delete all stored users?, Program 
will have to be restarted")
    if confirmation == False:
        return
    connection = sqlite3.connect("BSLComprehension.login_credentials")
    cursor = connection.cursor()
    cursor.execute('DROP TABLE username')
    cursor.execute('DROP TABLE password')
    connection.commit()

    connection.execute('''CREATE TABLE IF NOT EXISTS username(
                        firstname text,
                        lastname text,
                        email text,
                        username text
                        ) ''')
    connection.execute('''CREATE TABLE IF NOT EXISTS password(
                        firstname text,
                        lastname text,
                        password text
                        )''')
    connection.execute("INSERT INTO username VALUES(:firstname, :lastname, 
:email, :username)",
                    {"firstname": "master", "lastname": "master", "email": 
"master@gmail.com", "username": "master"})
```

```
    connection.execute("INSERT INTO password VALUES(:firstname, :lastname, 
:password)",
                    {"firstname": "master", "lastname": "master", "password": 
"b'w5rDgsOmw6jDisOk'"})
    connection.commit()
    connection.close()
    root.destroy()
```

TOOLBARMOVEMENT.PY

```
def start_drag(e):
        e.widget.offset =(e.x,e.y)

def move_window(e, root):
    root.geometry(f"+{e.x_root-e.widget.offset[0]}+{e.y_root-
e.widget.offset[1]}")
```

SQLCREATION.PY

```
import sqlite3

connection = sqlite3.connect("BSLComprehension.login_credentials")

connection.execute('''CREATE TABLE IF NOT EXISTS username(
                    firstname text,
                    lastname text,
                    email text,
                    username text
                    ) ''')
connection.execute('''CREATE TABLE IF NOT EXISTS password(
                    firstname text,
                    lastname text,
                    password text
                    )''')

connection.execute("INSERT INTO username VALUES(:firstname, :lastname, :email, 
:username)",
                   {"firstname": "master", "lastname": "master", "email": 
"master@gmail.com", "username": "master"})

connection.execute("INSERT INTO password VALUES(:firstname, :lastname, 
:password)",
                   {"firstname": "master", "lastname": "master", "password": 
"b'w5rDgsOmw6jDisOk'"})
connection.commit()
connection.close()
```

BSLDETECTIONMODEL.PY

```
import cv2
import mediapipe as mp
from math import hypot

capture = cv2.VideoCapture(0)
capture.set(cv2.CAP_PROP_FRAME_WIDTH, 1920)
capture.set(cv2.CAP_PROP_FRAME_HEIGHT, 1080)

mphands=mp.solutions.hands
Lhands=mphands.Hands()
Rhands=mphands.Hands()
mpdraw=mp.solutions.drawing_utils

while True:
    success, image = capture.read()
    image = cv2.flip(image, 1)
    cv2image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    Lfinish = Lhands.process(cv2image)
    Rfinish= Rhands.process(cv2image)
    LlmList = []
    RlmList = []
    if Lfinish.multi_hand_landmarks:
        for handlandmark in Lfinish.multi_hand_landmarks:
            for id, lms  in enumerate(Lfinish.multi_hand_landmarks):
                 lbl=Lfinish.multi_handedness[id].classification[0].label
                 if lbl == "Left":
                    for id, lm in enumerate(handlandmark.landmark):
                            height, width, depth = image.shape
                            cx, cy = int(lm.x * width), int(lm.y * height)
                            LlmList.append([id, cx, cy])
                            mpdraw.draw_landmarks(image, handlandmark, 
mphands.HAND_CONNECTIONS)
                 if lbl =="Right":
                    for handlandmark in Rfinish.multi_hand_landmarks:
                        for id, lm in enumerate(handlandmark.landmark):
                                height, width, depth = image.shape
                                cx, cy = int(lm.x * width), int(lm.y * height)
                                RlmList.append([id, cx, cy])
                                mpdraw.draw_landmarks(image, handlandmark, 
mphands.HAND_CONNECTIONS)
    if LlmList != [] and RlmList!=[]:
        Lx4, Ly4 = LlmList[4][1], LlmList[4][2]
        Lx8, Ly8 = LlmList[8][1], LlmList[8][2]
        Lx12, Ly12 = LlmList[12][1], LlmList[12][2]
        Lx16, Ly16 = LlmList[16][1], LlmList[16][2]
        Lx20, Ly20 = LlmList[20][1], LlmList[20][2]
        Lx0, Ly0 = LlmList[0][1], LlmList[0][2]
        Lx7, Ly7 = LlmList[7][1], LlmList[7][2]

        Rx12, Ry12 = RlmList[12][1], RlmList[12][2]
        Rx11, Ry11 = RlmList[11][1], RlmList[11][2]
        Rx8, Ry8 = RlmList[8][1], RlmList[8][2]
        Rx7, Ry7 = RlmList[7][1], RlmList[7][2]

        #print(LlmList[4])
        #print(RlmList[4])

        #for i in range(len(LlmList)):
        #    cv2.putText(image, "id: {} ".format(LlmList[i]), (LlmList[i][1],
LlmList[i][2]), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 1, cv2.LINE_AA)


        #cv2.line(image, (Lx4, Ly4), (Rx8, Ry8), (255, 0, 0), 2)
        #cv2.line(image, (Rx8, Ry8), (Lx8, Ly8), (255, 0, 0), 2)

        Alength=hypot(Rx8-Lx4, Ry8-Ly4)
        Elength = hypot(Rx8-Lx8, Ry8-Ly8)
        Ilength = hypot(Rx8-Lx12, Ry8-Ly12)
        Olength = hypot(Rx8-Lx16, Ry8-Ly16)
        Ulength = hypot(Rx8-Lx20, Ry8-Ly20)
        Llength = hypot(Rx8-Lx0, Ry8-Ly0)
        Xlength = hypot(Rx7-Lx7, Ry7-Ly7)
        Blength = hypot(Rx12-Lx12, Ry12-Ly12)

        if Alength<25:
            cv2.putText(image, "A",(100, 100), cv2.FONT_HERSHEY_SIMPLEX, 3, (0,
0, 0), 4, cv2.LINE_4)
        elif Elength<25:
             cv2.putText(image, "E",(100, 100), cv2.FONT_HERSHEY_SIMPLEX, 3, (0,
0, 0), 4, cv2.LINE_4)
        elif Ilength<25:
            cv2.putText(image, "I",(100, 100), cv2.FONT_HERSHEY_SIMPLEX, 3, (0,
0, 0), 4, cv2.LINE_4)
        elif Llength<75:
             cv2.putText(image, "L",(100, 100), cv2.FONT_HERSHEY_SIMPLEX, 3, (0,
0, 0), 4, cv2.LINE_4)
```

```
        elif Olength<25:
            cv2.putText(image, "O",(100, 100), cv2.FONT_HERSHEY_SIMPLEX, 3, (0,
0, 0), 4, cv2.LINE_4)
        elif Ulength<25:
            cv2.putText(image, "U",(100, 100), cv2.FONT_HERSHEY_SIMPLEX, 3, (0,
0, 0), 4, cv2.LINE_4)
        elif Xlength<50:
            cv2.putText(image, "X",(100, 100), cv2.FONT_HERSHEY_SIMPLEX, 3, (0,
0, 0), 4, cv2.LINE_4)
        elif Blength<50:
            cv2.putText(image, "B",(100, 100), cv2.FONT_HERSHEY_SIMPLEX, 3, (0,
0, 0), 4, cv2.LINE_4)

    cv2.imshow("Image", image)
    key = cv2.waitKey(1)
    if key == ord("p"):
         break
```

CNNMODELCREATION.PY

```
import matplotlib.pyplot as plt

from keras.models import Sequential
from keras.layers import Dense, Conv2D , MaxPool2D , Flatten , Dropout
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import Adam

import tensorflow as tf
import cv2
import os
import numpy as np

labels = ['A', 'B']
img_size=200
def get_data(data_dir):
    data=[]
    for label in labels:
        path= os.path.join(data_dir, label)
        class_num = labels.index(label)
        for img in os.listdir(path):
            try:
                img_arr = cv2.imread(os.path.join(path, img))[...,::-1]
                resized_arr = cv2.resize(img_arr, (img_size, img_size))
                data.append([resized_arr, class_num])
            except Exception as e:
                print(e)
    return np.array(data, dtype=object)

train = get_data('D:\Paradigms\Python\OpenCV\Input\Train')
val = get_data('D:\Paradigms\Python\OpenCV\Input\Test')

x_train = []
y_train=[]
x_val=[]
y_val=[]

for feature, label in train:

    x_train.append(feature)
    y_train.append(label)
for feature, label in val:
    x_val.append(feature)
    y_val.append(label)

x_train = np.array(x_train)/255.0
x_val = np.array(x_val)/255.0

x_train.reshape(-1, img_size, img_size, 1)
y_train = np.array(y_train)

x_val.reshape(-1, img_size, img_size, 1)
y_val = np.array(y_val)

datageneration=ImageDataGenerator(
                                featurewise_center=False,
                                samplewise_center=False,
                                featurewise_std_normalization=False,
                                samplewise_std_normalization=False,
                                zca_whitening=False,
                                rotation_range=30,
                                zoom_range=0.2,
                                width_shift_range=0.1,
                                height_shift_range=0.1,
                                horizontal_flip=True,
                                vertical_flip=False
                                )

datageneration.fit(x_train)

BSLmodel=Sequential()
```

```
BSLmodel.add(Conv2D(32,3,padding="same",
activation="relu",input_shape=(200,200,3)))
BSLmodel.add(MaxPool2D())
BSLmodel.add(Conv2D(32, 3, padding="same", activation="relu"))
BSLmodel.add(MaxPool2D())
BSLmodel.add(Conv2D(64, 3, padding="same", activation="relu"))
BSLmodel.add(MaxPool2D())
BSLmodel.add(Dropout(0.4))
BSLmodel.add(Flatten())
BSLmodel.add(Dense(128, activation="relu"))
BSLmodel.add(Dense(2, activation="softmax"))
BSLmodel.summary()

optimisation=Adam(learning_rate=0.001)
BSLmodel.compile(optimizer= optimisation,
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
metrics=['accuracy'])
history=BSLmodel.fit(x_train, y_train, epochs=100, validation_data=(x_val,
y_val))

accuracy=history.history['accuracy']
val_acc = history.history['val_accuracy']
loss=history.history['loss']
val_loss=history.history['val_loss']

epochs_range=range(100)

plt.figure(figsize=(15, 15))
plt.subplot(2, 2, 1)
plt.plot(epochs_range, accuracy, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()

model_json = BSLmodel.to_json()
with open("BSLmodel.json","w") as json_file:
    json_file.write(model_json)
BSLmodel.save_weights("model.h5")
```

```
print("Saved model to disk")
```

Insert as many project appendixes as you need for your project.

These might include, but are not limited to:

- Complete Code Listing (ESSENTIAL)
- Interview Transcripts
- Meeting notes
- Observation notes or questionnaires