Project Sign Language H446-04B WBS 28384 A – Level Computer Science Project June 2021

Contents

3.1 Analysis of the problem

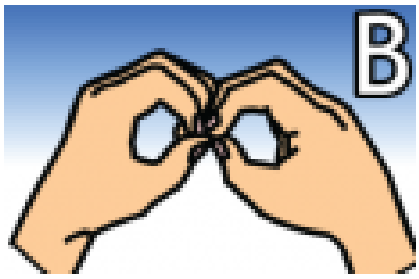
3.1.1 Problem identification

3.1.1a Describe and justify the features that make the problem solvable by computational methods

Sign language is a form of communication involving using a set of visual gestures to convey meaning. Sign language is often used by those that are deaf or hard of hearing. People who must also be fluent are those that surround people who require sign language to communicate. Other people who may wish to learn sign language include medical professionals or family members of someone who uses sign language as well as enthusiasts wishing to learn as a skill. Learning sign language can often be tough given that it requires a partner to communicate and practise with, correcting each other as you go along. Not having a partner makes it difficult to learn this complex skill by yourself.

Below are examples of typical tools of learning for someone studying sign language on their own. Top results include YouTube videos of key phrases (100 Basic Signs in British Sign Language BSL - https://www.youtube.com/watch?v=gMNHvXSW4iE&t=44s) as well as sign language dictionaries returning short videos of the searched phrase as well as related phrases.

Disadvantages of using YouTube Videos - YouTube videos may only show essential signs for phrases and are quite limited in their ability to expand the learners signing ability after a given point



Disadvantages of using Printable Signing Pictures - More basic signing dictionaries can output phrases letter by letter using sign pictures. This is very limited and is usually only useful for beginners.

Disadvantages of using Sign Language Dictionaries - Dictionaries are useful for searching for specific phrases that cannot be found in online videos, however, their phrase set is not all-encompassing. All phrases must be searched individually – does not take full sentences.

To try and combat these cons presented, one solution that my program will try to solve will be to use computational methods to compile the different mediums of content into one place that allows user to search through phrases with more freedom to choose the display format of picture/video as well as the type of sign language(BSL, ASL, etc). To maximise efficiency, the program would analyse the user's actions and outline some sort of feedback for the user's performance, allowing them to learn from their mistakes. Allowing the user to input their phrase via microphone further integrates the users reference to sign language creating more links to the signing action and speeding the learning process.

3.1.1b Explain why the problem is amenable to a computational approach

Currently the main problem that my program will solve is trying to become fluent in sign language on your own. Learners are reliant on Becoming fluent in sign language is a two-step process including: learning and practising. This two-step process is amenable to a computational approach given the huge, complex variety of resources for learning and lack of resources to help practise sign language that can be sorted into a more understandable and efficient format based on user input and machine learning.

Learning

Signing formats:

The learning portion of the program with compile a page of signs that form the sentence given using a GUI. Using a GUI allows me to present the multiple forms of data side by side each other allowing the user to acquire information whilst controlling the positioning of information as to not seem overwhelming to a new learner while still delivering key information for the searched phrase. I would only be able to do this using a computational approach by storing the data in a database that is categorised for optimised linear searches allowing the program to find the desired sign. Using a database also allows me to store a large set of data that is easily accessible. Another advantage of using a database having multiple databases which allows me to swap out a database for one containing another language.

Multiple language input and output:

To allow the user to fluently access the wanted phrases, microphone input will be advised. This will use the user's microphone via pre-existing speech_recognition module to input audio as audio data. Using this module also gives the ability to transcribe multiple languages. This elevates the client-base to allow me to reach more people. The transcribed text can be passed through a translator using a google translator API, converting the text into English which can then be used to search through the databases as normal. As mentioned before, using a computational approach allows the use of not only multiple input languages but also multiple output sign languages using multiple databases.

Practising

Personalised progress:

An OOP framework can be used to create a user login system so that the progress of each user can be saved independently of other users. Having multiple users stored in the same place with all the data accessible also creates the opportunity for leader boards and progress tracking over time for each user. Using an OOP framework allows for many users to be created from the same skeleton. Furthermore, this means that each user can be sorted and analysed in a universal way. The computational approach allows for advanced analysis of progress data and allows the user to view this data for constructive purposes.

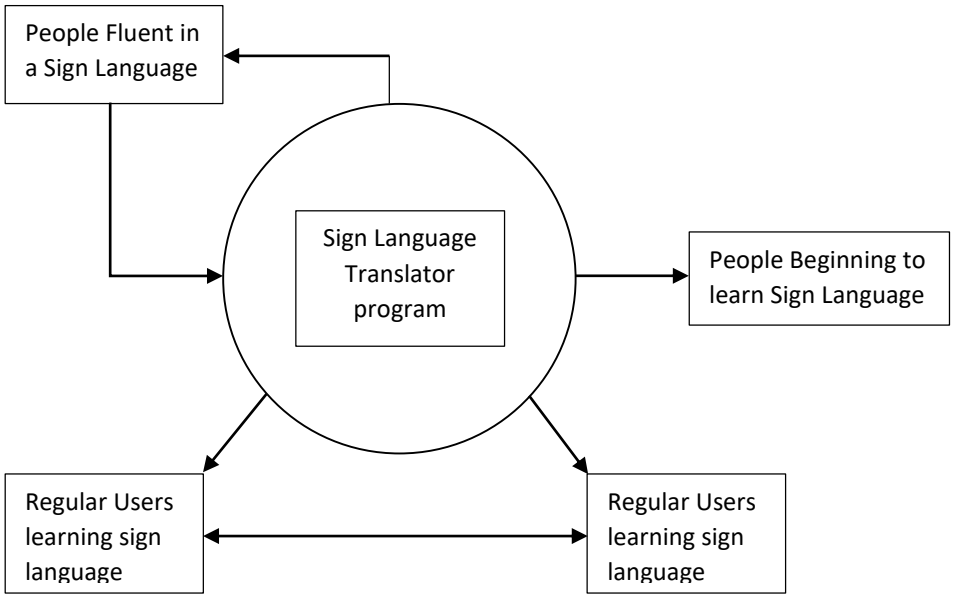
Peer assessment:

The premise of the peer assessment functionality is to allow to users to communicate in order for them to learn together. Given this is a large problem to solve computationally, a divide and conquer approach would be the most efficient way to develop it. Peer assessment functionality works by pairing one user with another user of similar ability and allowing them to "test" each other's sign language. A computational solution would allow the analysis of progress data and would be able to cross reference profiles via an automated selection process to find two users with similar progress. As a user progresses this arrangement can be changed given that people may progress at different rates and therefore must maintain a partner within the same range of capability. Once the pair has been established the user will be prompted to record a video of themselves signing a phrase. The prompted phrase will have been chosen at random from a stored array. The video can then be uploaded to an online database where the partner can access and watch it. The partner can then give feedback on the signing through a textbox in the program.

3.1.2 Stakeholder

3.1.2a Identify and describe those who will have an interest in the solution explaining how the solution is appropriate to their needs (this may be named individuals, groups or persona that describes the target end user

The sign language translator project is a program made for individuals that wish to learn sign language. The program will be available to the general public as a free resource to help further learning for educative purposes. The program requires an input of what the user wishes to know and in turn outputs the corresponding signing for the phrase. The program also has functionality to "test" the user by analysing the sign language performed and analysing this data to present a progress projection to the user. This functionality is appropriate to a beginner as it allows them to learn and practise sign language on their own by using whichever features they find most useful.



The above diagram presents the relationship between the types of users inside of the system.

User	Need	Justification
People beginning to learn sign language	Beginner users will be limited to the basic functionality such as searching in multiple languages and being output sign language in their preferred language.	Beginners without user accounts have no way of storing any progress and so can only access the essential learning tools
Regular users learning sign language	Regular users need to be able to access all basic functionality. Regular users need to be able to view their progress, gained through testing, over a period of time chosen by them. Regular users should be able to communicate with other users of similar ability to ask/answer questions.	Regular users with accounts have the ability to store progress and so can utilise extra functionality such as communicating with other people of similar skill.
People fluent in sign language	Fluent users will have the same needs as the regular users. These users also should have the ability to upload their own signings if they think better videos are needed	Fluent members with longer user history will have higher privileges. This will be judged based on their stored progress. Dedicated users can like/dislike videos based on usefulness or upload a more useful one.

3.1.3 Research the problem

3.1.3a Research the problem and solutions to similar problems to identify and justify suitable approaches to a solution

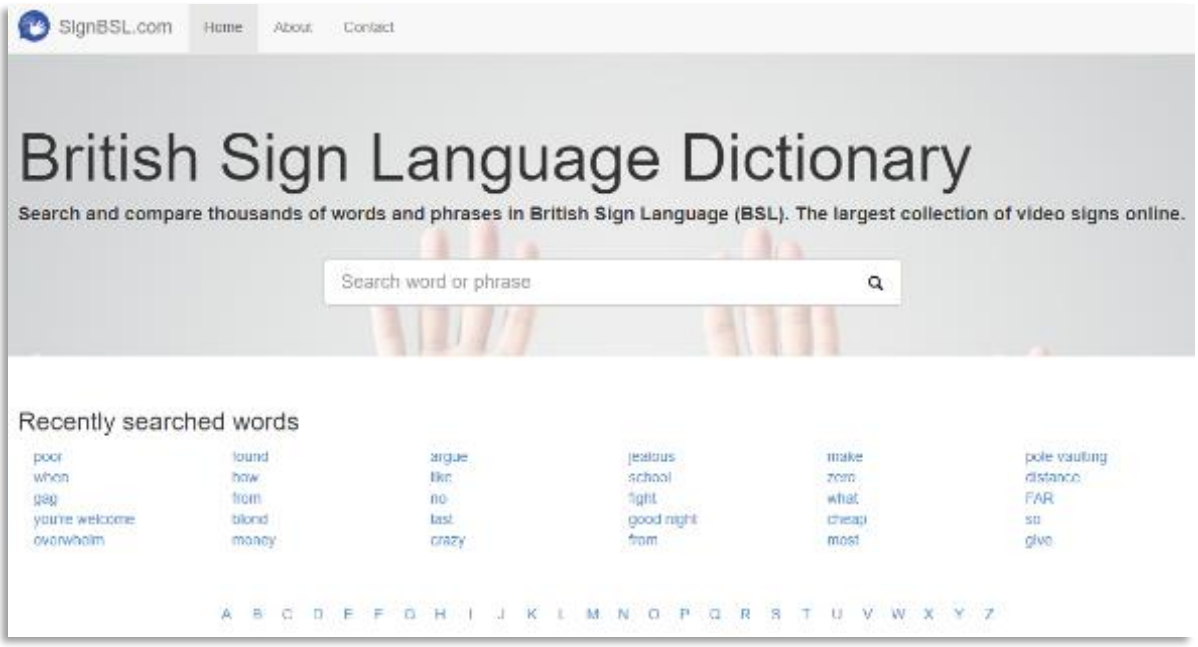
For my project I have split my problem into two parts: learning and practising. As my program is a source of information to help people become fluent in sign language it must be able to take input from the user in the form of a phrase to be translated into sign language. The program would then display the signing for the phrase in the form of a GUI.

Product 1 – YouTube videos

YouTube is a free online video sharing platform that allows users to share videos with others. These videos often contain information in the form of tutorials. In this regard sign language videos for beginners can be found. When searching for sign language tutorials videos such as this (100 Basic Signs in British Sign Language BSL - https://www.youtube.com/watch?v=gMNHvXSW4iE&t=44s) can be found. The signings are relevant, useful and well presented, with this particular video showing the signing from two different angles for the viewer along with subtitles with a large font to tell the user the translation.

YouTube videos can be a very useful source of information especially when of good quality, however, they are also vague in their progression – they do not offer a learning syllabus meaning the user is left to survey arbitrary videos for material to practise. This leaves the user without a systematic learning experience and only contributes to the users learning experience for the duration of the video before the user inevitably forgets the signings.

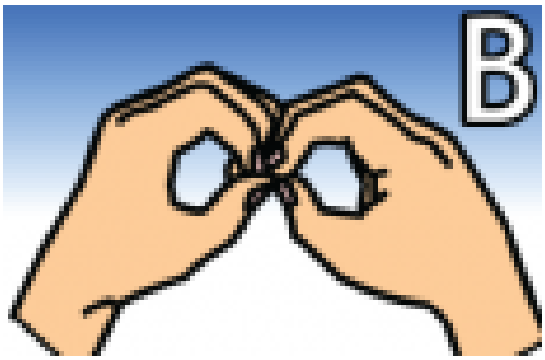
Product 2 – Sign Language Dictionaries



Sign language dictionary websites allow for users to search specific phrases and words in a search engine that will look for the phrase in its database and output it if it can find it. The output is presented as a short video that shows the movement for the signing. These videos can be accessed from searching through the search engine or by going through the alphabet at the bottom and sifting through the pages like a traditional dictionary. A nice additional feature on the home screen is the recently searched words that presents a live, most recently searched words that have turned up a result on the website. The fact that it is a website also means that it can be accessed from a large variety of places so long as the device has access to the internet.

However, in a similar way to the YouTube videos, the website has no log of what you have or haven't searched already and so only acts as the way a physical dictionary would in that it is not intuitive with the user unless the user directly inputs what they want to know. While the recently searched words are a good place to start for relevant phrases that are useful, from that point there would be no sense of direction for a beginner on where to go.

Product 3 – Printable Signing Pictures



Printable signing pictures can be found in a variety of places on the internet and are almost always free. They provide point of view signings, usually of the alphabet. These two things make this particular product good for beginner learners of sign language as they provide a starting point for learners to get their feet wet without the intimidation of huge amounts of signing movements at once. However, these pictures are usually limited to their phrase range that they display and can only display movement through arrows. This can make certain difficult movements such as the letter "S". Furthermore, pictures can often be cropped when copied or printed and leaving out the language in which they are a part of. This could cause unexpected confusion in users if they began learning the wrong sign language.

Comparison

To be able to draw a direct comparison between these relatively different products, each is scored on the following features, with **GREEN** contributing 2 points, **YELLOW** 1 point, and **RED** 0 points for a maximum of 20:

	Project Sign Language	YouTube Videos	Printable Signing Pictures	Sign Language Dictionaries
Cost	Free	Free	Free	Free
Searching capability	Capable	Difficult	None	Capable
Multiple signing languages (output)	Capable	Capable	Difficult	Capable
Multiple input languages	Capable	None	None	None
User Progress chart	Capable	None	None	None
Responsive GUI	Capable	Only basic functionality	None	Capable
Audio searching capability	Capable	None	None	Not in-built
User signing analysis	Capable	None	None	None
Community networking	Capable	Limited – only through comments section	None	None
Uploading/editing material	If user has editing privileges	None	None	None
Total Score:	20	7	3	9

It can be concluded from this comparison that my planned program has considerable advantages over the other solutions to this problem. The main advantage that my program holds over the competition is that it is tailor made for learning sign language. It removes the vague signing of phrases in place for pure learning in a structured and systematic manner. With unique features such as the ability to switch the input language for audio as well as the output language for signing, my program provides a larger client base that can therefore help more people become fluent in sign language.

3.1.3b Describe the essential features of a computational solution explaining these choices

A computational solution would include the following features:

- **Searching specific phrases** – A key feature of the program will be to allow the user to search for specific phrases using a search engine. This would iterate through the local database and select the corresponding phrase and outputting it to the user. The use of a computational method means that this process of iteration and selection can be done individually for each word and be joined, creating a signing phrase (a set of multiple signing actions) conforming to the users input. A computational method also allows this action to be carried out as many times as the user wishes.
- **Voice Search** - The program should give the user the choice to input phrases via spoken English to be searched. This should present a preview of what was said in the search engine before the user manually presses the search button. This will work by exporting the audio data to a google API and receiving the transcribed text in return.
- **Range of sign languages** – The program should advertise functionality of the ability to search one phrase in multiple sign languages. This should be easily accessible by the user via a drop-down box with all the available sign languages. Switching between the options should cause the program to search the respective database for that particular sign language while still using the same iteration and selection process of the original database.
- **Multiple user system** - The program should allow for multiple users to utilise the program. Using a computational solution this can be done via an OOP paradigm with each user as a separate object. Each user can therefore have their own data stored separately from other users via data stored in attributes of the object as well as via databases by using pointer attributes.
- **Confidence rating** - A user system will be created that allows users to rate whether they feel confidant or not with a certain action (signing a certain phrase). This can be done using a simple dropdown box with the level of confidence depicted by each option. Using a computational solution allows this data to be stored within a normalised database in direct correspondence with each user object, allowing for the unique tailored learning experience each time the program is used.
- **Tailored learning for beginners** – A good way to start learning sign language is to begin with fundamental signing phrases that can be often utilised in casual conversation. However, to get to a perfect set of phrases a place to start may be to sort through the phrases in the database via ‘number of times searched’ using a sorting algorithm and selecting the most searched for and creating an initial array of phrases. From here someone can manually add and subtract signs that are more or less useful, respectively.
- **Tailored consolidation of learning** - Based on the confidence rating for each user the program should determine whether the phrase should be presented more or less often. The program can change the rate of presentation of a certain phrase by first creating an initial array of phrases for the user to learn and adjusting this array based on the feedback from the user.
- **Data visualisation** - graphical presentation of progress over time. This allows the user to easily view their progress and act on this data as feedback. There should be options within this feature to adjust the time frame (e.g. daily). Data stored in the database could be date stamped to allow for the user’s progress data to be sorted chronologically to then be presented within the given date range.
- **Peer assessment** – This component of the program would work by comparing two users that have studied similar phrases and finding those with similar progress. These users can then be paired together to “test” each other’s sign language. Ideally this would be self-contained within the program. Pairing the users via their ability will be much easier once a standard set of initial phrases to learn is produced. It may be that these paired users can then continue to work together.
- **Sign language to text translation** - The program should allow the user to convert hand gestures into text to allow the user to test themselves. This would require the program to 3D track the user and analyse their movement patterns and compare the similarities to a pre-set of the action. This would require multiple motion captures of the signing being done correctly to which the program can compare the user’s attempt to in order to prepare an assessment.

3.1.3c Explain the limitations of the proposed solution

Aspects of the program that may encounter limitations:

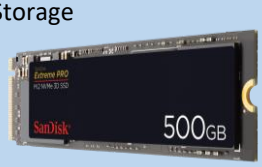
- **Limited resources** - There are bound to be phrases that are not present in the databases, meaning they will not be available to the client.
  Ways to combat this:
  - Store the user searches that could not return a result. This should be regularly checked and tried to be solved. However, it is important to point out that the main purpose of the program is as a learning tool for sign language with the inbuilt dictionary being a side benefit. The key aspect is to have essential phrases needed to communicate in sign language.
- **Limited searching capability** – when searching for certain sign language actions, the user may type in the phrase incorrectly. This may make it hard for the program to search through the database for the desired phrase.
  Ways to combat this:
  - Use the microphone to input the phrase, however, this may also encounter problems with mumbling or even hardware issues.
  - Another way the program should be designed to help combat searching issues is to make the search case insensitive.
- **True progress evaluation** – Given that each user starts off from the same starting point, some users that have more experience may find it frustrating being asked "beginner" phrases over and over again as the program seemingly tries to instil fluency in a beginner learner. There is no way for the program to measure the user's true capability to carry out a signing action correctly on command without looking at the screen.
  Ways to combat this:
  - Incorporate an "I am fluent in this" slider for each phrase to convey to the program that a particular signing is only to be shown every so often. This slider should be stored in relation to that user for that specific phrase.
  - Have fluency levels for the user to choose from when the user is created. From this the program can interpret that a user would practise less "beginner" phrases and more abstract phrases. However, this piece of functionality requires separate sets of phrases for different levels of fluency which may be difficult to create due to time constraints.
  - To try and instil true fluency in a learner (to be able to sign on command) a method of Look, Cover, Practise, Check. This is a common way to practise a skill on your own and can be utilised here by at first covering the action and only showing the "name" of the signing. Once the user is satisfied the user can click to reveal what should have been signed. At this point the program should ask whether they user got this correct or incorrect and store this data to be utilised later.
- **Multiple, complete sign language databases** – due to time constraints functionality/catalogue of phrases may not be complete. While I intend to have more than one database with some basic phrases in each database, it may be that the collection is not exhaustive of the language.
- **App development** – App development is a large and complex skill set that can take a long time to master. While there are lots of resources online from which one can learn app development, the level of mastery needed to complete this program may take time that is better utilised to strengthen the capabilities of my program.
- **Sign language to text translation** - Due to time constraints it may be that I am unable to complete the full integration of the sign language to text feature. This may prove difficult due to the articulate nature that sign language possesses. Because of this a high definition motion camera may be necessary which are often expensive to buy. While some softwares allow for motion tracking using normal cameras, this can often be imprecise for certain hand gestures. If this is deemed too large a task towards the end of the project, I will use the time to further improve the peer-assessment functionality to compensate for the evaluation functionality lost.

Phase 2: A second phase of the program would include some of the following feature: a fully functional app that contains full integration and functionality with the ability to display the user's progress data on both the app and the desktop program. Extra time can be used to solidify features such as the sign language to text translator as well as an inbuilt user-to-user messaging system that allows learning pairs to communicate internally.

3.1.4 Specify the proposed solution

3.1.4a Specify and justify the solution requirements, including hardware and software configuration (if appropriate)

My program uses mostly simple computing instructions that are not hardware intensive meaning it does not need specialised equipment making it suitable for a larger client base.

Hardware	Minimum Spec	Recommended Spec	Justification
CPU 	Intel Core i3 Cores: 2 Clock speed: 1.5GHz	Intel Core i5 Cores: 2 Clock speed: 2.5GHz	Given the lack of intensive processing power needed a low power, efficient CPU is optimal. However, to reduce latency and loading times a higher clock speed CPU is recommended
RAM 	Amount: 1GB	Amount: 2GB	Not much RAM is needed for the program, however, some space is needed to load databases into the program. Larger databases can benefit from the recommended amount.
Storage 	Size: 2GB Type: HDD	Size: 3-4GB Type: SSD	Some space is initially needed to store progress data in databases and the program. As the database grows, user may benefit from using recommended amount. Using an SSD ensures quick loading times.
Monitor 	Size: 18.5” Screen resolution: 1366x768p	Size: 21.5” Screen resolution: 1920x1080p	A larger monitor is preferable to be able to better see the sign language actions and to interact with the GUI.
Mouse 	N/A	N/A	A pointing device in necessary to navigate the GUI through buttons, etc.
Keyboard 	Type: membrane	Type: mechanical	A keyboard can be used to search for phrases. Mechanical is recommended for comfort.
Microphone 	Type: Inbuilt microphone from computer	Type: Headset microphone	Using a working inbuilt microphone is fine, however a headset microphone may be more suitable to ensure good audio detection from the program.

Furthermore, given the simplicity of the program, there are only a few softwares needed for the program to run. A lot of functionality of the program is derived from python modules that are built-in or can be installed. This allows for a contained program with less chance of disruption as well as making it easy to maintain.

Software/Capability	Justification
Python 3.7 	My program will be built using Python 3.7. Using python as my programming language allows me to utilise its vast libraries and use modules such as the speech_recognition module. Using GUIzero I will develop a graphical user interface that will allow the user to interact with the program.
Windows 10 or higher. **Mac OS 10.9 or higher.** 	Given that the program utilizes Python 3.7, it is needed to run the program. Python 3.7 can only by installed on Windows 7 or higher or Mac os 10.9 or higher. Having these operating systems with this setup ensures that the user will get the original user interface that was intended without any interference.
SQLite 	SQLite is a database engine supported by Python. SQLite databases will be used to store the users progress data which will be linked to the user object via a pointer.
Speech Recognition Python Module 	Similarly to SQLite, the speech recognition module of choice called “speech_recognition” is an imported module. The module allows the python program to convert audio from a microphone or external .wav file (or similar audio file) and convert into audio data. The module contains functions that can use this audio data – e.g. can utilise the audio data and a Google translation API and receive transcribed text in return.
Internet connectivity 	The program will use data from APIs to allow for special functionality such as the speech recognition as well as searching sign language dictionaries. These APIs are online resources and require an internet connection to reach them. This means the user’s device must be able to connect to WIFI/a data service such as 4G.

3.1.4b Identify and justify measurable success criteria for the proposed solution

For a success criteria to be effective, each criterion must follow the SMART acronym – Specific, Measurable, Achievable, Realistic and Time-phased. Adhering to these key pillars will allow me to build a solid base of criteria that is likely to be met. Testing my program against each of these criteria will allow me to assess how well my program has succeeded at solving my problem sufficiently.

By June 2021, the project's deadline, my program will hopefully meet the following success criteria:

Number	Success criteria	Justification	Test
SC 1	User will be able to search a phrase and get the result for that phrase.	It is important part of the program that the user is able to search for phrases that they do not know how to sign.	Search the phrase "good morning". This should output the corresponding signing action labelled "good morning".
SC 2	User will be able to search for a sentence and get the result for the sentence	It is important part of the program that the user is able to search for sentence that they do not know how to sign and receive the signing for the full sentence	Search the sentence: "good afternoon, good morning and goodnight". This should output the corresponding signing for the whole sentence – all of the sentence that can be found.
SC 3	Phrases that cannot be found should output an error message.	There may be phrases that are not in the database. On the other hand, the phrase may not be able to be found due to miss spelling, etc.	Search the string "fsdh". This should output an error message telling the user it could not be found
SC 4	User will be able to create a user profile with a username and password.	To ensure that the user's data can be stored and kept for the next time.	Create the following user: First name: "John" Last name: "Doe" Username: "john_doe01" Password: "Hello_world12" Fluency: "beginner" After making the user, try and log out then log back in again. Check to make sure all the details are the same.
SC5	User will be able to conduct a voice search.	Being able to search for a phrase via spoken English was a key piece of functionality and served as the basis of the program which was built upon. Being able to search via voice directly ties into the user's ability to directly link a signing action with the audio counterpart of that phrase. This is all in the service of further the learning of the user.	Click the microphone icon to start the voice search and say "good morning" into the microphone. The search result should display the corresponding signing action for the phrase labelled "good morning"
SC 6	User should be able to check their daily… progress that they made.	It is important for users to understand their progress for new skills as well as old skills. Showing progress not only reinforces the user's achievements but also shows them where they may need improvement. Furthermore, a progress map also encourages users to continue using the resource in order to keep a daily progress and keep improving.	Using the "john_doe01" user, attempt the "learning option" and randomly click "I got it right" and "I got it wrong" options for the shown phrases. Once satisfied, navigate to the "daily progress" page to view the progress that was completed. This should be an easy to read source of information in the form of a graph/chart.

Number	Success criteria	Justification	Test
SC 7	Users will be able to change the time frame of the progress data that they wish to see.	Adding on to SC 8 the user also needs to be able to change the time frame allowing them to see a greater view of their progress. This is important in all users but more useful to users that use the program more over a longer period of time.	Again, using the "john_doe01" user, navigate to the progress page. Switch between the time frame settings from "daily" to "weekly" and "monthly". There should not be any data here yet, but some data should be shown from the progress made beforehand.
SC 8	Users should not be able to create two accounts with the same username but the program should still accept users with the same first and last name.	Two users may try and create an account with the same username. This may cause confusion or even errors in the program and so should be prohibited. However, the program should still accept users with the same first and last name by using the username as the unique attribute.	Attempt to make the exact same user that was made SC 4 again. The program should output that the username is already taken. With all the details the same, attempt to remake the SC 4 user with the username "john_doe02" instead. The program should allow a brand-new user to be created.

These Success Criteria are the principle to which my final program will be weighed against and define the success of my program. The specified tests present how easy my program should be to use. This is a key part of its function as it is important that no piece of functionality is obfuscated and unreachable by the user. Completing these success criteria confirm that my final program outweighs the examples shown in 3.1.3a. these points will play a key role in the development of the program as its ability to achieve these success criteria will dictate the outcome of the final product.

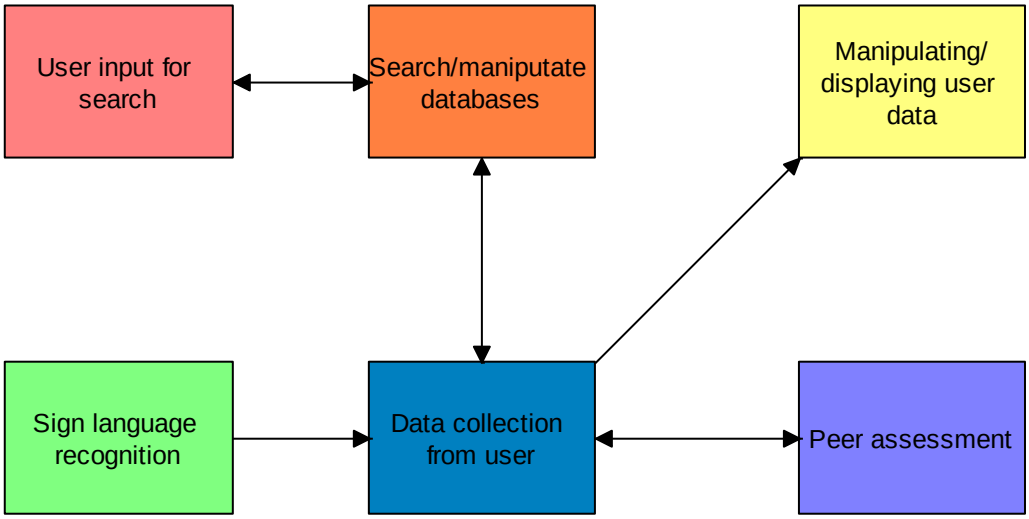
My success criteria will be tested in 3.4.1a at the end of development.

3.2 Design of the solution

3.2.1 Decompose the problem

3.2.1a Break down the problem into smaller parts suitable for computational solutions justifying any decisions made

To maximise the efficiency of development I will decompose my program into various solutions to many problems that will ultimately form the program that is the final product. Each module will be made in a way where they can be developed and tested individually to then be put together to form the final program. Each module represents a functional chunk that can work independently of other modules to carry out a specific function.



The need for modularity:

One of the advantages of using a modular approach means that each process is contained within its module. This means that if a module is corrupted it will not affect the whole program but only the corrupted module. This makes future maintenance easier in the case of breakage. Another advantage is less errors returned due to variable confusion within the program. This could happen if two variables are named the same and values become mixed causing errors. Modularity ensures that only data is shared between functional chunks, minimising errors, and making future maintenance easier.

Description	Justification
Data collection from user - Data collected from the user (e.g. entering their confidence rating of a signing phrase) will be used to measure the progress of the user and allow for the program to tailor the user's learning experience.	The data collection module will act as a hub for the user's progress and allow for other modules to pass through data that can then be transferred to the database module to be stored or in reverse with data being received from the database. Having a separate module for data collection means that a standardised system can be used to translate raw data from the user's ratings to processable data that can be stored. This data can be stored in an OOP framework for users allowing for the data to be easily accessed. This processed data can also be understood by the module when necessary and then displayed to the user. As well as editing data within the user objects this module can also be responsible for creating and deleting users.

Description of module	Justification
Search/manipulate databases - Multiple sign languages as well as user data will be stored using SQLite module for database storage. Databases can be searched through to find specific signing results to return.	Databases can be used to store multiple sign languages using SQLite given that the module is standardised, each sign language can have a separate database that can be easily switched. Furthermore, user profiles can be stored within a database with their data in a standardised form that allows to program to analyse the data systematically allowing for comparable results between users with their progress stored chronologically. This standardised format allows for only the essential information to be stored and reducing the total size of the databases.
Manipulating/displaying user data - Manipulating the data collected from the user allows the program to create a more tailored learning experience for the user and more efficient learning. This data and progress can then be acquired, processed, and displayed for the user to see and track.	This module woks in tandem with the data collection module. Data collected from the user will be passed to this module for processing. This allows this module to systematically regulate an array of signing phrases for the user to learn based on whether the user needs to practise a certain phrase more often. The processed information can then be presented to the user, using rating system of arbitrary values assigned to specific phrases. From this the module could present the data graphically using methods such as line graphs showing the users progress in an easily understandable way. This arbitrary value will be specific to this module and is presented to the user via the GUI. This module acts as a function of the main program but makes sense to segregate from the main program in order to decompose the problem.
User input for search - For dictionary functionality the user must be able to input the desired phrase that they wish to know. This can either be done by typing in the phrase or by using a microphone and searching via voice.	Having a search option that uses microphone input from the user requires a separate module to distinguish the speech_recognition code from the normal code. This is because the speech_recognition module uses specific and complex code to enable access to google APIs. For this reason, it makes sense to segregate this code alongside the text input to input the data into the system as a string.
Peer assessment - This part of the program would use video modules to import a user's attempt at signing a phrase and send this video for the user's pre-set partner to review. This module would also be the part of the program to assign a user a partner by analysing the progress of other users.	This module would need to obtain data that is normalised by the data collection module which obtained the data from the database. This normalised data will make it easier for the module to utilise the otherwise obfuscated data that would be stored in the database. Working with this data the module would analyse patterns in learning curves between different users and use this information to create partners. Using specific video capture modules to record the video and then send it to the specified partner would need to be segregated due to the unique nature of the video module code.
Sign language recognition - This will analyse the user's movement to determine if the user is signing correctly.	Sign language recognition will use highly specific code using motion capture to analyse the user's movements in order to calculate their fluency. This code is highly complex and needs to be segregated in order to maintain smooth function.

Maintenance:

As the program begins to age problems may begin to occur with the program as modules become deprecated and becomes outdated. To continue the lifespan of the program, these modules would have to be updated or replaced with another similar module to carry out the function. This is much easier to carry out with the implementation of modules. Firstly, modularity allows for the problem to be instantly narrowed down to the specific point of error for faster recognition of the problem and its source. Secondly, modularity allows for the code to be extracted from the main program, fixed externally, then re-implemented into the main program.

Using separate modules for the separate functions means that the entire program is not compromised if one part has a problem. Instead the user can freely use the rest of the program while maintenance is carried out.

To aid with this, it is important that the source code its clearly commented with descriptive identifiers. This will allow someone that may be other than the developer to understand the code in order to fix the problem. Additionally, it is important that the code itself is segregated into an understandable way (e.g. imports and constants at the top with the main code below). This makes the code much easier to understand with a coherent order to the instructions. Using constants also allows for future maintenance by adjusting the values to better suit the need of the program.

Scalability:

The program has a lot of room to scale up given the wide range of topics that this program can cover. The program can also be scaled in accordance to the number of users that utilise the program. If a large enough community is produced it may be that the program can be updated to better accommodate the users by using different, more efficient modules such as online databases and implementing new features. These features may have been suggestions from the community and can be implemented in future updates as separate or updated modules. Most notably is the addition of more phrases in more sign languages as this would be the main focus of the program to expand its client-base.

Scaling down is also important to consider given the change of circumstance as the program is developed or while in use by the user. This may be due to a flawed or useless feature that causes congestion to the program's main purpose. scaling down can also be used to ease the user out of a certain feature that has become less preferable. For example if the feature is to be replaced by another feature that has the same functionality it may be preferable to scale down the original feature and allow the users to adjust their approach and habits within the program to account for the change in a controlled way. This logic can also be applied to the program as a whole. As the program ages, more efficient and practical modules may become apparent to replace the original modules. Use of modules containing each functional chunk means that this update can be done in a systematic way that continues its ease of use for the user whilst upgrading the program. Using the original skeleton will allow for this to be done in a more efficient way.

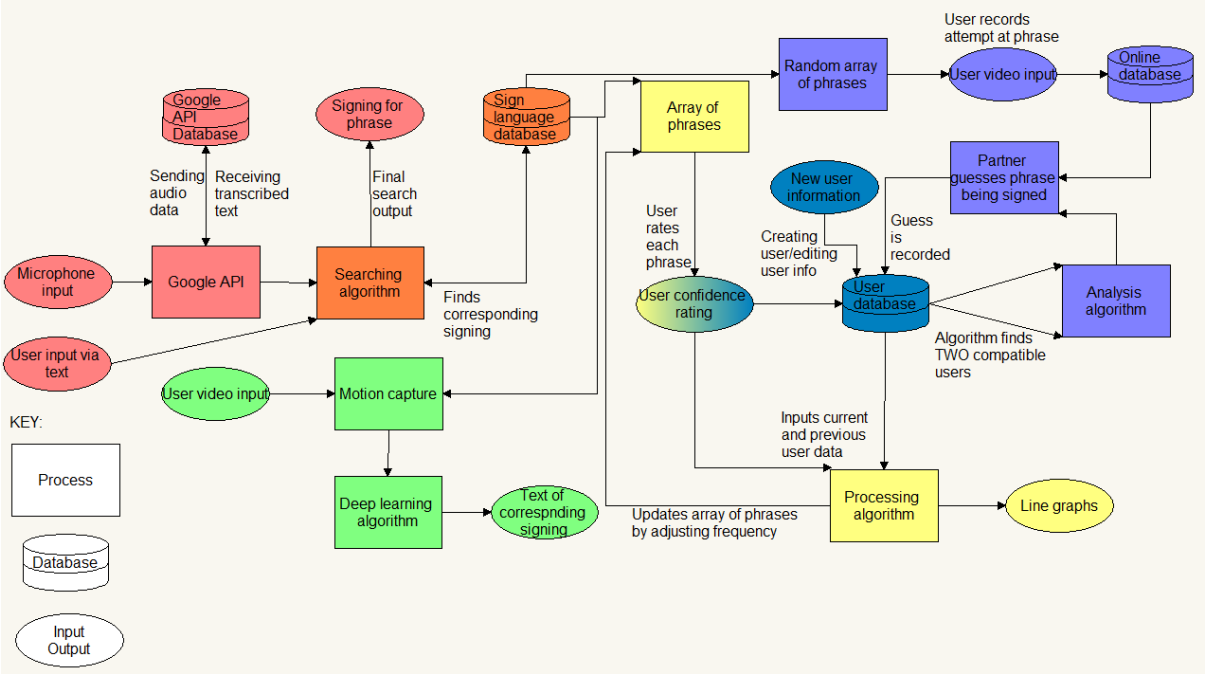
Phase 2:

Phase 2 presents an opportunity for further development of the program on top of the foundations laid in phase 1. For this reason, it is important for a strong framework to be built initially to ensure that the additional features can fit into the program with the least resistance. Some of the features include:

- Additional sign languages – other than BSL the program could incorporate databases for other sign languages such as ASL and ISL which will allow for the expansion of the client-base.
- Mobile app – a mobile app is an extremely convenient way for users to interact with the program as it makes the program omnipresent in the user's life. Using an app also means notifications can be sent to the user to remind them to practise and ensures the maintenance of the client-base and stops users from forgetting about the program.
- Email notifications – emails can be used to advertise features of the program as well as promote user awareness of sign language in their life. Similar to mobile app notifications, it stops the user from forgetting about the program and leaving the program redundant. This can be achieved with a simple notification each day addressing the signing phrase of the day with a link back to the program showing the action for the phrase.
- Spell checker – the program could incorporate a real-time spell checker while the user is typing in a phrase to search. Furthermore, this could help when no results are shown with the search results and instead the program could return phrases that have a similar spelling to.
- Accurate sign language recognition – it is hard to capture 3d motion accurately within video in an efficient way. As time progresses these processes will become less of a task for the program to perform. With this heightened computing power, more efficient motion tracing can be carried out to give a more accurate recording of the user's actions.

3.2.2 Describe the solution

3.2.2a Explain and justify the structure of the solution



This diagram represents the same structure shown in 3.2.1a in a more detailed matter. From this diagram you can see the interactions between different components of the program with the inputs, outputs, processes, and databases all working in a coherent way to form the solution to the problem. The following table will explain the structure of functionality of the solution using the same colour code at in 3.2.1a.

Description of part	Justification
Searching specific phrases – The user can enter into the program a phrase which they want to be translated in to sign language. The user can input this request via text or microphone. This input is then passed through to the searching algorithm that searches through the database for a phrase of the same description. If the phrase is found it is returned for the user to view.	Users must be able to know how to sign key phrases in sign language. This feature allows the user to take a phrase (e.g. "good morning") that they think they will be able to utilise in a conversation and find how to sign that phrase. The program should then output the signing action via one or more videos of signings. This feature allows users to add a range of signing actions to their signing knowledge.
Consolidation of learning – This feature creates an array of phrases specific to each user. Once the user begins the function, the program will display the description of each phrase (e.g. "good morning"). This user will then attempt to sign the phrase in their own time and once completed they can tell the program to "show answer". This will then present a video of the signing after which the program will ask the user if they signed correctly. If the program detects that the user "struggles" with a signing it will repeatedly add that signing to the array so that it appears more frequently and subsequently allows the user to re-attempt the signing until they are more confident. Once the user is confident with a signing (are repeatedly getting the signing right) the phrase will appear less often but *will* still appear every now and then.	Once the user has been introduced to a new signing phrase it is important that they practise signing the phrase so that they are able to apply it at will in a normal interaction. The use of computation methods for this piece of functionality allows for adaptive learning by testing the user with relevant phrases.

Creating new users – Each user will be able to create an "account" in which their data from practising will be stored. This allows users to sign into the program and continue from where they left off instead of repeating exercises that have been previously completed. This also means that a progressive history can be created from each user's data in a graphical format for the user to see. User accounts with "usernames" also allow for future "peer assessment" functionality with other users that you may know. The program would be able to search for users that the user input and create "links" with that specific user so that the peer assessment functionality can be completed with someone that the user wishes to work with.	Having a framework for multiple users is important when dealing with multiple individuals with their own user data and results. Using an OOP framework allows for each user's data to be stored separately with unique identifiers such as "user number" and "username".
Editing user information – User will be able to change certain aspects of their profile to better match their true self if they did not do so originally. A common edit may be to change the level of fluency of a user to better suit their abilities. Editing user profiles may also be to change attributes such as "username", however, some attributes such as "date joined" will not be able to be changed.	As a user begins using the program, they may want to change certain aspects of their user profile. This may be originally to change aspects such as "username", etc but eventually can be used as an interface to change GUI setting such as background colour, etc.
Peer assessment – This piece of functionality will form a large part of the program. The program will ask the user whether they wish to search for a pair through a "username input" or randomly select one. Once a partner has been assigned to the user a connection will be established. Each user will be presented the description of signing action that they will sign themselves. The program will record the user signing the phrase and confirm that the user wishes to send the video. Once the user has confirmed that they wish to send the video, the video will be uploaded an online database. The partners program should then recognise the new video uploaded to the online database and extract that video and present it to the given user. Each user, having received their partner's video will "guess" the signing phrase and enter their answer into the program (this can be done using either a text box for more advanced users or a set of 4 options t o choose from). The program should then present the correct answer and correct signing and ask the user if the signing was correct.	Peer assessment works in a similar fashion to the **consolidation of learning** functionality with a random array of signing phrases generated for the users to attempt. The same generator can be used given the similar functionality. Peer assessment modules offers greater functionality as it allows the user to interact with real life people and see common mistakes and errors. It allows for users to learn from other's mistakes as well as their own. A messaging system could potentially be incorporated to allow users to communicate and advise each other.
Sign language to text – This piece of functionality should allow users to utilise a camera to record and analyse the user. The program would use motion tracking alongside pre-tracked motion captures to analyse the user's movements in order to predict the meaning of a movement. This functionality would use deep learning and **OpenCV** to analyse the movements. This functionality would only allow for basic analysis of the user's actions given that accurate representation of what was signed required expensive specialised hardware.	Sign language to text functionality can offer a lot of value to a user's learning experience. It allows the user to create a learning environment using self-checking methods if there is not access to a person-to-person experience available. Given the use of deep-learning, as more and more users utilise the functionality it will gradually get better at analysing more complex signings.

External modules:

External modules are useful to use as they utilise predeveloped code that significantly cuts down the time needed to be spent in development. They also allow for less time to be spent maintaining the program as the module is maintained by the developer. External modules increase the efficiency of the program by using optimised code that allows the process to be run at a greater rate.

The external modules that I will be using are:

- **Speech recognition** – This module will be used to allow the program to accept user input via microphone. This can be tied into the GUI to allow for the user to use this functionality at quite literally the push of a button. While this is the main use of the module, the module also has other features regarding speech recognition that may be useful in other places in the program.
- **GUIZERO** – This module will be used to create the graphical user interface. GUIZERO is a relatively simple module once downloaded and has enough complexity to handle the navigation of the program in a simple and clean way.
- **SQLite** – this module will be used to input and export data to and from the program respectively to an external database. This allows for the user's progress and details to be stored in a non-volatile manner. User data can be condensed by using a standard throughout the program (e.g. having the user's name at index 0 and the age at index 1). This allows for the data to be stored in an efficient manner but requires another module to be developed to "read" that data back into the program and make it understandable to the rest of the program. Having a relatively small database means that the data can be uploaded to an online database in the future for better security and accessibility.
- **Plotly –** this module allows me to plot the data acquired from the user's feedback to present the user's progression over time. Using this module, I will create a graph with two lines shown over time. Each point on the graph is represented as a vertex in 2d space. On the "green line" the point will represent the number of times the user indicated "I know this" and on the "red line" the point will represent the number of times the user indicated "I do not know this". Plotly is very diverse and so multiple line graphs can be produced from the module meaning one graph could have a monthly scale with another graph on a daily scale. A daily scale could also show the set of phrases practised and introduce the specific values for each term.

APIs:

- **Google cloud speech to text API** – this API will be used in tandem with the speech recognition module to send audio data and receive the text transcript of what was said. The Google API only works with audio data, but audio is at first stored as an audio file. The speech recognition module's purpose is to convert this file into data and then use the API.
- **Gspread –** the Gspread API will be used to upload the data used within the program such as the user data, etc. In order to allow for functionality such as peer-assessment, data must be uploaded to an online database so that the program in two different places can both access this database and exchange data through there.

3.2.2b Describe the parts of the solution using algorithms justifying how these algorithms form a complete solution to the problem

Searching specific phrases using user input algorithm:

Given raw data is being taken from the user, it is essential to have some sort of validation to minimise errors further along in the program. This makes sense on the basis of "GIGO" (Garbage In Garbage Out). The main function of this functionality is to output a string of words/phrases that the user wishes to search for which can be passed onto the searching algorithm functional chunk.

```
from speech_recognition import adjust_for_ambient_noise, listen,
recognise_google

function user_search(user_input,use_of_mic)
      if use_of_mic == True
            user_input = input(record user via mic)
            //this recording will be in audio file format
            user_input = adjust_for_ambient_noise(user_input)
            //filters the recording
            user_input = listen(user_input)
            //uses the speech_recognition module's functions to
            convert the audio file to audio data
            user_input = recognise_google(user_input)
            //uses Google cloud speech to text API to convert audio
            data to text
      elseif use_of_mic == False
            if user_input.isalpha()
            //isalpha function returns True if variable is only made
            of letters
                  pass
            else
                  user_input = None
            endif
      else
            user_input = None
      endif
      return user_input
endfunction
```

Back to development

Searching through database algorithm:

Once the program has the input from the user of what they wish to search for the next step would be to search through the database to find the desired representation of the phrase. The "position" variable in the function serves the purpose to identify whether the function is to search the local database or the online database. Once an online database structure has been established this functionality will be added to the pseudocode algorithm.

```
function database_search(user_input,position)
      file_path = None
      if position == "local"
            for i=0 to sign_database.length()
            //sign_database will be a list with all the available
            signings alongside their file location in a 2d list
                  if sign_database[i][0] == user_input
                        file_path = sign_database[i][1]
                  endif
            next i
            if file_path == None
                  user_input = user_input.split()
                  //removes all whitespace
                  user_input = list(user_input)
                  //splits the user's input into an array of
                  individual characters
                  file_path = [user_input.length()]
                  //creates dynamic array the length of the user input
                  for i=0 to user_input.length()
                        for j=0 to sign_database.length():
                              if user_input[i] == sign_database[j][0]
                                    file_path.append
                                    (sign_database[j][1])
                              endif
                        next j
                  next i
            endif
      endif
      return filepath
endfunction
```

This algorithm also includes the function of the program if the phrase cannot be found in the database. In this case the program splits the string into a series of characters and translates the string letter by letter. This is useful as it means the user will get some form of output instead of an error screen.

File path – This variable is used given that it makes sense to assign the relative path of the phrase against the description. This is because it makes it easier to store the database. Once the desired file paths have been obtained the program can just use the file paths to find the image when needed instead of holding the images in memory. This can cause error if the file path is incorrect or the image cannot be found, and so suitable validation should be put in place in the program before calling for an image to reduce errors.

Return to Development

Consolidation of learning algorithm:

This part of the program must first create an array of random phrases for the user to practise with and then test the use and store the results. Because these two parts can be processed individually of each other, two functions will need to be developed under the same functional chunk.

Algorithm 1 – creating random array of phrases:

```
function create_random_array_of_phrases(number_of_phrases)
      phrase_list = [number_of_phrases]
      //phrase list will have a set length but will be dynamic as
      file paths are added to it
      for i=0 to number_of_phrases
            random_num=random number from 0 to sign_database.length()
            if sign_database[random_num][1] in phrase_list
                  //doesn't allow multiple of the same phrases to be
                  added
                  i = i-1
            else
                  phrase_list.append(sign_database[random_num][1])
            endif
      next i
      return phrase_list
endfunction
```

Algorithm 2 – storing test data:

```
function storing_testing_data(testing_phrase,user_feedback,
                                    test_date,user_database)
      database_updated = False
      current_testing = [3]
      current_testing[0] = test_date
      current_testing[1] = user_feedback[0]
      current_testing[2] = user_feedback[1]
      //user_feedback is an array of length 2 with values
      [correct,incorrect] as integers
      for i=0 to user_database.length()
            if user_database[i][0] == testing_phrase
                  user_database[i].append(current_testing)
                  database_updated = True
            endif
      next i
      if database_updated == False
            new_phrase = [2]
            new_phrase[0] = testing_phrase
            new_phrase[1] = current_testing
            user_database.append(new_phrase)
      endif
      return user_database
endfunction
```

In algorithm 2 the program will check to see if the user practised (has the phrase in their database) the specific phrase. If they have not it creates a new array containing the data in the standardised format and adds the data. It then returns the new updated user_database to be stored elsewhere.

Algorithm 3 – collecting confidence rating from user

```
function confidence_rating(next_phrase,current_date)
      //next_phrase is a string of the phrase to be displayed next
      print("The next phrase to guess is: " + str(next_phrase))
      user_input = None
      while user_input!="continue"
            user_input = input("please enter 'continue' to display
                                    the answer: ").lower()
      end while
      next_phrase = next_phrase.display()
      user_result = input("if you were correct enter 'yes' or 'no'
                           if incorrect: ").lower()
      //given that this user input will be incorporated via the GUI
      buttons validation will not be needed
      phrase_index = None//this variable will store the index for
                           the phrase in the sign_database
      current_session_index = None
      for i=0 to sign_database.length()
            if sign_database[i][0] = next_phrase
                  phrase_index = i
                  for j=0 to sign_database[phrase_index].length()
                        if sign_database[phrase_index][j][0] ==
                                                      current_date
                              current_session_index = j
                        endif
                  next j
            endif
      next i
      session_list =
      sign_database[phrase_index][current_session_index]
      if user_result == "yes"
            session_list[1] = session_list[1] + 1
      elseif user_result == "no"
            session_list[1] = session_list[2] + 1
      endif
      return sign_database//returns updates sign_database
end function
```

Return to Development

Peer assessment functionality algorithm:

As described before in 3.2.2a the peer assessment functionality is complex in its nature and so will be split into different functions that will interact and exchange data with one another. The 4 algorithms have been depicted below.

Algorithm 1 – creating random array of phrases:

```
function create_random_array_of_phrases(number_of_phrases)
      phrase_list = [number_of_phrases]
      //phrase list will have a set length but will be dynamic as
      file paths are added to it
      for i=0 to number_of_phrases
            random_num=random number from 0 to sign_database.length()
            if sign_database[random_num][1] in phrase_list
                  //doesn't allow multiple of the same phrases to be
                  added
                  i = i-1
            else
                  phrase_list.append(sign_database[random_num][1])
            endif
      next i
      return phrase_list
endfunction
```

Algorithm 1 has the same function as that of **Algorithm 1 in consolidation of learning** functional chunk. While these chunks are segregated the code is identical and performs the same action

Algorithm 2 – import video from online database:

```
function import_video_online(partner_username)
      for i=0 to online_database.length()
            if online database[i][0] == partner_username
                  user_video = online_database[i][1]
                  //under the online database array would be the
                  username of the user that uploaded the video along
                  with the path to the video
            else
                  return False
            endif
      next i
      return user_video
endfunction
```

Algorithm 3 – export video to online database:

```
function export_video_online(user_video,username)
      upload_array = [2]
      upload_array[0] = username
      upload_array[1] = user_video
      online_database.append(upload_array)
      return True
endfunction
```

Algorithm 2 and 3 imports/exports the partner's video from the online database. This will use the Gspread API but the general flow of the algorithm with its steps will be similar to the final product

Algorithm 4 – Guessing the phrase and collecting user data

```
function peer_assessment_guess(user_video,ability,correct_phrase)
      user_score = 0
      partner_score = 0
      answer_options = [6]
      user_video = user_video.display
      //displays the video for the user to see
      if ability == "hard"
            user_guess = input("please enter your guess at the signed
                                    phrase: ")
      elseif ability == "easy"
            for i=0 to 4
                  answer_options.append(sign_database[random_num][0])
            next i
            answer_options.insert(random num between 0 and 6,
                                    correct_phrase)
            for i=0 to answer_options.length()
                  print("option " + str(i) + ". " + answer_options[i])
            next i
            user_guess = input("input the number for your guess at
                                    the answer")
            user_guess = answer_options[user_guess]
      endif
      for i=0 to sign_database.length()
            if sign_database[i][0] == correct_phrase
                  correct_video = sign_database[i][1]
            endif
      next i
      correct_video = correct_video.display
      //displays the video of the correct signing of the set phrase
      user_rating = int(input("rate partners attempt with 1 being
                                    wrong and 3 being perfect: "))
      if user_rating == 3
            if user_guess == correct_phrase
                  user_score = 3
            endif
      else
            user_score = 1
      endif
      return [user_score,partner_score]
endfunction
```

Algorithm 4 is a large algorithm that covers the testing part of the functional chunk. Algorithm 4 is arguably the most important given that it returns the data of the users to be stored. It obtains this data by a simple principle between the user and their partner. Once the partner has sent their video the user views the vide and must guess the phrase (the form of guess may be different, as shown in the code, depending on the ability of the user). After guessing the phrase, the user is shown the real action for the phrase. The user then rates their partner's accuracy to the real action from 1 to 3 (1 being the worst and 3 being perfect). If the user agrees that the actioning was well done, the user is judged on whether they guessed right. Guessing right scores 3 points with guessing wrong scoring 0 points. If the partner signed wrong the user gains 1 point regardless. The partner is scored based on their quality of action rating from the user. These point values are then outputted in an array.

Collecting user data:

For the creation of new users the program will begin by taking the users name and chosen username. The program will automatically set the date joined as well as the unique user_number. As shown in the pseudocode the user_number is based on the class variable user_number and increments by 1 each time a user object is created. This means that no 2 users can be created with the same user_number. This algorithm represents the skeleton structure of this functional chunk, to which more will be added in the future. The OOP paradigm used in this functional chunk allows for the addition of methods and attributes however the current code represents the essential functionality of the given functional chunk.

Algorithm 1 – creating users

```
class User
      class variable user_number = 0

      private name
            private procedure set_name(givenName)
                  name = givenName
            endprocedure
      private username
            private procedure set_username(givenUsername)
                  username = givenUsername
            endprocedure
      private date_joined
            private procedure set_date_joined(givenDateJoined)
                  date_joined = givenDateJoined
            endprocedure
      private user_number
            private procedure set_user_number(getUserNumber)
                  User.user_number = User.user_number + 1
                  user_number = User.user_number
            endprocedure
endclass
```

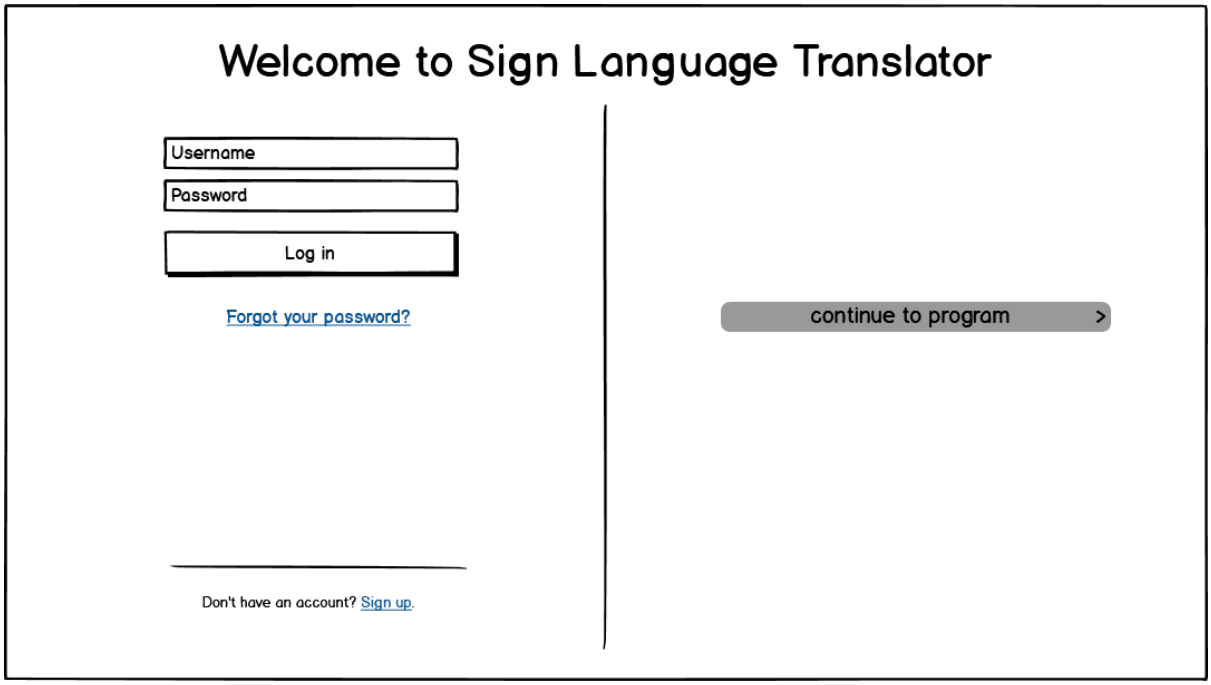
Sign language to text algorithm:

Below represents a highly extracted algorithm that represents the overall process that is carried out by the functional chunk. The true program will follow these steps with the use of mathematically calculated percentages and binary areas, etc. the general algorithm works by comparing the data from the users input to the data of every other phrase in a mathematical way. From here the program selects the phrase with the "highest similarity" and outputs the description of it.

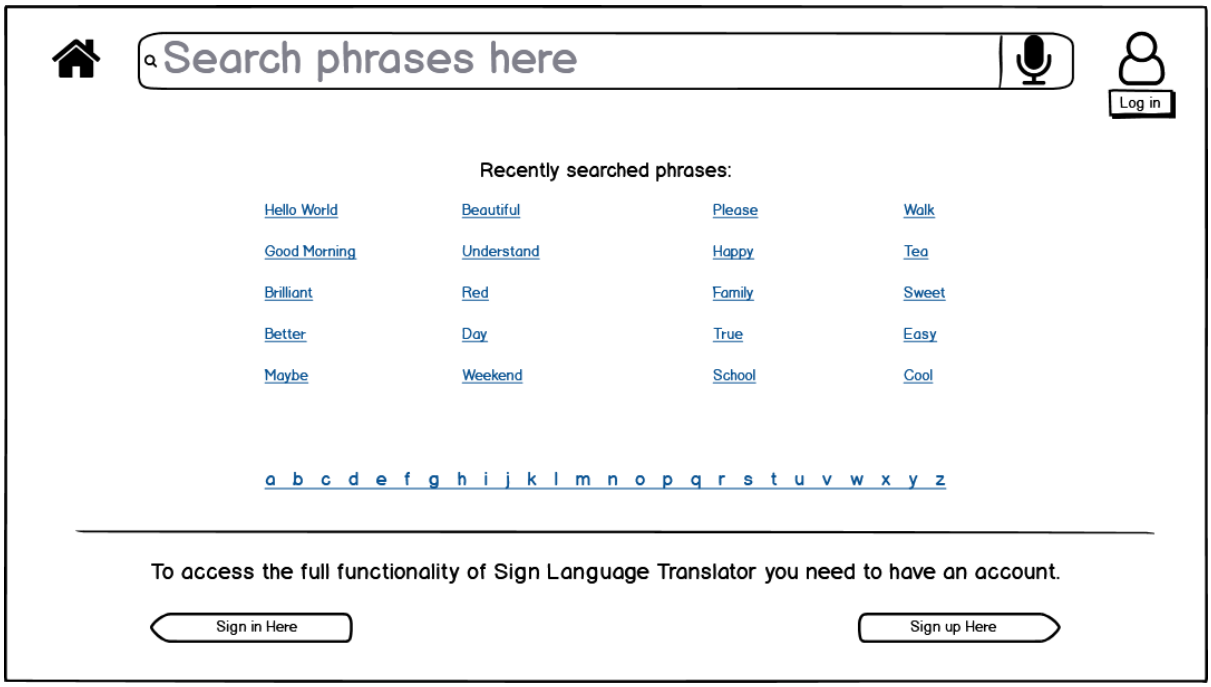
```
function sign_to_text(user_input)
      //userinput in this functional chunk represents by video data
      percentage_list = [sign_database.length()]
      for i=0 to sign_database.length()
            percentage_list.append(percentage comparison between
            user_input and sign_database[i][2])
            //sign_database[i][2] represents the motion tracking data
            of each phrase
      next i
      index = index of percentage_list largest values
      return sign_database[index][0]
      //returns the description of the phrase that the program
      predicts was signed
endfunction
```

3.2.2c Describe usability features to be included in the solution

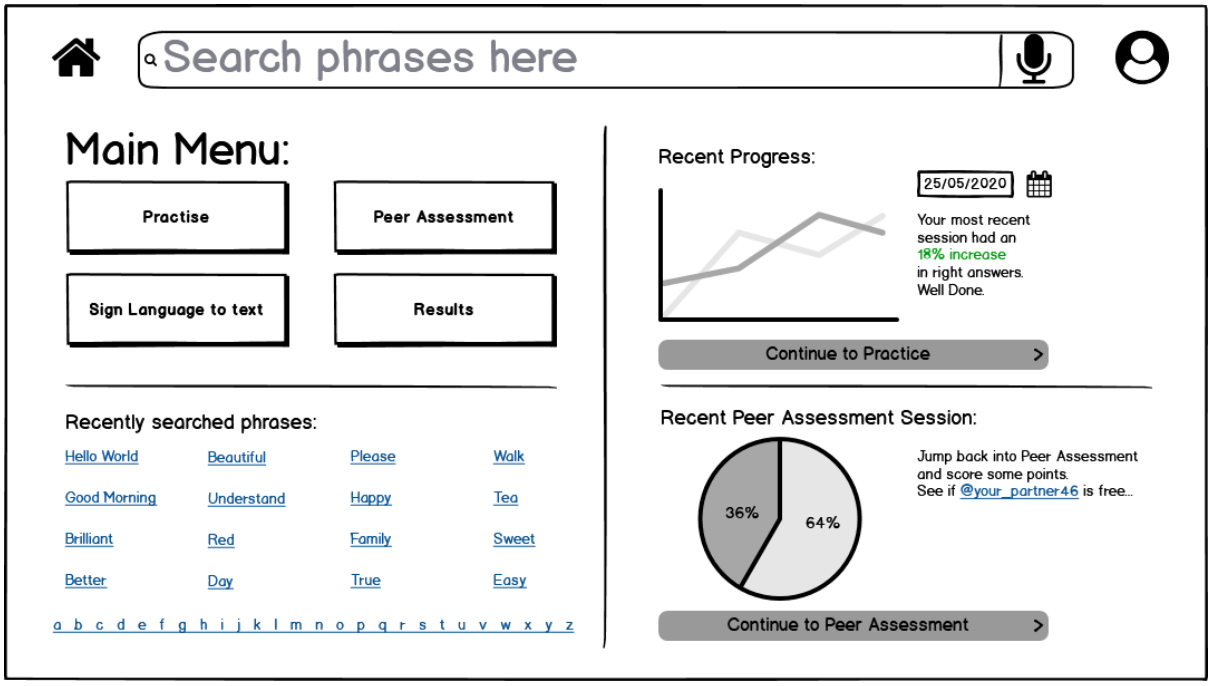
Below is a design mock-up of each of the UI that the user can expect to find in the final program. The mock-ups show the general form of the program and present the format of functionality. They will eventually be used as templates to implement the GUI with full functionality.



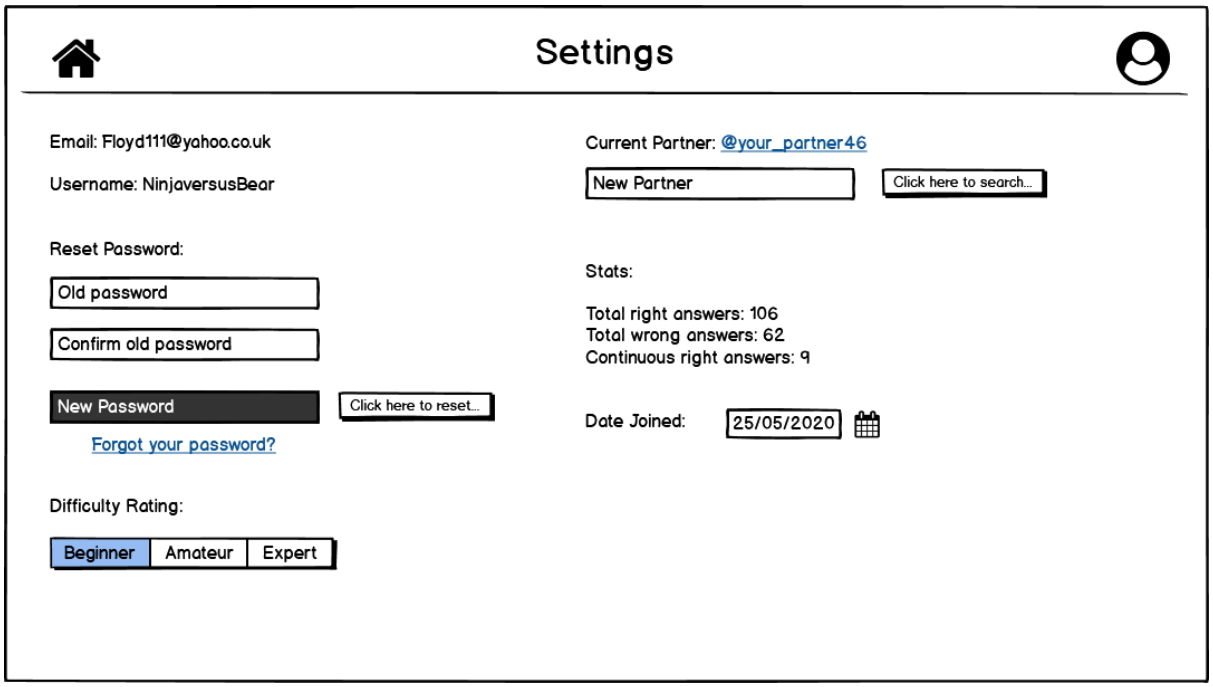
This slide shows the initial page from which the user can either log in with a username and password or use the program individually. The slide displays the further functionality available such as: “Forgot your password” and “Sign up” via links.



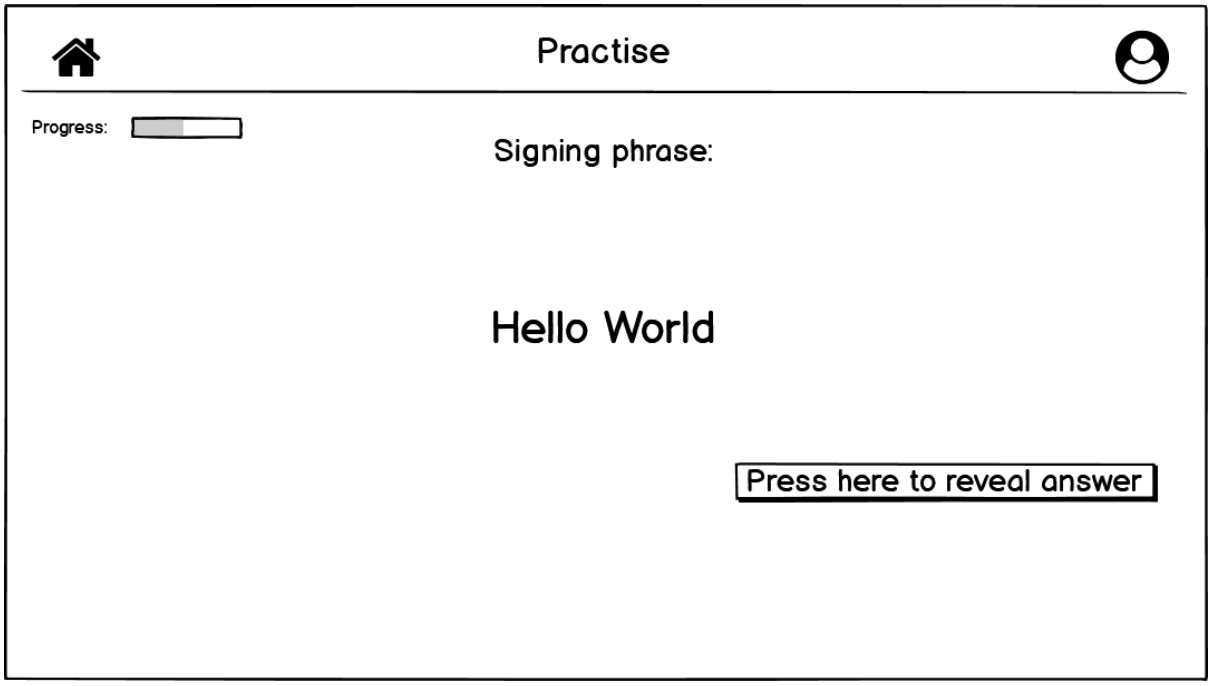
This slide shows the format of the home page that is used without being logged in. For this reason, the only functionality of the program that can be accessed is the speech/text to sign language dictionary. It also presents the 20 most recently searched phrases and the alphabet to search via first letter.



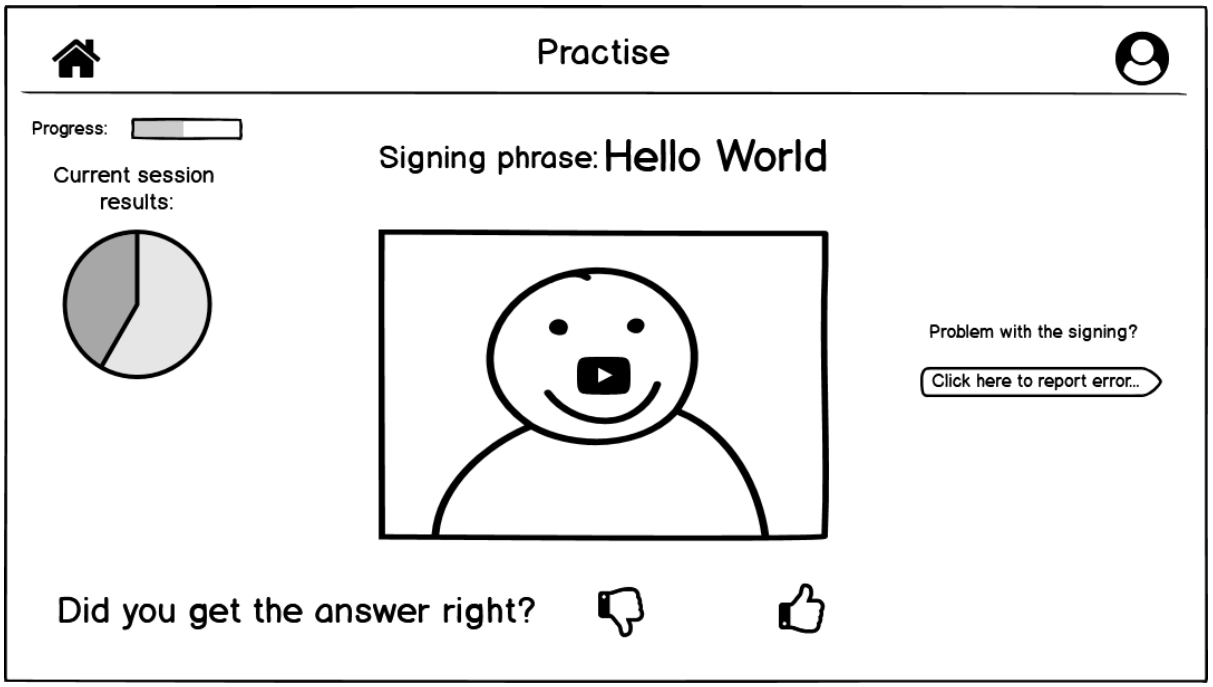
This slide shows the main menu for a logged in user. Main menu presents full functionality of the program additionally to all functionality that was present with the logged out program (previous slide). There are button links (“Practise”, etc) as well as the recently searched phrases with alphabet as seen in the previous slide. Widgets for recent practise and peer assessment sessions, displaying graphs and some information.



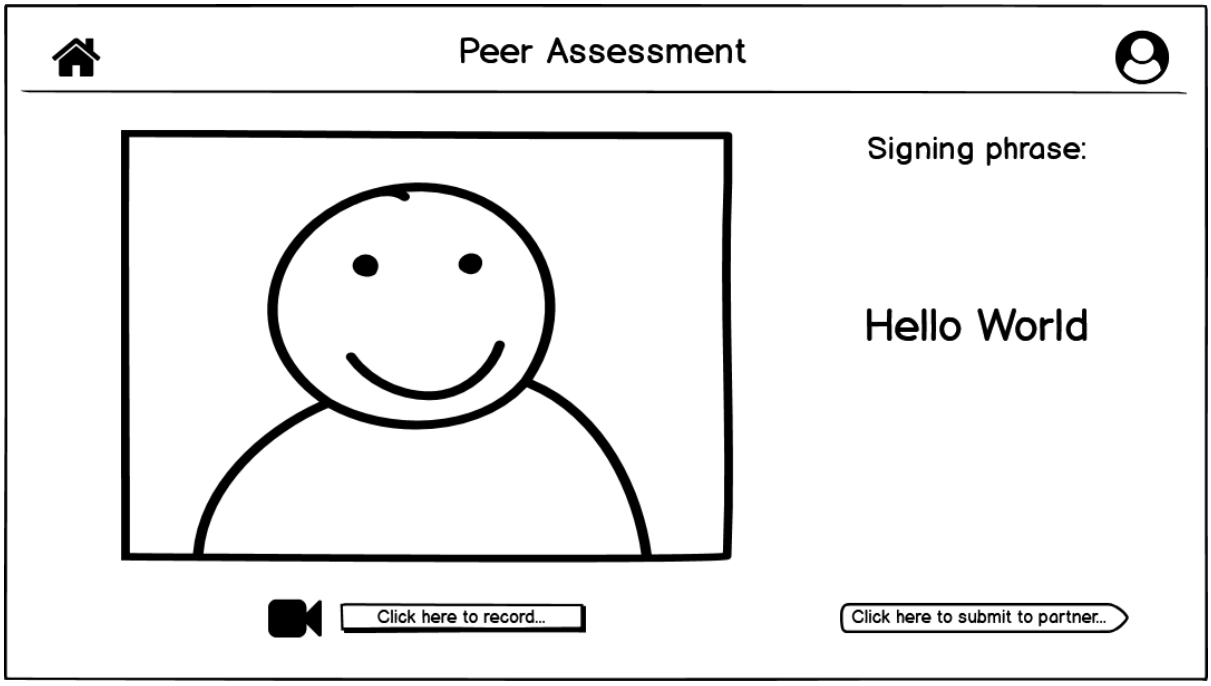
The profile settings slide allows for standard editing of user information such as changing password. It also allows the user to change their difficulty rating as they progress in their sign language. This would allow the user to automatically pair with more experienced user for more of a challenge. Some basic stats are shown as well as the option to search for a partner instead of having a predefined one.



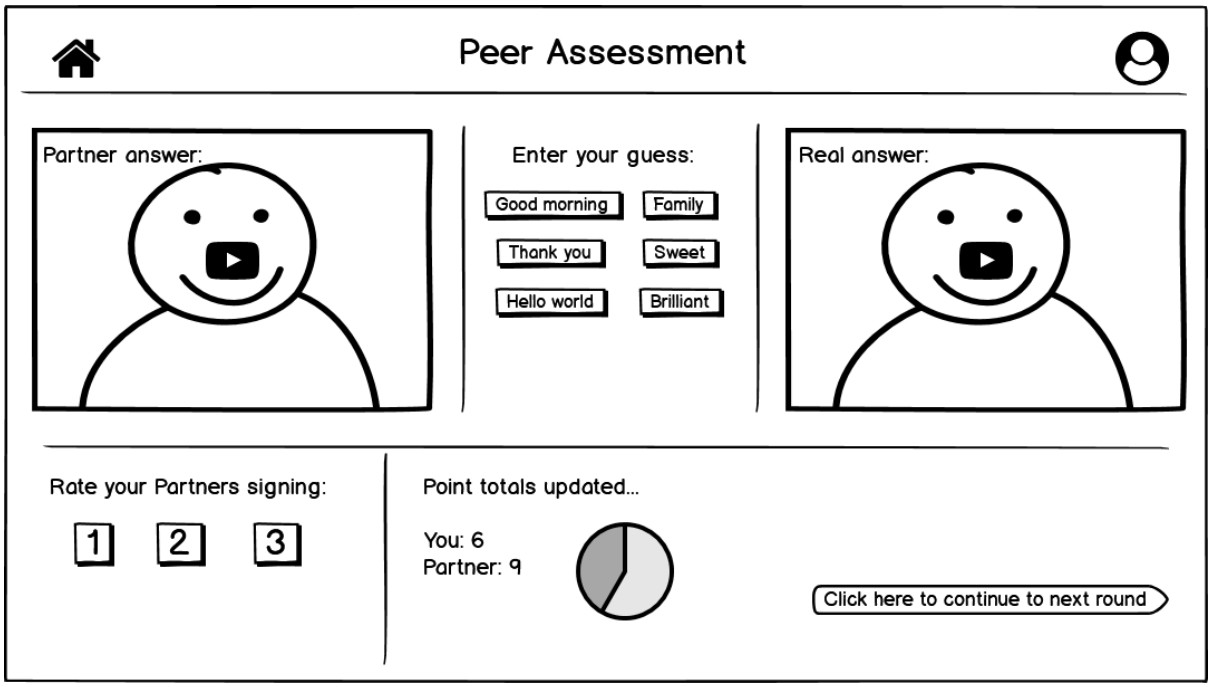
The Practise functionality begins with a minimal display for the user. This functionality utilises a self-practise, self-rate system where the user is first shown the description of the signing phrase. This phrase is derived from a predefined array of phrases with a progress bar in the top left dictating how far into the array the user has completed. This slide allows the user to practise this and attempt to sign in their own time by allowing them to press the button whenever they are ready.



The second slide for Practise functionality presents the true signing of the phrase in the form of a video that the user can play. Depending on whether the user dictates that they indeed did sign correctly/incorrectly they would then rate “yes/no” via the thumbs up/down respectively. The session results in the form of a pie chart is then adjusted. If the user doesn’t believe they actually signed wrong, they may wish to use the “report error” button to inform that the video may be wrong. Any results from a reported video should automatically be discarded.

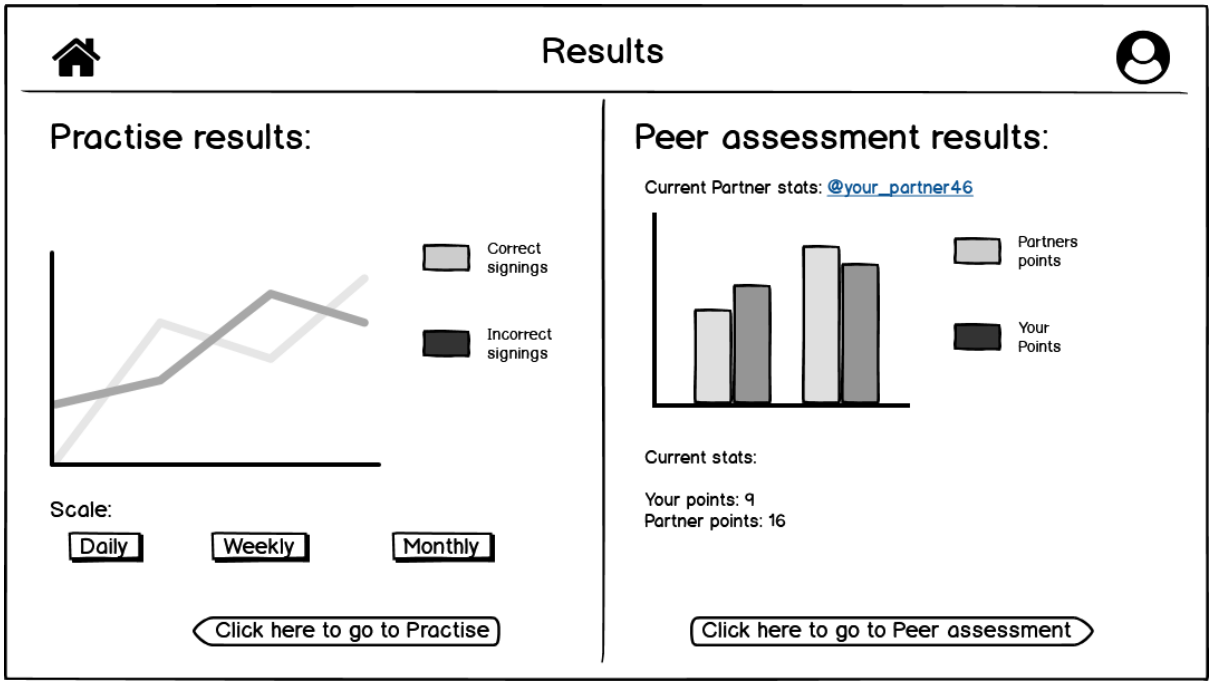


The first slide of the Peer assessment functionality presents the description of the phrase that the user is to sign for their partner to guess. It shows a window of the user's camera which the user can then use to record a video of themselves signing the phrase. Once recorded the user can click the button in the bottom right to send to their partner. If they wish to re-record they would simply stop and start the recording again.

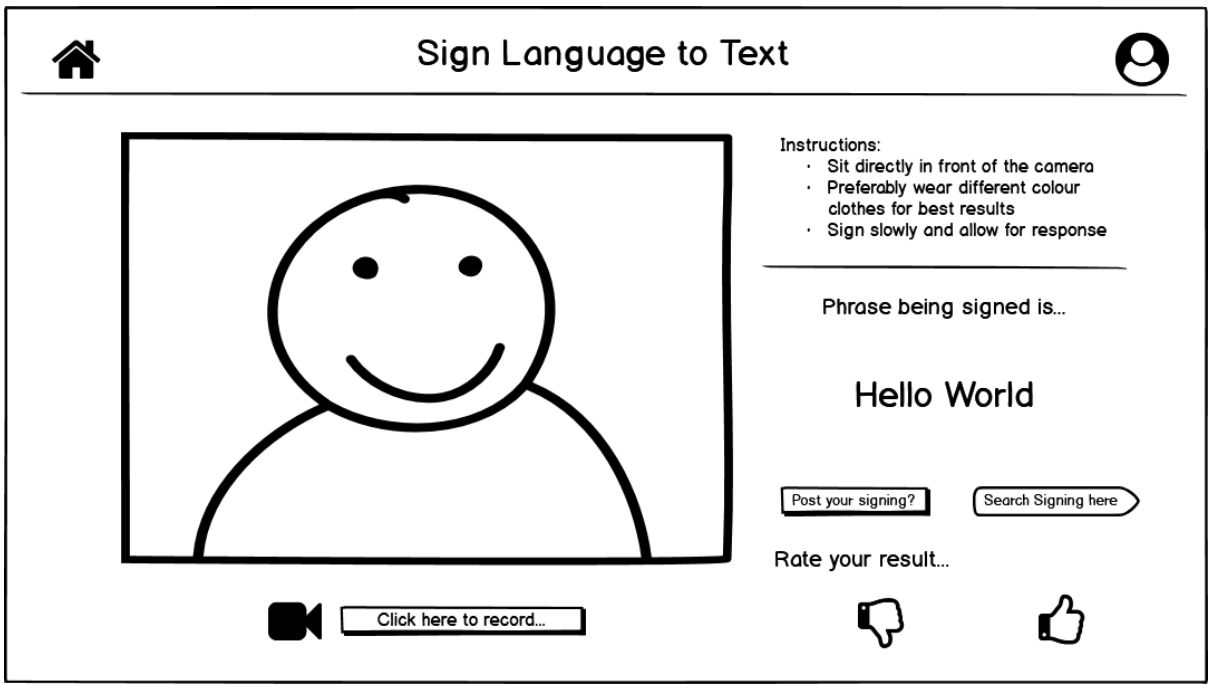


This second slide of Peer assessment functionality shows the partners perspective of the rating system. Each line in the slide shows a different part of the process:

Step 1 – watch partners video, Step 2 – enter guess, Step 3 – view real answer, Step 4 – rate partners signing. Once the partner has understood and entered all the relevant information the point total of the session is updated and a button to move on is presented. Each step is presented after each other (step 1 must be completed before step 2 will show, etc).



The results slide is purely for user interaction with the data collected from their use of the program. The left side shows the results of the user's practise with a key on the side and buttons at the bottom to change the scale of the graphical data. The right side shows the data from Peer assessment sessions and the point totals for both the user and the partner. This data is presented in this form for the user to be more understandable, however, the point totals gathered from Peer assessment are a means of keeping score while the practise results are used to judge the users average increase/decrease in rate of learning, etc.



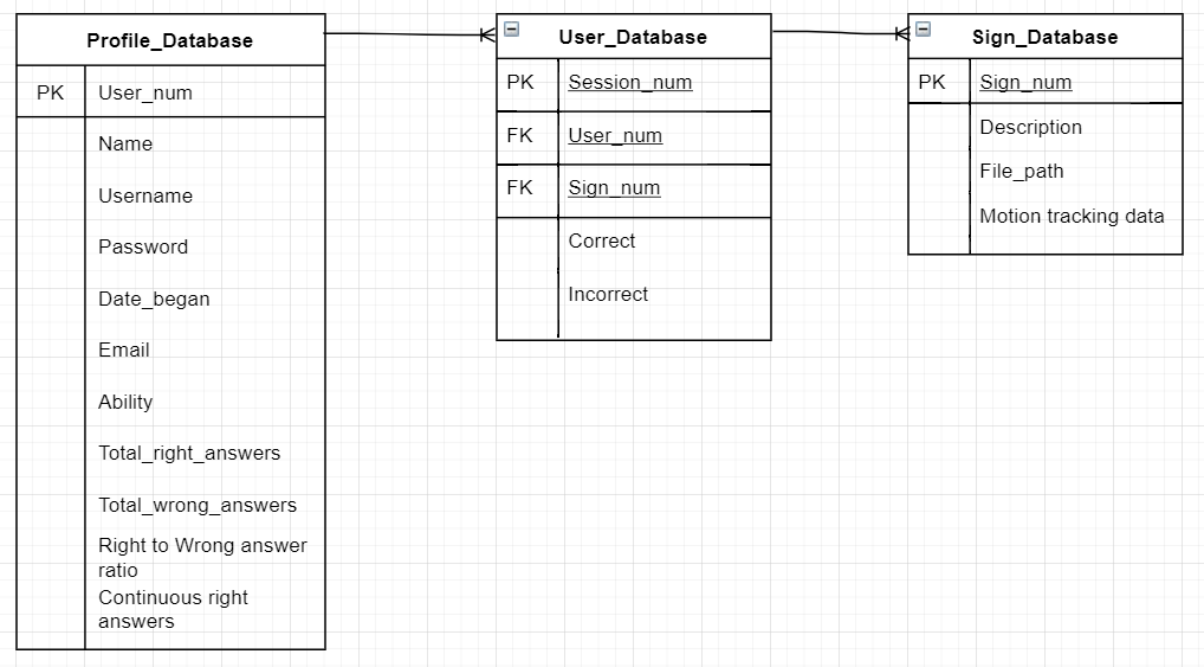
Sign language to text slide presents a camera window that can be recorded at the user's desire on the left and a live translation of the users most recent signing on the right. Further options to post signings to the dictionary as well as searching the result for other peoples attempts. Finally, a rating system is added to receive consumer satisfactions.

3.2.2d Identify key variables / data structures / classes justifying choices and any necessary validation

The following are key variable that are inputted by the users or outputted by the program.

Variable	Type	Validation	Justification
User_name – This will be a unique identifier of the user made of letters and numbers. Can be used to sign into account.	String	No special characters. Letters and numbers only. Max length 15. Has to be unique.	Username acts as a unique identifier for each user and allows for other users to search them as potential partners without mix-up. The username should not be too long that it is unwieldy for the user themselves.
Email – user's email for notification. Also can be used to sign into account.	String	Confirmation link could be sent to the users email to make sure the email is legit	Email address will be used to email updates, stats and "sign of the day" as a way to remind users to keep practising.
Password – user set password to sign into account.	String	Minimum 6 characters. Max 20 character. At least 1 capital letter, 1 number and 1 special character.	Having a length between 6 and 20 means that the password will be strong enough but not too long that the user will forget it. Non alphabet characters ensure further security.
User_num – automatically set variable that dictates the total number of users.	Integer	n/a	This variable will be used to store the user number of each user object. This will be set when the object is instantiated and be used as a unique identifier.
Profile_database – an array of all the user objects.	Array	n/a	This array will store all the user data in one place. This will be formed using a normalised method so that all the data is easily accessible and understandable. This array can be stored online so that the data can be accessed anywhere and updated automatically when any changes are made.
Sign_database– this will contain all the data regarding the description and file path for a specific signing.	Array	Only manually confirmed phrases will be added to the database.	This array can also be stored online and accessed as an online database that automatically updates as changes are made. The third index for this array of "motion tracking data" may also be added in the future.
User_database – this will contain phrases practised by the specific user and score data about each phrase segregated by date.	Array	n/a	Having a normalised database that stores all the user data regarding individual phrases makes producing graphs and diagrams much easier and accessible. This array can also be uploaded with some sort of unique identifier to allow for automatic updates when changes are made.
Partner_username – the username of the partner that was auto generated or searched for by the user.	String	Validated when the partner made it.	This is stored in a separate variable in the user object so that is attached to the user but is not confused with the user's own username.
Ability – entered when creating account and can be edited in settings.	String	n/a – use of buttons	This variable stores the user skill level and control the pace that they like to learn at as well as potential partners.

Variable	Type	Validation	Justification
Date_joined - this is defined when the user profile is created.	Object	n/a – abstracted straight from external module	An extra statistic for the user profile. The date joined allows the user to see how long they have been a member and how far they have come.
Total_right_answers – this will show the total number of right answers from practise sessions.	Integer	n/a	An extra statistic for the user profile. It allows the user to view their total progress.
Total_wrong_answers – this will show the total number of wrong answers from practise sessions.	Integer	n/a	An extra statistic for the user profile. It allows the user to view their total progress.
Continuous_right answers_record – this would store the total number of right answers achieved in a row and set a personal record.	Integer	n/a	Statistic provides further depth to the program and user experience and acts as a reference point of comparison between users.
Right_to_wrong_ratio – uses total right answers/total wrong answers to find a ratio.	Float	n/a	Statistic provides further depth to the program and user experience and acts as a reference point of comparison between users.



Above is an Entity Relationship Diagram representing the relationship between the different parts of the database. Each entity is described in the table above as arrays. This is due to the fact the data collection will occur within the program and stored in memory in array format until the program is closed.

The diagram shows that each user can have many sessions with each session containing multiple signing phrases. The use of primary and foreign keys means that the data can be stored without overlap as shown in the diagram.

3.2.3 Describe the approach to testing

3.2.3a Identify the test data to be used during the iterative development and post development phases and justify the choice of this test data

Testing is an important part of development as it tells me as the developer where the program may need to be adjusted. Filling these gaps in the program allows greater confidence that the program will respond as is intended when being used by the client.

To ensure that these gaps are initially found I will be using a range of different tests to systematically test the functionality of the program as it is developed as well as once the function has been fully envisioned. These test will test the programs response to different data inputs such as: valid, valid extreme, invalid, invalid extreme and erroneous.

Testing will be divided into sections to provide the systematic approach. Each functional chunk will be tested throughout its development as well as again once it has been completed. Once all of the functional chunks have been developed, the entire program will be tested to ensure that the program works cohesively and data flows smoothly through the program without causing errors/managing the data in unexpected ways. Testing the whole program will be done using the success criteria as a template of requirements from the program.

Searching specific phrases using user input:

Number	Test	Expected Output	Justification
FC 1.1	User should be able to enter a phrase into the search box and press enter to initiate the search algorithm. e.g. Input “Hello world”.	This should not be case sensitive. Displays the search result of what was entered as a series of visual data (photos/vids). e.g. Output search results for “hello world” with any corresponding signings.	This would be testing the programs ability to act as a sign language dictionary. Tests whether the GUI is correct for this particular output (e.g. is clear with no overlap, etc).

Number	Test	Expected Output	Justification
FC 1.2	User should be able to enter phrases via microphone using a mic button.	Program should transcribe the user’s dictation into the text box and allow them to edit it before searching.	Test whether the microphone input functionality works as expected.

Number	Test	Expected Output	Justification
FC 1.3	User can click on one of the links on the home page for “Recently searched phrases”.	Program should display the search result screen for that particular phrase.	Test whether the link provided presents the required output page for the user.

Number	Test	Expected Output	Justification
FC 1.4	User can click on one of the letters in the alphabet on the home screen.	Program should display all the phrases found in the database that begin with that letter.	Works in the same way a dictionary does. Ensures that the program correctly displays all of the appropriate phrases on the window without overlap given this page will have a larger number of outputs.

Searching through database:

Number	Test	Expected Output	Justification
FC 2.1	Enter an invalid string of letters as an unknown phrase. e.g. Input "John"	Program should not return any signing phrases except the individual letters in sign format (e.g. Will output the signing for "J", "O", "H", "N").	This allows the user to input unknown phrases such as their specific name and still get an output of some kind.

Consolidation of learning:

Number	Test	Expected Output	Justification
FC 3.1	User should be able to access consolidation of learning by pressing "Practise" button.	The program should direct the user to the "Practise" start-up page and allow the user to begin their practise session.	Tests whether the program successfully presents the start-up page for the practise functionality via the GUI.

Number	Test	Expected Output	Justification
FC 3.2	User can begin a practise session by pressing the "Start" button.	leads the user to the exercise page that displays the first phrase description for the user to sign.	Tests whether the program successfully presents the correct page and only presents the description without the actual signing. This is a fundamental concept of the function.

Number	Test	Expected Output	Justification
FC 3.3	User can escape from the practise session at any point in the practise by clicking on the home icon or profile icon.	This will direct the user to the corresponding page..	If the user accidentally starts the practise there needs to be some way to exit the session. This will most likely be the home button but as the profile icon will also be present it can also be used.

Number	Test	Expected Output	Justification
FC 3.4	User can re-enter practise session if they previously left.	Program should present the same screen that was shown at the point of the user leaving.	This would allow a user to continue with a session if they had not previously finished.

Number	Test	Expected Output	Justification
FC 3.5	Progress bar should automatically increment as the user works through the practise.	As the user continues through the practise phrases the progress bar should increase showing progress being made.	This allows the user to see how far through a session they are. If they have completed 4 phrases the bar should increment 40% if the total number of phrases is 10.

Number	Test	Expected Output	Justification
FC 3.6	User can press reveal button to view the answer.	The reveal button displays the description alongside the signing for that particular phrase.	This allows the user to utilise the adapted "look, cover, write, check" method to learn.

Number	Test	Expected Output	Justification
FC 3.7	User can report errors on a specific phrase during practise on the "Check" portion.	User should be directed to another screen where they can input their concern. Should also present a back or home button for the user to navigate back.	User feedback is necessary if the user believes there is a problem with the program or the phrase signing that they have been told. This allows the program to be developed and improved upon and allows for a better experience for other users.

Number	Test	Expected Output	Justification
FC 3.8	Users can input the results of their efforts via buttons.	Program should store this result and move onto the next phrase in the array for the user to practise. Personalised user results data presented via graphs after data has been entered to show progress.	User feedback on whether they were correct or not allows the program to adapt to the user and allow for a more tailored learning experience. This allows the user to view an up to date analysis of their progress of a session as they continue to practise.

Peer assessment functionality:

Number	Test	Expected Output	Justification
FC 4.1	User should be able to access peer assessment by pressing "Peer Assessment" button.	The program should direct the user to the "Peer Assessment" start-up page and allow the user to begin their peer assessment session. This page should display "Start" button to initialise the function.	Tests whether the program successfully presents the start-up page for the practise functionality via the GUI. The function should begin with the "Start" button so that the program can create the random array of phrases as well as set up any other sections such as the video interface.

Number	Test	Expected Output	Justification
FC 4.2	Users can press the "Start" button to start the peer assessment function.	Program should display the initial set up screen where the user is presented with a video window that displays a preview of what the camera sees. It should also present a description of the first phrase in the array for the user to sign. There should be a button to allow the user to start/stop recording. Once a recording has been made a button should be presented to the user allowing them to send the video to be sent to their partner	All of these options allow the user to create a video inside of the program of themselves signing the descripted phrase and then sending that video to their partner. This functionality should all be on one screen and be fully visible to the user to allow the greatest ease of use.

Number	Test	Expected Output	Justification
FC 4.3	User's partner can click on "view partners submission" button on their home screen.	This should direct the partner to the peer assessment module and allow them to view the video submitted by the original user.	A message or button appearing on the home screen allows the users to know that their partners have submitted something so that they can respond.

Number	Test	Expected Output	Justification
FC 4.4	User's partner should be able to view their partners submission by watching their submitted video.	Program should allow the user's partner to view the video of themselves signing the phrase.	Each user needs to be able to view their partners submission in order to "assess" them.

Number	Test	Expected Output	Justification
FC 4.5	User's partner should be able to enter their guess for the user's signing video. This can be through text box (e.g hello) or drop drown box input.	Program should accept this input and reveal the next section and display the true signing for the phrase	The partner needs to be able to guess what phrase was signed in the video sent. Only once they have done this should the program allow them to view the real video to confirm if the submission and the real video show the same signing.

Number	Test	Expected Output	Justification
FC 4.6	User's partner should be able to view the video of the real signing and compare it to the submitted signing by rating the submitted signing on how accurate it was.	Based on the rating that was given to the submitted signing the users will be scored accordingly. The program should display the current total score for the session and allow the user to continue to next phrase.	To make the peer assessment fair the partner must first guess what they think was signed. If the submitted signing was incorrect one point will be given to the guesser but none to the signer. If the signing was correct but guess was incorrect, points will be given to the signer but not to the guesser. If the signing and guess was correct points will be given to both users.

Collecting user data

Number	Test	Expected Output	Justification
FC 5.1	Users that do not have a profile should be able to sign up via sign up button.	Program should display the profile set-up page and allow the user to enter all the required details for a profile. This should include: user name, password, real name, email(optional), current ability.	In order for a user to interact fully with the program they need to sign up for an account.

Number	Test	Expected Output	Justification
FC 5.2	Users should be able to sign in with username/email and password.	With the correct details the user should be entered into the program signed in. If the details are wrong, they will be notified to retry.	Username and email will both be unique identifiers and so both will be available for signing in. The profile will only be able to be accessed with the correct password.

Number	Test	Expected Output	Justification
FC 5.3	Users can reset password by pressing “forgot my password…”.	Program should direct the user to a reset page.	It is important that the users have some way of resetting their password in case that they forget.

Number	Test	Expected Output	Justification
FC 5.4	Users can use their email to reset their password.	Program should allow user to enter the email associated with their account. If they do not have an email associated, they cannot reset. If a valid email is entered, an email will be sent allowing for the user to reset their password.	Users should not be able to reset their password unless they have the required email due to security reasons.

Number	Test	Expected Output	Justification
FC 5.5	Users can access profile setting and change ability preference.	Program should recognise the change in setting and display to the user that this setting has changed whilst simultaneously changing the attribute for the profile.	As the user progresses, they may find that they need to re-adjust their ability setting to better match their stance on sign language. This also helps the program create a more tailored learning experience.

Number	Test	Expected Output	Justification
FC 5.6	User should be able to search/request a partner to pair with. e.g.@JohnDoe	Program should indicate that a request has been sent and also display to the user on the home screen when a response has been made from partner (accept/reject)	Program should allow user to request a partner in case they wish to study with someone in particular. The user must request first to allow the other user the options of accept/reject.

Sign language to text:

Number	Test	Expected Output	Justification
FC 6.1	Users can press the "Start" button to start the sign language to text function.	Program should display the initial set up screen where the user is presented with a video window that displays a preview of what the camera sees.	This allows the user to create a video inside of the program of themselves signing the desired phrase.

Number	Test	Expected Output	Justification
FC 6.2	User can press the star/top record button when they want to start/stop recording to be used for the sign language recognition.	Start/stop button should tell the program when to record. Program will then display a preview of what it predicts the signing is that the user was signing.	For privacy reasons the program should not start analysing the video data until the user has pressed the start button. It also should not analyse until after the recording has stopped. The preview should present the phrase/ sting of phrases that were signed.

Number	Test	Expected Output	Justification
FC 6.3	After a recording has been made, the user can utilise the "Post" button.	The "post" button should direct the user to another screen to confirm the part of the recording and description of the recording, etc that is to be posted.	This allows the users to create their own videos and their own community by adding to the sign language dictionary.

Number	Test	Expected Output	Justification
FC 6.4	After a recording has been made, the user can utilise the "search other people's attempts" button.	This directs the user to the search result of that particular signing that was confirmed by the preview.	This allows the user to view other attempts if they wish to see other people technique, etc.

Number	Test	Expected Output	Justification
FC 6.5	Users can use appropriate buttons to rate the result of their signings	Program should accept this data and use it for future machine learning to adapt the translation algorithm.	Using these buttons the user can rate whether they think the program successfully or unsuccessfully translates the signing to text.

Success Criteria:

My final program will be tested along the following success criteria. They will be used to test the cohesiveness of the program and how well it can transmit data throughout the functional chunks and modules whilst still holding that data accurately.

Number	Success criteria	Test	Justification
SC 1	User will be able to search a phrase and get the result for that phrase.	Search the phrase "good morning". This should output the corresponding signing action labelled "good morning".	It is important part of the program that the user is able to search for phrases that they do not know how to sign.

Number	Success criteria	Test	Justification
SC 2	User will be able to search for a sentence and get the result for the sentence	Search the sentence: "good afternoon, good morning and goodnight". This should output the corresponding signing for the whole sentence – all of the sentence that can be found.	It is important part of the program that the user is able to search for sentence that they do not know how to sign and receive the signing for the full sentence

Number	Success criteria	Test	Justification
SC 3	Phrases that cannot be found should output an error message.	Search the string "fsdh". This should output an error message telling the user it could not be found	There may be phrases that are not in the database. On the other hand, the phrase may not be able to be found due to miss spelling, etc.

Number	Success criteria	Test	Justification
SC 4	User will be able to create a user with a username and password.	Create the following user: First name: "John" Last name: "Doe" Username: "john_doe01" Password: "Hello_world12" Fluency: "beginner" After making the user, try and log out then log back in again. Check to make sure all the details are the same.	To ensure that the user's data can be stored and kept for the next time.

Number	Success criteria	Test	Justification
SC 5	User will be able to repeatedly conduct a voice search.	Click the microphone icon to start the voice search and say "good morning" into the microphone. The search result should display the corresponding signing action for the phrase labelled "good morning"	Being able to search for a phrase via spoken English was a key piece of functionality and served as the basis of the program which was built upon. Being able to search via voice directly ties into the user's ability to directly link a signing action with the audio counterpart of that phrase. This is all in the service of further the learning of the user.

Number	Success criteria	Test	Justification
SC 6	User should be able to check their daily progress that they made.	Using the "john_doe01" user, attempt the "learning option" and randomly click "I got it right" and "I got it wrong" options for the shown phrases. Once satisfied, navigate to the "daily progress" page to view the progress that was completed. This should be an easy to read source of information in the form of a graph/chart.	It is important for users to understand their progress for new skills as well as old skills. Showing progress not only reinforces the user's achievements but also shows them where they may need improvement. Furthermore, a progress map also encourages users to continue using the resource in order to keep a daily progress and keep improving.

Number	Success criteria	Test	Justification
SC 7	Users will be able to change the time frame of the progress data that they wish to see.	Again, using the "john_doe01" user, navigate to the progress page. Switch between the time frame settings from "daily" to "weekly" and "monthly". There should not be any data here yet, but some data should be shown from the progress made beforehand.	Adding on to SC 8 the user also needs to be able to change the time frame allowing them to see a greater view of their progress. This is important in all users but more useful to users that use the program more over a longer period of time.

Number	Success criteria	Test	Justification
SC 8	Users should not be able to create two accounts with the same username, but the program should still accept users with the same first and last name.	Attempt to make the exact same user that was made SC 4 again. The program should output that the username is already taken. With all the details the same, attempt to remake the SC 4 user with the username "john_doe02" instead. The program should allow a brand-new user to be created.	Two users may try and create an account with the same username. This may cause confusion or even errors in the program and so should be prohibited. However, the program should still accept users with the same first and last name by using the username as the unique attribute.

Client System testing

After the white box testing (testing done by the developer) has been completed I will move onto black box testing (testing done by the client). This will allow a different perspective to navigate through the program and determine if the information is easily understandable. Given that this program is not specialised it needs to be understandable to a wide range of people and therefore easy to use for beginners as well as experienced people. The average sign language student will not know the inner working of the program and will only be able to interact in the clearly labelled sections. Using the feedback gained from this testing I will be able to further improve the program for a more intuitive interface.

3.3 Developing the solution

3.3.1 Iterative development process

3.3.1a Provide annotated evidence of each stage of the iterative development process justifying any decision made

Due to the size of this project, the process of decomposition is imperative. Breaking down the project into functional chunks was the first step in this process and will be followed by further decomposition by developing each component of the chunk, algorithm by algorithm. Working on one algorithm at a time allows me to make sure the process is fully working before moving onto the next process. Doing this efficiently will require me to initially use “dummy data” until enough of the program’s foundation is developed to supply data through the system and analyse the outputs. Developing the foundations of the program after the processing sections makes sense due to this allowing me to enter tailored data into the processing section and monitor the outputs instead of relying on the pre-made data and only analysing outputs. Not relying on other pieces of code reduces the risk of buried errors and further justifies the need for decomposition.

Development Contents Page

Functional Chunk Development Sections:

Functional chunk 1 – Searching specific phrases

Functional chunk 2 – Searching through database

Functional chunk 3 – Consolidation of learning

Functional chunk 1 – Searching specific phrases

Related sections:

- Algorithms
- Prototypes
- Code
- Tests
- Remedial actions
- Code listing

This functional chunk will focus on developing main aspects of searching phrases such as:

- Allowing the user to input a phrase to be searched
- Conditioning the string inputted by the user into the search engine
- Creating a “top 20” most recently searched phrases section
- Creating an alphabet line where the user can click on an individual letter and search via beginning letter
- Allowing for the user to use the microphone with speech recognition to search for phrases

Together these pieces of code will provide the user with the functionality to search for specific phrases, making up the main aspects of the home page front end.

Back to contents page

Inputting a phrase Part 1 – Iteration 1:

```
from guizero import *

def store_data():
    output_text.value = "your input was: " + str(search_engine.value)

app = App(title="Searching dev",width=1000,height=700,layout="grid",bg=(99,207,237))

instructions = Text(app,text="Please enter a phrase to be searched:",grid=[1,1],color="white")
search_engine = TextBox(app,width=50,grid=[1,3])
enter_button = PushButton(app,command=store_data,text="Enter",grid=[2,3])
output_text = Text(app,text="",grid=[1,5])

app.display()
```

Using GUIZERO for the graphical user interface:

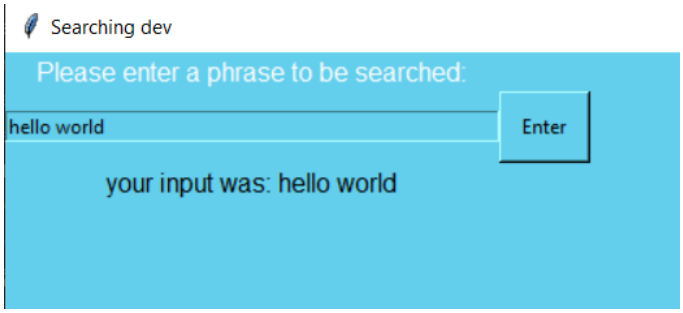
- Import - This code shows the use of the GUIZERO module
- Store data – this function stores the data inputted by the user to be used later – this is demonstrated by outputting it to the user in the GUI
- App - An application box is created with a grid layout – this allows better formatting further on.
- Text - The module is used to create a text object that details the instructions for the user
- TextBox - Another object is made using the TextBox function – this is where the user will enter their input
- Button – an enter button is used for the user to submit their entry. This button calls the function that collects and then utilises the data

Description/Justification:

To allow the user to input data into the system some sort of interface need to be developed. In this case the use of a Graphical User Interface is essential as it not only provides an easy method of inputting data into the program but allows the program to output a variety of types of data such as pictures/videos more easily.

Testing for Inputting a Phrase – Part 1 - Iteration 1

Output:



Input Conditioning – Iteration 1:

```
def search_conditioning(user_input):
    """function gets rid of any special characters or numbers but keeps spaces.
    returns the conditioned string"""
    new_string = ""
    for i in range(len(user_input)):
        #loops through the string and checks if every character is a letter
        if user_input[i] == " ":#keeps the spaces
            new_string += user_input[i]
        elif user_input[i].isalpha():
            new_string += user_input[i].lower()
            #only adds the letters to the final string
    return new_string

my_string = "h3ello w0orld!"
new_string = search_conditioning(my_string)
print("The new conditioned string is: " + str(new_string))
```

Description/Justification:

Search conditioning is made into a function to allow it to be used throughout the program for multiple purposes. The function inputs a string and adjusts the string by removing any numbers and special characters. The use of this function is in anticipation of mistyping by the user. The act of removing the special characters and numbers acts as low-level validation and is used because it does not remove anything that could possibly be part of the user's desired input while removing characters that could possibly cause future errors.

Output:

```
The new conditioned string is: hello world
```

Testing for Input Conditioning – Iteration 1

Top 20 Recently searched phrases – Part 1

```
def top_20_adjust(top_20_list,new_value):
    """function adds the new value to the front of the list.
    if the list is bigger than 20 value is removes one from the back
    and adds the new one to the front"""
    if len(top_20_list) == 20:
        top_20_list.pop(-1)#removes value from the back of the list
        top_20_list.insert(0,new_value)#adds to front
    else:
        top_20_list.insert(0,new_value)#adds to front
    top_20_format(top_20_list)
```

Description/Justification:

This function makes up the first step of the process to allowing the program to store the 20 most recent phrases searched by editing the current list so that it always only has 20 values in the specific format. This function allows the program to progressively maintain the top 20 list and keep it in a normalised form. Keeping the list 20 values is essential due to the graphical implications of a longer list which could cause the outputted text to infringe on other objects in the GUI.

Testing for Top 20 Recently Searched Phrases – Part 1

Output:

```
['hello', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i',
'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's']
```

This output screen shot shows a list being printed after being processed by the above function. The list of 20 values was passed through the function as the “top_20_list” alongside a string to be added, “hello”. The output showcases the program successfully processing the list without error while maintaining the length of the list at 20.

Top 20 Recently searched phrases – Part 2 – Iteration 1

```
def top_20_format(my_list):
    """function formats the lists so that they can be more easily displayed.
    creates a 2d list with up to 4 values in each of the inner lists."""
    list_1 = []
    list_2 = []
    for i in range(len(my_list)):#loops through all the current top 20 values
        if len(list_1) == 4:#if there are 4 values in the holder list
            list_2.append([])#creates a new empty list
            for i in range(4):
                list_2[-1].append(list_1[i])#adds the 4 values to the empty list
            del list_1[:]#clears the holder list

        elif (i+1) == len(my_list):#if at the last index in list
            list_1.append(my_list[i])#adds the last value to holder list
            list_2.append([])#creates space for new values
            for i in range(len(list_1)):
                list_2[-1].append(list_1[i])#adds the rest of values to the empty list
        else:
            list_1.append(my_list[i])#adds value to holer list
    top_20_output(list_2)
```

Description/Justification:

Part 2 of the Top 20 development continues from Part 1 in processing the data to obtain the desired format. In this program the list of values for the top 20 phrases is broken down into groups of 4 by using a 2d list. Each of the lists containing 4 values represent the 4 columns that will be made by the lists. This continues the process of normalising the list to maintain the desired functionality while keeping the larger algorithmic functions separate for them to be used in multiple ways (E.G. used by alphabet line)

Testing for Top 20 Recently Searched Phrases – Part 2 – Iteration 1

Output:

No output due to error. Please see testing and remedial actions.

Top 20 Recently searched phrases – Part 2 – Iteration 2

```
def top_20_format(my_list):
    """function formats the lists so that they can be more easily displayed.
    creates a 2d list with up to 4 values in each of the inner lists."""
    list_1 = []
    for i in range(len(my_list)):
        if i %4 == 0:#if the index of the list is a multiple of 4 - this includes 0
            list_1.append([])#creates new inner list
            list_1[-1].append(my_list[i])#adds value to the back-most inner list
        else:
            list_1[-1].append(my_list[i])#adds value to the back-most inner list
    top_20_output(list_1)
```

Description/Justification:

Corrected and updated code from Iteration 1.

Testing for Top 20 Recently Searched Phrases – Part 2 – Iteration 2

Output:

```
[['hello']]
[['this', 'hello']]
[['is', 'this', 'hello']]
[['a', 'is', 'this', 'hello']]
[['test', 'a', 'is', 'this'], ['hello']]
```

This output shows the 2d list formed as string values are added to the top 20 list. It successfully shows the production of a 2d list with each inner list forming a maximum length of 4 before starting a new inner list. It also shows no issues for the program when the number of values is not a multiple of 4 by adding the “spare” values to a separate list. The output showcases the maintenance of the order of the list as the reverse chronologically inputted.

Top 20 Recently searched phrases – Part 3 – Iteration 1

```
def top_20_output(my_list):
    """function uses the formatted lists to display the values in 4 colums with up to 20 values"""
    output_string = ""
    short_list = []
    iteration = 5
    if len(my_list)<iteration:#output should have 5 rows or less
        iteration = len(my_list)
    for i in range(iteration):
        short_list.append(my_list[i])#takes the first 5 "rows" of data
    col_width = max(len(word) for row in short_list for word in row)+2
    # calculates the necessary spacing between each columns for neat formations
    counter = 0
    for row in short_list:
        if counter<5:#second level of validation to make sure no more than 5 rows
            output_string += ("".join(word.ljust(col_width) for word in row) + "\n")
            #creates a string with all values in ordered fashion
        counter+=1
    testing_output.value = output_string
```

Description/Justification:

Part 3 uses the formatted arrays to present the output of the Top 20 phrases to the user. Using the normalised 2d lists for the foundation it allows for a maximum of 5 rows. This means no more than 20 values will ever be shown. This extra condition in the code is put in place since other lists with less normalised values such as “the alphabet line” can still be used. It also means any errors in the code have an extra layer of validation.

The function displays the 4 columns by using a string and adding each value to the string. The distance between the columns is dependent on the longest string and so is tailored to the strings themselves meaning the will be equally spaced out. This string is then outputted to the GUI.

Testing for Top 20 Recently Searched Phrases – Part 3 – Iteration 1

Output:

```
test    string  long    a
is      this    world   hello
```

This output shows the use of the function to output the text to the user in the desired fashion (4 columns and 5 rows). The output also shows the use of the function when there are less than 20 values and how it still creates the 4 rows due to the split earlier – creating the 2d list.

Alphabet Line – Part 1 – Iteration 1

```
alphabet = Combo(app,options=["a","b","c","d","e","f","g","h","i","j","k","l",
"m","n","o","p","q","r","s","t","u","v","w","x","y","z"],command=alphabet_display,grid=[9,1])
```

This combo object presents a dropdown box for the user to select a letter of the alphabet.

```
with open("words_dictionary.json") as f:
    database = json.load(f)
f.close()
```

Creating database dictionary from local .json file.

```
def alphabet_display(letter):#beginning letter passed through
    """function takes a letter and creates a list of all the phrases beginning
    with that letter from the database"""
    letter_list = []
    for phrase in database:
        if phrase[0] == letter:#if beginning letters match
            letter_list.append(phrase)#adds it to the list
    top_20_format(letter_list)#formats the list for a top 20
```

Description/Justification:

Once the desired letter is selected, the program can identify which values need to be returned. This function shows the process of looping through the database to find values with the corresponding first letter and adding them to a list. The database is a dictionary created at the start of the program from a local .json file. This file contains an abundance of phrases that may be entered by the user. Anything not in this database will not be classed as a real word.

The alphabet line acts as an extra feature in the program and allows the user one more method of interaction with the program. This list is then used and formatted by the "top_20_format" function. This is the same function used in the "Top 20 most recent searches" algorithms as it uses the same concept by only returning the first 20 values with the corresponding first letter. This allows the results to be presented in a aesthetically pleasing way.

Testing for Alphabet Line Part 1 – Iteration 1

Output:

```
e           ea          eably       eaceworm
each        eachwhere   ead         eadi
eadios      eadish      eager       eagerer
eagerest    eagerly     eagerness   eagers
eagle       eagled      eaglehawk   eaglelike
```

This output shows the phrases outputted by the program when a letter is selected. In this case the letter selected, "e", causes the program to search for all values beginning with "e" and creates a large list with each of the values. With a large word bank, a lot of values are returned however the use of this function makes it so that only the first 20 values are outputted.

Note: a button could be developed to cycle through the phrases 20 values at a time.

Alphabet Line – Part 2 – Iteration 1

```
def alphabet_iteration():
    """function allows the user to iterate through the values from the letter_list
    and shows the next 20 values"""
    if len(letter_list) == 0:
        #if there are no values in the letter_list
        #function will do nothing
        return
    else:
        current_letter = letter_list[0][0]
        if len(letter_list)<=20:#if no further iteration can happen
            alphabet_display(current_letter)#redo the list
        else:
            output_string = top_20_format(letter_list)#formats the list for a top 20
            alphabet_output.value = output_string#outputs to GUI
        return letter_list
```

Description/Justification:

The function shows the processing code for the button that allows the user to iterate through the letter list to "refresh" the values. Is does this by deleting the first 20 values from the list and reformatting the next 20 values to be displayed. This allows the user to sift through the vast number of values that are outputted to the user. Note: this will most likely be done on a separate window in the final product.

Testing for Alphabet Line – Part 2 – Iteration 1

Output:

No output will be shown due to incorrect output. See Testing and Remedial Actions.

Alphabet Line – Part 2 – Iteration 2

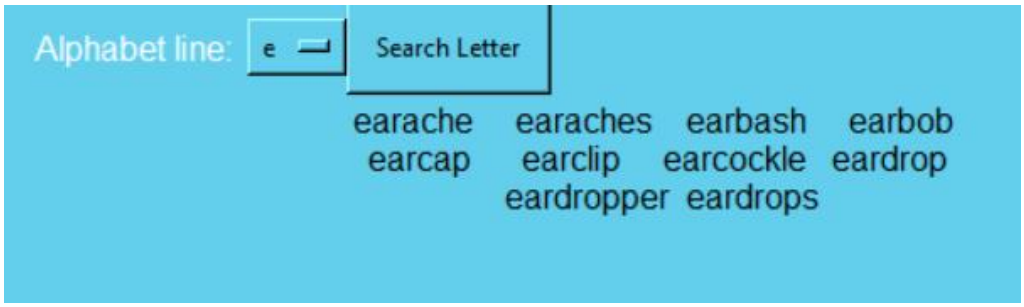
```
def alphabet_display(letter):#beginning letter passed through
    """function takes a letter and creates a list of all the phrases beginning
    with that letter from the database"""
    del letter_list[:]#list is cleared before initiation so values are not added to a full list
    for phrase in database:
        if phrase[0] == letter:#if beginning letters match
            letter_list.append(phrase)#adds it to the list
    output_string = top_20_format(letter_list)#formats the list for a top 20
    alphabet_output.value = output_string
    return letter_list

def alphabet_iteration():
    """function allows the user to iterate through the values from the letter_list
    and shows the next 20 values"""
    if len(letter_list) == 0:
        #if there are no values in the letter_list
        #function will do nothing
        return
    else:
        current_letter = alphabet.value
        if len(letter_list)<=20:#if no further iteration can happen
            alphabet_display(current_letter)#redo the list
        else:
            del letter_list[:20]
            output_string = top_20_format(letter_list)#formats the list for a top 20
            alphabet_output.value = output_string#outputs to GUI
        return letter_list
```

Description/Justification:

This code shows a combination of the code from Part 1 and Part 2 due to the remedial actions in Part 1 allowing for Part 2 to work correctly.

Testing for Alphabet Line – Part 2 – Iteration 2

Output:



Alphabet Line – Part 2 – Iteration 3

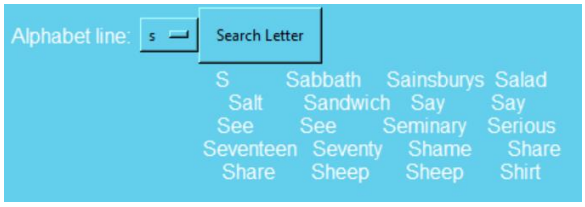
```
def alphabet_display(letter):#beginning letter passed through
    """function takes a letter and creates a list of all the phrases beginning
    with that letter from the database"""
    del letter_list[:]#list is cleared before initiation so values are not added to a full list
    counter = 0
    for word in total_vids:
        if word["Phrase"][0].lower() == letter:#if beginning letters match
            letter_list.append(word["Phrase"])#adds it to the list
    output_string = top_20_format(letter_list)#formats the list for a top 20
    alphabet_output.value = output_string
    return letter_list

def alphabet_iteration():
    """function allows the user to iterate through the values from the letter_list
    and shows the next 20 values"""
    if len(letter_list) == 0:
        #if there are no values in the letter_list
        #function will do nothing
        return
    else:
        current_letter = alphabet.value
        if len(letter_list)<=20:#if no further iteration can happen
            alphabet_display(current_letter)#redo the list
        else:
            del letter_list[:20]
            output_string = top_20_format(letter_list)#formats the list for a top 20
            alphabet_output.value = output_string#outputs to GUI
        return letter_list
```

Description/Justification:

This iteration uses the phrases in the online database with corresponding signings. This is used instead of the word database as it shows the user what can be searched in the actual database and output a dedicated signing phrase. The means that the word database will only be used for input validation.

Testing for Alphabet Line – Part 2 – Iteration 3

Output:



This shows the program correctly iterating through the array of phrases correctly for a given letter (e.g. “s”)

Microphone Input – Iteration 1

```
def mic_input():
    """function called when mic button is pressed.
    allows the user to use the microphone for audio input to text
    uses local function to insert result into text box"""
    microphone_input = ""
    r = sr.Recognizer()
    with sr.Microphone() as source:#uses local module as a microphone inputter
        r.adjust_for_ambient_noise(source)
        audio = r.listen(source)#converts the input from the microphone to audio data object
    try:
        #performs a speech recognition on an audio data instance using Google Speech Recognition API
        microphone_input = ('%s' % r.recognize_google(audio))
    except:
        #excepts unknown input that cannot be translated or any other errors such as no network connection
        microphone_input = ("sorry, unable to translate audio")
    if 0: # plot and/or play back captured audio
        s = audiomath.Sound(audio.get_wav_data(), fs=audio.sample_rate, nChannels=1)
        s.Play()
        s.Plot()
    insert_text_box(search_engine,microphone_input)#local function inserts mic result into text box
```

```
def insert_text_box(text_box,text):
    """function used to replace tech in text box"""
    #function used in mic input function
    text_box.value = text
```

Description/Justification:

This part of the program allows the user to use the microphone as a source of input for searching. Using the speech_recognition module for python the program can translate the audio data into text. It does this by using the Google Speech Recognition API. The program then also uses a function (insert_text_box) to output the text to the textbox to be searched or return an error using try and except. The use of microphone as an input is justified as it acts as a defining feature of the program and allows the user to interact with the program in a whole new dimension instead of always typing into text boxes. The use of this function can also be implemented at other points of the program.

Testing for Microphone Input – Iteration 1

Output:

No output will be shown due to incorrect output. See Testing and Remedial Actions.

Microphone Input – Iteration 2

```
import audiomath; audiomath.RequireAudiomathVersion( '1.12.0' )
import speech_recognition

class DuckTypedMicrophone( speech_recognition.AudioSource ):
    # descent from AudioSource is required purely to pass an assertion in Recognizer.listen()
    def __init__( self, device=None, chunkSeconds=1024/44100.0 ):
        # 1024 samples at 44100 Hz is about 23 ms
        self.recorder = None
        self.device = device
        self.chunkSeconds = chunkSeconds
    def __enter__( self ):
        self.nSamplesRead = 0
        self.recorder = audiomath.Recorder( audiomath.Sound( 5, nChannels=1 ), loop=True, device=self.device )
        # Attributes required by Recognizer.listen():
        self.CHUNK = audiomath.SecondsToSamples( self.chunkSeconds, self.recorder.fs, int )
        self.SAMPLE_RATE = int( self.recorder.fs )
        self.SAMPLE_WIDTH = self.recorder.sound.nbytes
        return self
    def __exit__( self, *blx ):
        self.recorder.Stop()
        self.recorder = None
    def read( self, nSamples ):
        sampleArray = self.recorder.ReadSamples( self.nSamplesRead, nSamples )
        self.nSamplesRead += nSamples
        return self.recorder.sound.dat2str( sampleArray )
    @property
    def stream( self ):
        # attribute must be present to pass an assertion in Recognizer.listen(),
        #and its value must have a .read() method
        return self if self.recorder else None

# source must be an instance of a speech_recognition.AudioSource subclass
# source.stream must be non-None while the source is active
# source.CHUNK must be the (integer) number of samples per chunk
# source.SAMPLE_RATE must be the sampling rate
# source.SAMPLE_WIDTH must be the number of bytes per sample
# source.stream.read(numberOfSamples) must return raw single-channel audio data
```

```
def mic_input():
    """function called when mic button is pressed.
    allows the user to use the microphone for audio input to text
    uses local function to insert result into text box"""
    microphone_input = ""
    r = sr.Recognizer()
    with DuckTypedMicrophone() as source:#uses local module as a microphone inputter
        r.adjust_for_ambient_noise(source)
        audio = r.listen(source)#converts the input from the microphone to audio data object
    try:
        #performs a speech recognition on an audio data instance using Google Speech Recognition API
        microphone_input = ('%s' % r.recognize_google(audio))
    except:
        #excepts unknown input that cannot be translated or any other errors such as no network connection
        microphone_input = ("sorry, unable to translate audio")
    if 0: # plot and/or play back captured audio
        s = audiomath.Sound(audio.get_wav_data(), fs=audio.sample_rate, nChannels=1)
        s.Play()
        s.Plot()
    insert_text_box(search_engine,microphone_input)#local function inserts mic result into text box
```

Description/Justification:

The updated iteration of the microphone input includes the internal module coded in replacement for the non-functional Microphone method. All other code is identical to Iteration 1.

Testing for Microphone Input – Iteration 2

Output:

Outputting a Phrase – Part 1 – Iteration 1

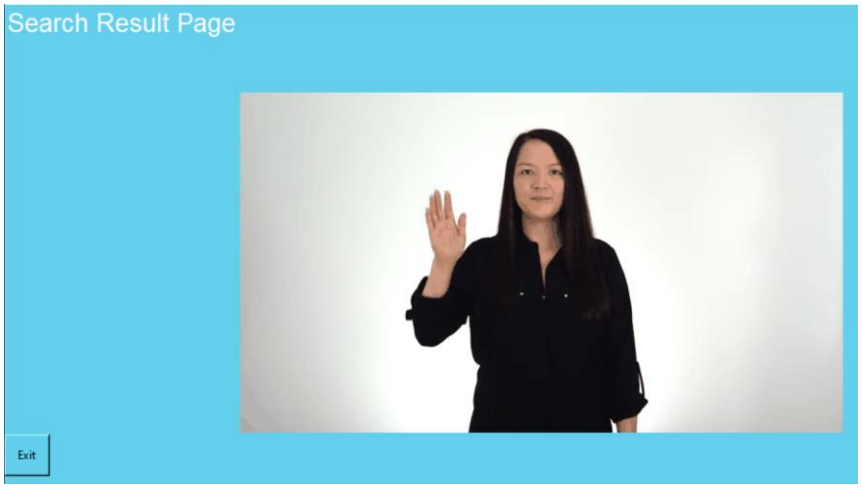
```
def search_to_search_result():
    #switches pages
    search_page.hide()
    search_result_page.show()
    signing_page_button.visible = False
    #obtains the user's validated input
    corrected_input = ""
    for i in range(len(output_text.value)):
        if output_text.value[i] == ":":
            corrected_input = output_text.value[i+2:]
            break
    #outputs the signing
    search_result_gif_output.image = corrected_input + ".gif"
```

Description/Justification:

This piece of code shows the program outputting the signing phrase to the user. It does this on a separate page by using different Box objects for the different pages. The program obtains the user's validated input from the main page and uses it to obtain the local file path of the .gif to output.

Testing for Inputting a Phrase – Part 2 – Iteration 1

Output:



This screenshot shows the result on a new GUI page by using a Box object to separate the search screen and the output screen.

Outputting a Phrase – Part 2 – Iteration 1

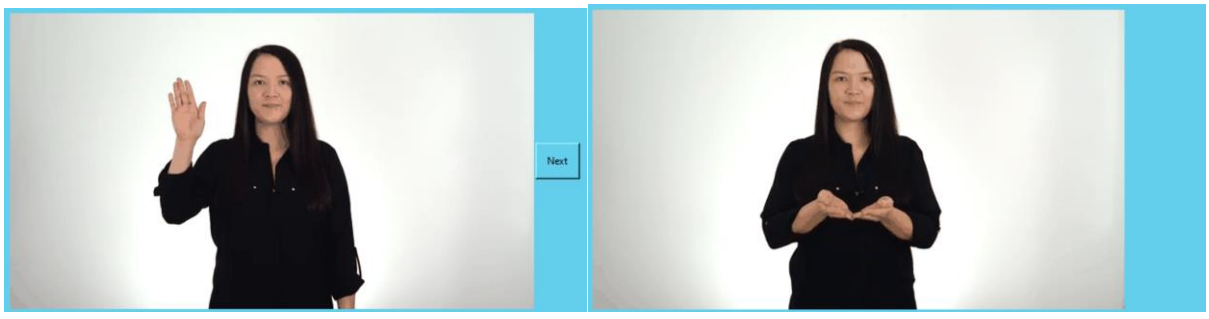
```
def search_to_search_result():
    """switches to the search result page where the signings are presented to the user.
    this function finds the signings of the specific phrases in online database and
    outputs it to the user in the correct order via a .gif"""
    #switches pages
    search_page.hide()
    search_result_page.show()
    signing_page_button.visible = False
    #obtains corrected user input
    corrected_input = ""
    for i in range(len(output_text.value)):
        if output_text.value[i] == ":":
            corrected_input = output_text.value[i+2:]
            break
    #sets intro text value
    search_result_user_input.value = "Searching for: " + corrected_input
    clear_search_page()
    correct_input_list = corrected_input.split()#splits the user input into a list
    del search_output_file_names[:]#clears the global list
    #iterates through each word in the input
    for i in range(len(correct_input_list)):
        temp_file_name = correct_input_list[i] + ".gif"
        search_output_file_names.append(temp_file_name)
    iterate_search_result()#presents the result to the user

def iterate_search_result():
    """Outputs each signing and iterates to the next signing phrase in the list when the button pressed.
    The iteration button only appears if there is more than one phrase.
    Also updates the text at the top indicating what is currently being displayed."""
    #shows the iterate button if there are more than one phrases
    if len(search_output_file_names)>1:
        search_result_next_button.show()
    else:
        search_result_next_button.hide()
    try:
        #presents description of the current phrase being shown
        search_resut_current_output.value = "Currently showing: " + str(search_output_file_names[0]).replace(".gif","")
        search_result_gif_output.image = search_output_file_names[0]#outputs first value in list
        del search_output_file_names[0]#gets rid of the first value in list
    except:
        pass
```

Description/Justification:

This piece of code shows the implementation of multiple phrase inputs in one go. To do this a separate button had to be designed with the function “iterate_search_result”. This function allows the button to iterate through a list of signing phrases. To do this the program also needed a global variable to store the array of file paths to be iterated through by the button.

Testing for Outputting a Phrase – Part 2 – Iteration 1

Output:



“hello world”

Outputting a Phrase – Part 3 – Iteration 1

```
def search_to_search_result():
    """switches to the search result page where the signings are presented to the user.
    this function finds the signings of the specific phrases in online database and
    outputs it to the user in the correct order via a .gif"""
    #switches pages
    search_page.hide()
    search_result_page.show()
    signing_page_button.visible = False
    #obtains corrected user input
    corrected_input = ""
    for i in range(len(output_text.value)):
        if output_text.value[i] == ":":
            corrected_input = output_text.value[i+2:]
            break
    #sets intro text value
    search_result_user_input.value = "Searching for: " + corrected_input
    clear_search_page()
    correct_input_list = corrected_input.split()#splits the user input into a list
    del search_output_file_names[:]#clears the global list
    #iterates through each word in the input
    for i in range(len(correct_input_list)):
        temp_file_name = correct_input_list[i] + ".gif"
        if not path.exists(temp_file_name):#if the video file has not already been created
            for value in total_vids:#searches for the URL
                if correct_input_list[i] == value["Phrase"].lower():
                    output_url = value["URL"]
            ytd = YouTube(output_url).streams.first()
            ytd.download(filename=correct_input_list[i])#downloads video from URL
            print("Video downloaded to local directory")
            file_name = correct_input_list[i] + ".mp4"
            clip = (VideoFileClip(file_name))#creates clip instance
            file_name = file_name.replace(".mp4",".gif")#changes the file type on string
            clip.write_gif(file_name)#converts .mp4 video to .gif
            search_output_file_names.append(file_name)
        else:
            search_output_file_names.append(temp_file_name)
    iterate_search_result()#presents the result to the user
```

Description/Justification:

This piece of code shows the adapted “search_to_search_result” function which works alongside the original “iterate_search_result” function. In this updated code the function uses the online BSL database (developed in functional chunk 3). It also uses the pytube module to download the video from the specific URL and the moviepy module to convert the mp4 video into a gif to be outputted to the user.

Testing for Outputting a Phrase – Part 3 – Iteration 1

Output:

No output was given due to the module not responding as it should. See testing and remedial actions.

Outputting a Phrase – Part 3 – Iteration 2

```
def search_to_search_result():
    """switches to the search result page where the signings are presented to the user.
    this function finds the signings of the specific phrases in online database and
    outputs it to the user in the correct order via a .gif"""
    #switches pages
    search_page.hide()
    search_result_page.show()
    signing_page_button.visible = False
    #obtains corrected user input
    corrected_input = ""
    for i in range(len(output_text.value)):
        if output_text.value[i] == ":":
            corrected_input = output_text.value[i+2:]
            break
    #sets intro text value
    search_result_user_input.value = "Searching for: " + corrected_input
    clear_search_page()
    correct_input_list = corrected_input.split()#splits the user input into a list
    del search_output_file_names[:]#clears the global list
    #iterates through each word in the input
    for i in range(len(correct_input_list)):
        temp_file_name = correct_input_list[i] + ".gif"
        if not path.exists(temp_file_name):#if the video file has not already been created
            for value in total_vids:#searches for the URL
                if correct_input_list[i] == value["Phrase"].lower():
                    output_url = value["URL"]
            ytd = YouTube(output_url).streams.first()
            ytd.download(filename=correct_input_list[i])#downloads video from URL
            print("Video downloaded to local directory")
            file_name = correct_input_list[i] + ".mp4"
            clip = (VideoFileClip(file_name))#creates clip instance
            file_name = file_name.replace(".mp4",".gif")#changes the file type on string
            clip.write_gif(file_name)#converts .mp4 video to .gif
            search_output_file_names.append(file_name)
        else:
            search_output_file_names.append(temp_file_name)
    iterate_search_result()#presents the result to the user
```

Description/Justification:

This piece of code is the same as the one found in the previous iteration with the fixes to the module itself.

Testing for Outputting a Phrase – Part 3 – Iteration 2

“captain hat”

Outputting a Phrase – Part 4 – Iteration 1

```
from gspread import service_account

gc = service_account(filename="credentials.json")
sh = gc.open_by_key("1i5imoTFGfOWSYctfUqwdrHlRHvprSgTXCwWbc327kEs")
worksheet = sh.sheet1#selects the first sheet of the documment
total_vids = worksheet.get_all_records()

def create_url_list(user_string):
    found_values = []#list of phrases that are in input
    user_input_list = user_string.split()
    for i in range(len(user_input_list)):
        #creates 2d list of input [user word,index of given phrase in found_values]
        user_input_list[i] = [user_input_list[i]]
        user_input_list[i].append(None)

    for value in total_vids:#loops through all phrases
        db_phrase = value["Phrase"].lower()#uses lowercase phrases
        if db_phrase in user_string:#if phrase present in input
            found_values.append(value["Phrase"])#appends formatted phrase to list
            for i in range(len(user_input_list)):#loops through each word of user input
                if db_phrase.split()[0] == user_input_list[i][0]:#lines up phrase and user input
                    for j in range(len(db_phrase.split())):#loops through length of phrase
                        if user_input_list[i+j][1] == None:#if there isnt anything found for a phrase
                            user_input_list[i+j][1] = len(found_values)-1
                            #current phrase set as value via pointer
                        elif len(db_phrase.split()) > len(found_values[user_input_list[i+j][1]]):
                            #if current phrase is bigger than previous phrase
                            user_input_list[i+j][1] = len(found_values)-1
                            #current phrase set at value via pointer

    for i in range(len(user_input_list)):#iterating through user input
        if user_input_list[i][1] == None:#find word that has no signings
            temp_list = []
            for j in range(len(user_input_list[i][0])):#loops through length of word
                for k in range(len(found_values)):#loops through the found values
                    if user_input_list[i][0][j] == found_values[k].lower():
                        #if current letter is the letter in found values
                        temp_list.append(k)#appends index of letter to temp list
                        break
            user_input_list[i][1] = temp_list
            #temp list is used in place where other words have a single phrase

    print(found_values)
    return user_input_list
```

Description/Justification:

This piece of code shows the algorithm used to sift through the user's input and find the phrases that best suit the needs of the user. This is done by using a pointer system where the program will iterate through the database until it finds a phrase in the user's input. Once the phrase has been identified if the given phrase is larger than the previous phrase the pointer will point to the larger phrase instead. This ensures that the largest phrases are used and therefore the most accurate. This algorithm also compensates for words that are inputted by the user but are not specific phrases and so are instead made up of their alphabetical components.

Testing for Outputting a Phrase – Part 4 – Iteration 1

Output:

```
        print(found_values)
        return user_input_list

    print(create_url_list("i love you turing"))

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\zaina\Documents\Computer Science\Zain Altaf 4132 OCR H446-0
n37/python.exe" "c:/Users/zaina/Documents/Computer Science/Zain Altaf 4
['G', 'I', 'I Love You', 'Love', 'N', 'R', 'T', 'U', 'You']
[['i', 2], ['love', 2], ['you', 2], ['turing', [6, 7, 5, 1, 4, 0]]]
PS C:\Users\zaina\Documents\Computer Science\Zain Altaf 4132 OCR H446-0
```

Output shows the list of phrases found and the list of user input with the assigned pointer for each phrase. Each word in the list points to the phrase that will be displayed. Phrases that do not have a specific phrase show a list of indexes of letters that it will point to in progression.

Outputting a Phrase – Part 4 – Iteration 2

```
def gif_setup():
    """this function finds the signings of the specific phrases in online database"""
    #obtains corrected user input
    corrected_input = ""
    for i in range(len(output_text.value)):
        if output_text.value[i] == ":":
            corrected_input = output_text.value[i+2:]
            break
    #sets intro text value
    search_result_user_input.value = "Searching for: " + corrected_input
    clear_search_page()
    del search_output_file_names[:]#clears the global list
    user_input_list,found_values = create_url_list(corrected_input.lower())
    #obtains the total phrases found in the input and a 2d list containing
    #each word in the input and a pointer to a value in the found_values list
    temp_list = []
    counter = 0
    while counter < len(user_input_list):
        search_output_file_names.append(found_values[user_input_list[counter][1]])
        counter+=len(found_values[user_input_list[counter][1]].split())
    iterate_search_result()

def iterate_search_result():
    """Outputs each signing and iterates to the next signing phrase in the list when the button pressed.
    Downloads the next signing video phrase and converts it into .gif.
    The iteration button only appears if there is more than one phrase.
    Also updates the text at the top indicating what is currently being displayed."""
    search_result_gif_output.value = None
    #shows the iterate button if there are more than one phrases
    if len(search_output_file_names)>1:
        search_result_next_button.show()
    else:
        search_result_next_button.hide()
    try:
        #presents description of the current phrase being shown
        search_resut_current_output.value = "Currently showing: "+str(search_output_file_names[0])
        if not os.path.exists(search_output_file_names[0]+".gif"):
            #if video needs to be downloaded
            for value in total_vids:#obtains correct URL
                if search_output_file_names[0] == value["Phrase"]:
                    output_url = value["URL"]
                    break
            ytd = YouTube(output_url).streams.first().download(filename=search_output_file_names[0])
            print("Video downloaded to local directory")
            file_name = search_output_file_names[0] + ".mp4"
            clip = (VideoFileClip(file_name))#creates clip instance
            file_name = file_name.replace(".mp4",".gif")#changes the file type on string
            clip.write_gif(file_name)#converts .mp4 video to .gif

            search_result_gif_output.image = file_name#outputs first value in list
            del search_output_file_names[0]#gets rid of the first value in list
        else:

            file_name = search_output_file_names[0]+".gif"
            search_result_gif_output.image = file_name#outputs first value in list
            del search_output_file_names[0]#gets rid of the first value in list
    except:
        pass
```

Description/Justification:

This piece of code shows the integration of the create_url_list function shown in the previous iteration into the GUI. This required the program to abstract the user's validated input (corrected_input) and searching through the results for the signing phrases found, outputting it to the iterate_search_result function. This function is called once during initialisation and every time the next signing button is pressed. Here the program uses the list of names of phrases to output and the online database to download and convert the videos to .gif format. This process occurs after each calling of the function – this is due to the fact that the conversion from .mp4 to .gif takes a few seconds in python and a multiple phrase string may cause a sizable delay. CONT ->

Testing for Outputting a Phrase – Part 4 – Iteration 2

Output:

```
    search_output_file_names.append(found_values[user_input_list[counter][1]])
TypeError: list indices must be integers or slices, not list
```

See testing and remedial actions.

Outputting a Phrase – Part 4 – Iteration 3

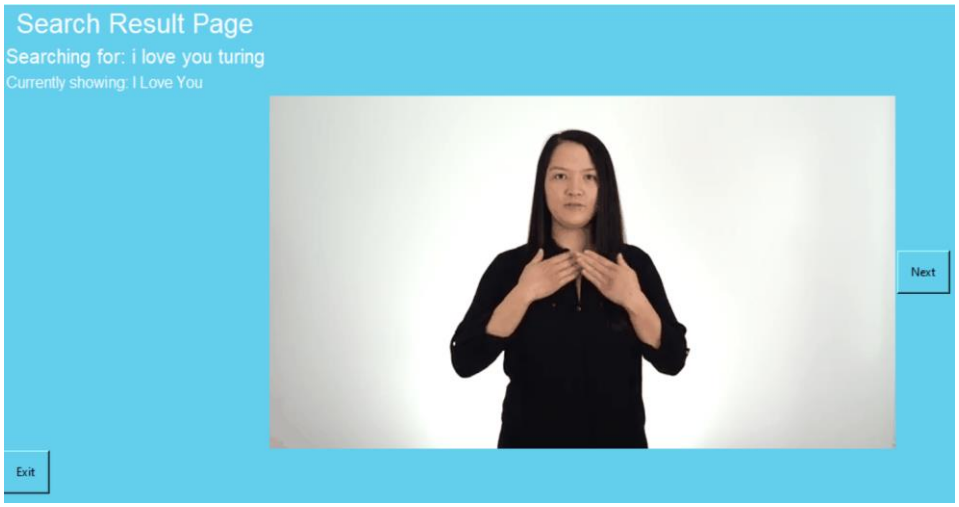
```
def gif_setup():
    """this function finds the signings of the specific phrases in online database"""
    #obtains corrected user input
    corrected_input = ""
    for i in range(len(output_text.value)):
        if output_text.value[i] == ":":
            corrected_input = output_text.value[i+2:]
            break
    #sets intro text value
    search_result_user_input.value = "Searching for: " + corrected_input
    clear_search_page()
    del search_output_file_names[:]#clears the global list
    user_input_list,found_values = create_url_list(corrected_input.lower())
    #obtains the total phrases found in the input and a 2d list containing
    #each word in the input and a pointer to a value in the found_values list
    temp_list = []
    counter = 0
    while counter < len(user_input_list):
        #this loops round the 2d list and abstracts all of the phrase descriptions
        #by looking at each of the indexes in found_values list
        if type(user_input_list[counter][1]) is list:
            for j in range(len(user_input_list[counter][1])):
                search_output_file_names.append(found_values[user_input_list[counter][1][j]])
            counter+=1
        else:
            search_output_file_names.append(found_values[user_input_list[counter][1]])
            counter+=len(found_values[user_input_list[counter][1]].split())
    iterate_search_result()#iterates once

def iterate_search_result():
    """Outputs each signing and iterates to the next signing phrase in the list when the button pressed.
    Downloads the next signing video phrase and converts it into .gif.
    The iteration button only appears if there is more than one phrase.
    Also updates the text at the top indicating what is currently being displayed."""
    #shows the iterate button if there are more than one phrases
    if len(search_output_file_names)>1:
        search_result_next_button.show()
    else:
        search_result_next_button.hide()
    try:
        #presents description of the current phrase being shown
        search_resut_current_output.value = "Currently showing: "+str(search_output_file_names[0])
        if not path.exists(search_output_file_names[0]+".gif"):
            #if video needs to be downloaded
            for value in total_vids:#obtains correct URL
                if search_output_file_names[0] == value["Phrase"]:
                    output_url = value["URL"]
                    break
            ytd = YouTube(output_url).streams.first()
            ytd.download(filename=search_output_file_names[0])#downloads video from URL
            print("Video downloaded to local directory")
            file_name = search_output_file_names[0] + ".mp4"
            clip = (VideoFileClip(file_name))#creates clip instance
            file_name = file_name.replace(".mp4",".gif")#changes the file type on string
            clip.write_gif(file_name)#converts .mp4 video to .gif
        else:
            file_name = search_output_file_names[0]+".gif"
        search_result_gif_output.image = file_name#outputs first value in list
        del search_output_file_names[0]#gets rid of the first value in list
    except:
        pass
```

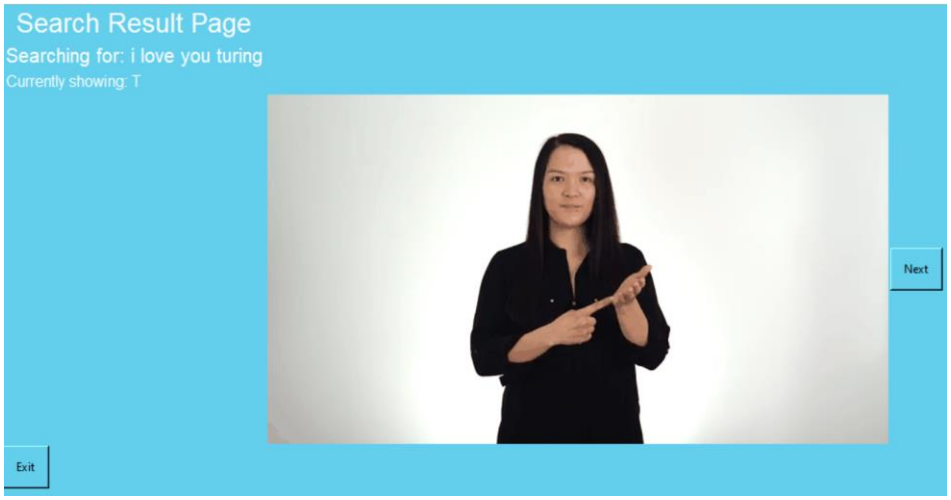
Description/Justification:

This piece of code checks whether the value containing the index is a list first. If so the program extracts the indexes and utilises them the same way it does for non-alphabet phrases. This means the same format can still be used.

Testing for Outputting a Phrase – Part 4 – Iteration 3

Output:





"I love you, T" (only the t is currently shown since it is the first letter of "turing" which is not a registered word in the database and so the program tries to spell it out letter by letter.

Functional chunk 2 – Searching through database

Related Sections:

- Algorithms
- Prototypes
- Code
- Tests
- Remedial Actions
- Code Listings

Functional Chunk 2 will focus on developing a searching algorithm to search through the database more efficiently. This function chunk will include the steps taken to optimise the searching algorithm to achieve the fastest result.

Back to contents page

Optimised Database Search Part 1 – Iteration 1

```
database_list= [["good morning",None,None],["good evening",None,None],["rain",None,None],["raining",None,None]]
user_input = "rains78932"
new_string = ""
#input validation - removes numbers/special characters
user_input = user_input.lower()
for i in range(len(user_input)):#iterates through user input
    if user_input[i] == " ":
        new_string+=user_input[i]#adds spaces to final string
    elif user_input[i].isalpha():
        new_string+=user_input[i]#adds letter characters

def phrase_finder(database,user_input,index):
    """iterates though each phrase in the database list letter by letter
    and checks agains the same index letter of the user input.
    returns the values that correspond"""
    temp_data_list = []
    for i in range(len(database)):#iterates through all the values of database
        data_string = database[i][0]#actual string value of each iteration
        if index<len(data_string):#in case the user input is longer than the the phrase
            phrase_letter = data_string[index]
            if phrase_letter == user_input[index]:
                temp_data_list.append(database[i])#adds corresponding phrases to be returned
    return temp_data_list

for i in range(len(new_string)):#loops through each character in the entered string
    holder_list  = phrase_finder(database_list,new_string,i)
    if len(holder_list) == 0:
        pass
    else:
        database_list = holder_list
    #updates database_list each iteration so that only the currently corresponding
    #values are added means that each letter iteration for the user input will be
    #searched against an increasingly condensed list of values
    #making it more efficient to find
print("user input: " + str(user_input))
print("Values found from database: " + str(database_list))
```

Description/Justification:

Alongside the input validation developed in the previous prototype this prototype includes a letter by letter searching algorithm. This works by searching through the database to find phrases with the corresponding first letter and adding those corresponding phrases to a list to be returned. This function is carried out in a loop so that once the first letter has been checked and the list containing all phrases with the same first letter has been formed the list can be re-iterated through using the second letter to find phrases with the same second letter. This continues for the length of the phrase until all the values are found. Note: making the new database_list each time for the "new batch" of corresponding phrases makes sense as it shortens the length of the list so that the program has to iterate through less values and therefore makes it faster.

Testing for Optimised Database Search - Part 1 – Iteration 1

Output:

```
user input: rains78932
Values found from database: [['rain', None, None], ['raining', None, None]]
```

Optimised Database Search Part 2 – Iteration 1

```
from fuzzywuzzy import fuzz, process #'fuzzy' matching
import json
with open("Development\Chunk1_Searching\words_dictionary.json") as f:
    database = json.load(f)#loads the word bank database
f.close()

user_input = input("please enter the value you would like to search: ")
total_list = []
user_input = user_input.split()
return_list = []

for user_word in user_input:
    single_word_list = []#this list stores potential phrases for the user input
    for phrase in database:#loops through the entire database
        options_list = []#temp list to store each potential phrase
        percent_acc = fuzz.ratio(user_word,phrase)
        #uses fuzz.ratio to find how similar the input is to words in the database
        if percent_acc > 80:#if the word is relatively accurate
            options_list.append(phrase)#adds the phrase to the temp list
            options_list.append(percent_acc)#adds the level of accuracy to the temp list
            for i in range(len(single_word_list)):#loops through the entire selection of possibilities
                phrase_percent = int(single_word_list[i][1])
                if percent_acc>=phrase_percent:
                    single_word_list.insert(i,options_list)
                    #inserts the temp list with the newest phrase so that the final list is sorted
                    break#breaks out of the loop
                elif (i+1) == len(single_word_list):
                    single_word_list.insert(i+1,options_list)
                    #if the phrase is the least accurate it is inserted to the back of the list
                    break#breaks out of the loop
            if len(single_word_list) == 0:#if there are no values in the list
                single_word_list.append(options_list)
    total_list.append(single_word_list)#adds the list of all the potential phrases to the total list

for word_result in total_list:
    if len(word_result) == 0:#if no results were found
        return_list.append([])
        return_list[-1].append("no results found please try again")#error message
        return_list[-1].append(0)#identifier value that can be used to show no correlation
    else:
        return_list.append(word_result[0])#adds the most accurate value

#outputs to the user what the program thinks was meant by the user
print("\nSearch results: ")
for i in range(len(return_list)):
    if return_list[i][1] == 0:
        print("input '" + str(user_input[i]) + "' could not be understood. please try again.")
    elif return_list[i][1]<100:
         print("input '" + str(user_input[i]) + "' was not found. showing results for: " + str(return_list[i][0]))
    else:
        print("showing results for: " + str(return_list[i][0]))
```

Description/Justification:

This piece of code showcases the use of the Fuzzywuzzy module as a source of validation for the strings inputted by the user. This program takes the input from the user and compares it to every string in the database and stores the phrases that have an 80% or higher accuracy. This means that multiple phrases may be added with only the most accurate being outputted.

Testing for Optimised Database Search – Part 2 – Iteration 1

Output:

```
please enter the value you would like to search: this is a random string klduvhreoahj

Search results:
showing results for: this
showing results for: is
showing results for: a
showing results for: random
showing results for: string
input 'klduvhreoahj' could not be understood. please try again.
```

Optimised Database Search Part 2 – Iteration 2

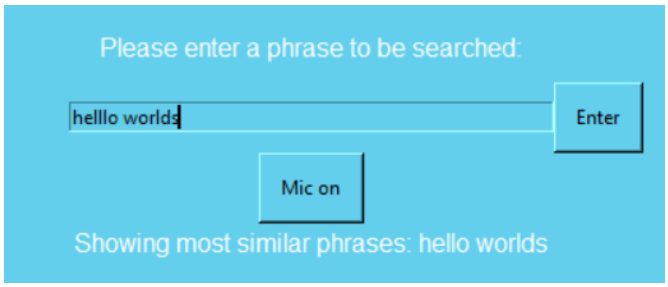
```
def searching_database(user_input,output_text):
    """takes user input as an attribute and searches through the database for it.
    this includes a fluffy search which outputs the most accurate word it can find in the database.
    this function also updates the top 20 list"""
    user_input_list = user_input.split()
    output_string = ""
    temp_list = []
    total_list = []
    for i in range(len(user_input_list)):#iterates through each inputted word
        single_word_list = fluffy_search(database,user_input_list[i],80)#creates a list of possible phrases
        total_list.append(single_word_list)#appends all possible phrases for each inputted word to total list
        if len(single_word_list) != 0:#if there were no found possibilities
            my_gui_string = top_20_adjust(top_20_list,total_list[i][0][0])
            #adds the searched phrase to the top 20 list and returns the new top 20 list to output
            output_string += total_list[i][0][0] + " " #adds new value to output to be printed

    if output_string == "":#if nothing was added to the string - if no words found in database
        output_text.value = "sorry. that couldn't be found, please try again"
    else:
        output_string = output_string.strip()#gets rid of any possible spaces on the ends
        correction = False#used to check if input needed to be corrected
        for word_list in total_list:
            if word_list[0][1] != 100:#if the accuracy value was not 100
                correction = True#phrase was corrected
        #different output message shown telling user if the phrase was corrected or not
        if correction == True:
            output_text.value = "Showing most similar phrases: " + str(output_string)
        else:
            output_text.value = "Found it: " + str(output_string)
        #presents the validated output to the user
    top_20_output_text.value = my_gui_string
```

Description/Justification:

Iteration 2 shows the implantation of the Fluffy search into the graphical user interface via the function searching_database. It uses the fuzzy wuzzy module to search through the user input and obtain the most accurate phrases. These phrases can then be outputted with the most accurate one (index 0) being used for the database search.

Testing for Optimised Database Search – Part 2 – Iteration 2

Output:



Functional chunk 3 – Consolidation of learning

Related Sections:

- Algorithms
- Prototypes
- Code
- Tests
- Remedial Actions

Functional chunk 3 will focus on the main piece of functionality associated with the user's solo learning. This will be accomplished by implementing a trial and improvement strategy into the GUI. This will work by showing the user a phrase and allowing them to attempt the signing before showing the video of the true signing. The user can then confirm whether they got it right or wrong; based on this the program will increase or decrease the frequency that a certain phrase is shown to allow for spaced repetition.

Spaced repetition is a popular way of learning new content as it allows the user to continuously practise lots of smaller pieces of a bigger topic. The BSL phrases available can be learned to fluency using spaced repetition. Alongside this is the ability to choose the flow of learning – the user can choose whether they would like to see the signing action (gif) first and interpret a phrase description or vice versa. This allows the user to develop what it may feel like in real life when both watching someone sign as well as signing yourself.

Back to contents page

Creating Online Database – Part 1 – Iteration 1

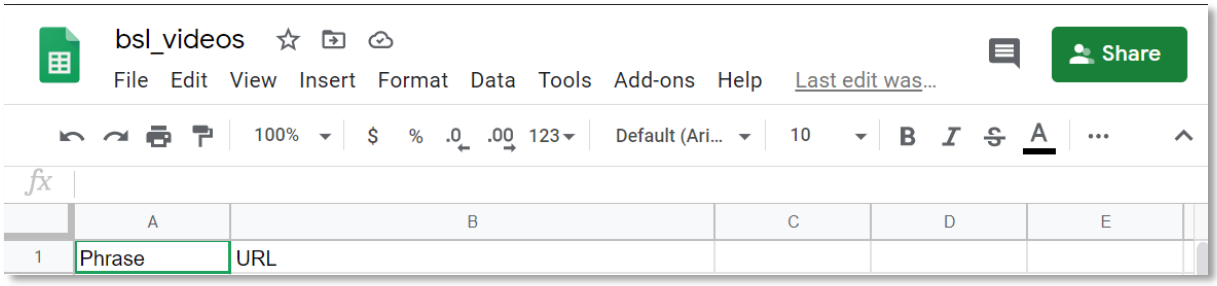
```
import gspread
import json

gc = gspread.service_account(filename="Development\Chunk2_Consolidation\credentials.json")
sh = gc.open_by_key("1iSf4VziTbvs0nuzu7X-kcxqPyiuBg-iB4Usykh0mS7I")
#this is the key for the google docs. this can be found in the URL after the /d/
worksheet = sh.sheet1#selects the first sheet of the documment
#example functions
new_phrase = ["World","https://www.youtube.com/watch?v=pDFTmla-SrE&list=UUHctNnLPrEr9_MX2ZPtS7qQ&index=134"]

worksheet.insert_row(new_phrase,2)
#inserts the profile after row 1 so that it is row 2
#this can be used if the doccument needed to be in alphabetical order
```

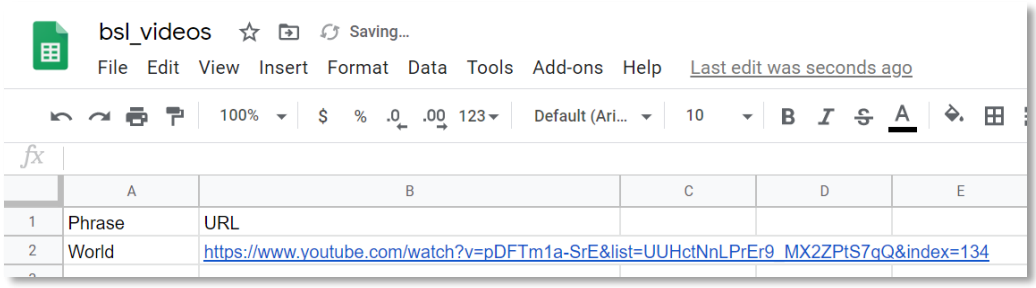
Description/Justification:

These screenshots show the use of the Gspread module as an online database to store data for the program. This allows the program to be used from anywhere without the need for local databases. After installing the Gspread module credentials needed to be made to access the online file from the python program. Once this was done and the program had full access to the file, methods such as append and insert could be used to edit the document in whichever way was necessary. These screenshots show the use of an online database to store BSL phrases and their URLs. Creating the online database of phrases will require me to continuously add relevant phrases and their URL to the database.

Testing for Creating Online Database – Part 1 – Iteration 1

Output:





Creating Online Database – Part 2 – Iteration 1

```
import gspread
import random
gc = gspread.service_account(filename="Development\Chunk2_Consolidation\credentials.json")
sh = gc.open_by_key("1iSf4VziTbvs0nuzu7X-kcxqPyiuBg-iB4Usykh0mS7I")#api key
worksheet = sh.sheet1#selects the first sheet of the documment
total_vids = worksheet.get_all_records()#gets all values

my_list = []
for i in range(20):#creates a list of 20 values
    random_phrase = random.choice(total_vids)#selects a random value
    if random_phrase not in my_list:#checks if value is a duplicate
        my_list.append(random_phrase)#adds value to the list

for i in range(len(my_list)):
    input(my_list[i]["Phrase"])#showcases all the values
```

Description/Justification:

This piece of code shows the use of the online database to create an array of random phrases for the user to practise. The random selection of phrases will only be used for a "Non-signed in" user but program will need to access the online database whether the user is logged in or out. While iterating through the values in the database the program ensures the array is formed without any duplicates.

Testing for Creating Online Database – Part 2 – Iteration 1

Output:

```
Sainsburys
Music
Dry
W
Avocado
Moon
Phone
K
No (Don't)
Bug
Everything
More
Review
Snow
When
Sofa
Home (House)
Sick (Poorly)
Comb
Swimming
```

Creating Online Database – Part 3 – Iteration 1

```
#imports
from pytube import YouTube, Playlist
from moviepy.editor import *

#funtions
def Search(search):
    #opens .csv file and reads in data
    database = open("Development\Chunk2_Consolidation\BSL_DATABASE.csv","r")
    database = database.readlines()
    #loops through each phrase
    for i in range(len(database)):
        #checks whether the user input string is in the database string
        if search in database[i]:
            #if the database string is found the URL is extracted
            result = database[i]
            result = result.replace(search, '')
            result = result.replace(',','')
            result = result.strip()
            #URL is then outputted
            return result

def Download(URL):
    #if no URL was found - if the phrase was not found
    if URL==None:
        print("ERROR")
    else:
        #video is downloaded to local drive
        ytd = YouTube(URL).streams.first().download()#downloads video from URL
        print("Video downloaded to local directory")
        VariableName = (YouTube(URL).streams[0].title)#obtains title of yt video
        file_name = ""
        for i in range(len(VariableName)):#iterates through title string
            #gets rid of any non alphabet characters in title
            if VariableName[i] == " ":
                file_name += VariableName[i]
            elif VariableName[i].isalpha():
                file_name += VariableName[i]
        file_name += ".mp4"
        print("Current file name: ")
        print(file_name)
        clip = (VideoFileClip(file_name))#creates clip instance
        file_name = file_name.replace(".mp4",".gif")#changes the file type on string
        print("Converted file name: ")
        print(file_name)
        clip.write_gif(file_name)#converts .mp4 video to .gif

user_input = input("Enter a word: ")
#formats the user input to the correct format
user_input = user_input[0].upper() + user_input[1:].lower()
#finds the URL for the phrase
link = Search(user_input)
#downloads the video
download = Download(link)
```

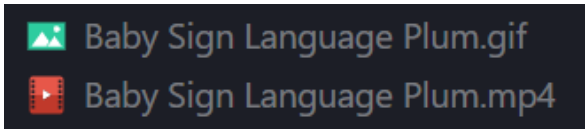
Description/Justification:

Program obtains a specific video for a specific phrase from the BSL database. Using the phrase description, the program obtains the URL to download the video. This is then converted to .gif format to be outputted into the GUI. In this case the local database is used instead of the online database. This will be amended when implemented into the GUI.

[Testing for Creating Online Database – Part 3 – Iteration 1]

Output:

```
Enter a word: Plum
Video downloaded to local directory
Current file name:
Baby Sign Language Plum.mp4
Converted file name:
Baby Sign Language Plum.gif
MoviePy - Building file Baby Sign Language Plum.gif with imageio.
```



Creating User Interface – Part 1 – Iteration 1

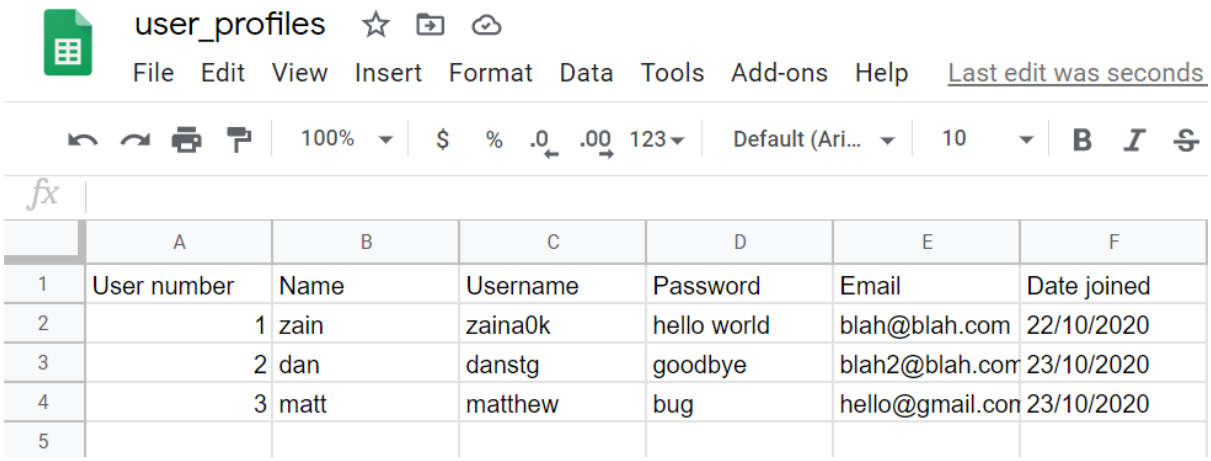
```
import gspread
from datetime import date
gc = gspread.service_account(filename="credentials.json")
sh = gc.open_by_key("1eYarci7vaZndWdLMJJlJj_KkWKNece3s-kIm-o2pUgA")
#this is the key for the google docs. this can be found in the URL after the /d/
worksheet = sh.sheet1#selects the first sheet of the documment
total_users = worksheet.get_all_records()

class User:
    """class that creates user objects"""
    user_counter = 0
    def __init__(self,name,username,password,email):
        """initialising the class"""
        today = date.today()
        self.name = name
        self.date_joined = today.strftime("%d/%m/%Y")
        self.username = username
        self.password = password
        self.email = email
        self.user_number = User.user_counter + 1
        User.user_counter += 1
    def __str__(self):
        """print method shows all the values of the object"""
        print_str = ""
        print_str+= "user number: " + str(self.user_number) + "\n"
        print_str+= "name: " + str(self.name)+ "\n"
        print_str+= "username: " + str(self.username)+ "\n"
        print_str+= "password: " + str(self.password)+ "\n"
        print_str+= "date joined: " + str(self.date_joined)+ "\n"
        print_str+= "email: " + str(self.email)
        return print_str

def add_user_to_database(user_object):
    """function abstracts all of the data from the user object and appends the
    data to the database"""
    temp_list = []
    temp_list.append(user_object.user_number)
    temp_list.append(user_object.name)
    temp_list.append(user_object.username)
    temp_list.append(user_object.password)
    temp_list.append(user_object.email)
    temp_list.append(user_object.date_joined)
    worksheet.append_row(temp_list)
def set_user_counter():
    """method that sets the user counter as the most recent user number.
    This function should be called each time the program is run so that
    the user number can be set to the correct number."""
    User.user_counter = total_users[-1]["User number"]

set_user_counter()
user_object = User("dan","danstg","goodbye","blah2@blah.com")
user_object2 = User("matt","matthew","bug","hello@gmail.com")
add_user_to_database(user_object)
add_user_to_database(user_object2)
```

Description/Justification:

This piece of code uses a User class to create user objects containing data about the client. This data is then outputted to an online database containing all user data (this user database is separate from the bsl database) via a function. A procedure is used to pre-set the user counter to the correct amount at the beginning of the running of the program.

Testing for Creating User Interface – Part 1 – Iteration 1

Output:



	A	B	C	D	E	F
1	User number	Name	Username	Password	Email	Date joined
2	1	zain	zaina0k	hello world	blah@blah.com	22/10/2020
3	2	dan	danstg	goodbye	blah2@blah.com	23/10/2020
4	3	matt	matthew	bug	hello@gmail.con	23/10/2020
5						

Due to the set_user_counter procedure being run at the beginning of the program, the program identifies that there was already a user in the database and increments the user_counter by 1 in order to accommodate for this.

Creating User Interface – Part 2 – Iteration 1

```
import hashlib#importing module

user_input1 = input("please enter your first input: ")
user_input1 = str.encode(user_input1)#turns string into byte string format
user_input2 = input("please enter your second input: ")
user_input2 = str.encode(user_input2)#turns string into byte string format
#uses hash function to digest the byte strings into hex code
hashed_user_input1 = hashlib.sha224(user_input1).hexdigest()
hashed_user_input2 = hashlib.sha224(user_input2).hexdigest()
#checks whether the hashed strings are the same
#this should only occur if the initial strings were the same
if hashed_user_input1 == hashed_user_input2:
    print("both hashed strings are the same")
else:
    print("the hashed strings are not the same")
```

Description/Justification:

This piece of code shows the processed used in order to "encrypt" the user's password using a one-way encryption. The encrypted string will be stored as the user's password so that only when inputting the original password will it be the same as the stored string.

Testing for Creating User Interface – Part 2 – Iteration 1

Output:

```
please enter your first input: hello
please enter your second input: world
the hashed strings are different. try again
```

```
please enter your first input: hello
please enter your second input: hello
both hashed strings are identical
```

Here the output shows that the program will compare the two encrypted strings (the encrypted input and the password already stored in the database) which will only be the same if the original input was the same.

Creating User Interface – Part 3 – Iteration 1

```
#imports
import smtplib
import os
#obtaining email and password from local environment variables
EMAIL_ADDRESS = os.environ.get("EMAIL_USER")
EMAIL_PASSWORD = os.environ.get("EMAIL_PASS")

with smtplib.SMTP("smtp.outlook.com",587) as smtp:
    #initialising the connection
    smtp.ehlo()
    smtp.starttls()
    smtp.ehlo()

    smtp.login(EMAIL_ADDRESS,EMAIL_PASSWORD)

    subject = "This is a test subject"
    body = "This is a test body for using python to send emails"

    msg = f"Subject: {subject}\n\n{body}"#formats the message
    smtp.sendmail(EMAIL_ADDRESS,EMAIL_ADDRESS,msg)
    #the .sendmail method takes the sender's email, recipient email, message
```

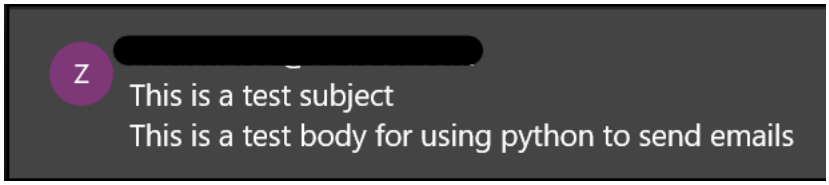
Description/Justification:

this piece of code shows the use of the smtplib library to email a message to a specific email. The program obtains the email and password used to send the email via environment variables. On my local machine this allows me to present my code to a user without personal data being compromised given that my personal email was used for the duration of this project.

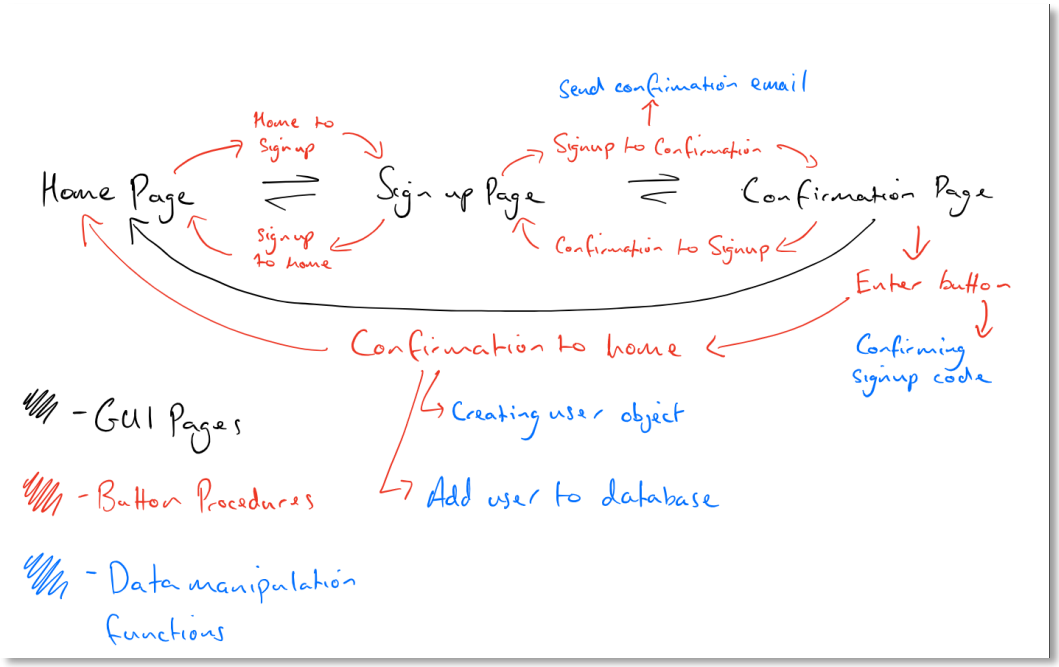
The email module will be used to send an email to the user in order to confirm that they have entered the correct detail (their email, etc).

Testing for Creating User Interface – Part 3 – Iteration 1

Output:



Creating User Interface – Part 4 – Iteration 1



Description:

This diagram represents the process of creating a new user in the graphical user interface, with the colours representing different things. The **black** in the diagram represents the different pages such as the home page and sign up page with arrows indicating the direction of travel. The red describes the procedures that are called in order to change the pages and clear the pages that are left if necessary (e.g. when travelling from the signup page back to the home page the signup page is cleared and returned to its default look). The blue shows the functions that are called in order to manipulate functions and check the user's input in order to obtain the desired output. The data manipulation functions are called at some point by the indirect pressing of a button (e.g. a button may call a procedure which may call a data manipulation function)

```
def send_confirmation_email():
    """function sends an email to the user containing a confirmation code"""
    sign_up_code = randint(100000,999999)#generates a random 6-digit code as the validation code
    #obtains the user's username to be used as refrence in the email
    username = signup_username_input.value
    email = signup_email_input.value
    password = signup_password_input.value
    name = signup_name_input.value
    passed = False
    with smtplib.SMTP("smtp.outlook.com",587) as smtp:
        #initialising the connection
        smtp.ehlo()
        smtp.starttls()
        smtp.ehlo()
        smtp.login(EMAIL_ADDRESS,EMAIL_PASSWORD)
        #creates the message in the email
        subject = "Sign Language Translator Sign Up"
        body = "This is a confirmation email for an account made for Sign Language Translator.\n"
        body+="Username: " + str(username) + "\n\n"
        body += "Please enter the following code to verify your account: " + str(sign_up_code)
        msg = f"Subject: {subject}\n\n{body}"#formats the message
        try:
            if len(name)>0 and len(password)>0 and len(username)> 0:
                smtp.sendmail(EMAIL_ADDRESS,email,msg)#sends message from admin email to user
                passed = True
        except:
            pass
    return sign_up_code,passed
```

```
def confirming_sign_up_code():
    """procedure checks whether the user has entered the correct confirmation code"""
    try:
        user_input = int(confirmation_code_box.value)
        sign_up_code = int(confirmation_holder.value)
        if user_input == sign_up_code:
            confirmation_output_text.value = "Code confirmed."
            confirmation_continue_button.show()
            confirmation_enter_button.hide()
        else:
            #if the code wasn't correct
            confirmation_output_text.value = "Incorrect code. Please recheck your email"
    except:
        #if the input contains something that isnt a number
        confirmation_output_text.value = "Please enter 6-digit code"
```

```
def creating_user_object():
    """creates and returns a user object using values from signup page"""
    name = signup_name_input.value
    username = signup_username_input.value
    password = signup_password_input.value
    email = signup_email_input.value
    global user_object
    user_object = User(name,username,password,email)#creates user object

def add_user_to_database(user_object):
    """function abstracts all of the data from the user object and appends the
    data to the database. This data is added to the online database"""
    temp_list = []
    #values are added to an array
    temp_list.append(user_object.user_number)
    temp_list.append(user_object.name)
    temp_list.append(user_object.username)
    temp_list.append(user_object.password)
    temp_list.append(user_object.email)
    temp_list.append(user_object.date_joined)
    #array added to the online user database in a row
    user_worksheet.append_row(temp_list)
```

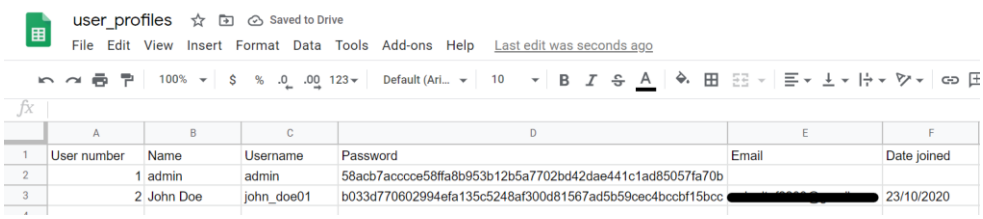
Description:

These pieces of code present the data manipulation functions of the diagram (blue) and are therefore the most significant parts. These functions allow the program to run by handling the user's input by validating it. This is shown with the confirmation code email used to confirm the email and username via a confirmation code. This code is then checked by the confirming_sign_up_code function which if successful, allows the user to create an account via a user object and uploading their details to the online database.

Testing for Creating User Interface – Part 4 – Iteration 1

Output:

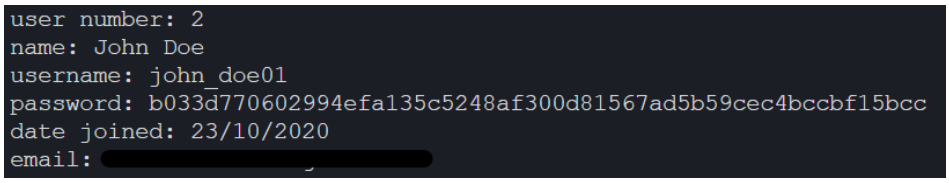
```
This is a confirmation email for an account made for Sign Language Translator.
Username: john_doe01

Please enter the following code to verify your account: 448395
```



user_profiles Saved to Drive

File Edit View Insert Format Data Tools Add-ons Help Last edit was seconds ago

	A	B	C	D	E	F
1	User number	Name	Username	Password	Email	Date joined
2	1	admin	admin	58acb7acccce58ffa8b953b12b5a7702bd42dae441c1ad85057fa70b		
3	2	John Doe	john_doe01	b033d770602994efa135c5248af300d81567ad5b59cec4bccbf15bcc	[redacted]	23/10/2020



```
user number: 2
name: John Doe
username: john_doe01
password: b033d770602994efa135c5248af300d81567ad5b59cec4bccbf15bcc
date joined: 23/10/2020
email: [redacted]
```

These screenshots show the data that has been processed and stored by the program. The emails are blacked out due to testing being done with my personal email.

Creating User Interface – Part 4 – Iteration 2

```
def existing_user_login(username,password):
    """Creates a user object from user database.
    Correct details are assured as username is unique(primary key)"""
    #refreshes the local database from the online database
    current_user = False
    for profile in total_users:
        if profile["Username"] == username:#looks for user profile
            init_password = str.encode(password)
            #encodes the password in the same way so that they match if inputs match
            hashed_password = hashlib.sha224(init_password).hexdigest().strip("0")
            if profile["Password"] == hashed_password:#if password is correct
                #obtains all the user information
                user_number = profile["User number"]
                name = profile["Name"]
                username = profile["Username"]
                password = profile["Password"]
                email = profile["Email"]
                date_joined = profile["Date joined"]
                started_TL = profile["Started tailored learning"]
                learning_path = profile["Learning path"]
                total_right = profile["Total right"]
                total_wrong = profile["Total wrong"]
                creating_user_object(name,username,password,email,
                user_number,date_joined,started_TL,learning_path)
                #creates a user object
                current_user = True
    return current_user
```

Description/Justification:

The program uses the username as a primary key (this is validated during signup where a user must make a unique username, etc.) to search for the specific user. Once found it encodes the given password in the same way it was encoded during signup. This means that if the password input is the same as the password set during signup then it will be encoded the same and therefore have the same value. The rest of the values are then obtained from the online database.

Testing for Creating User Interface – Part 4 – Iteration 2

Output:

Interface gets stuck here and doesn't allow the user to enter the signed in program.

See testing and remedial actions.

Creating User Interface – Part 4 – Iteration 3

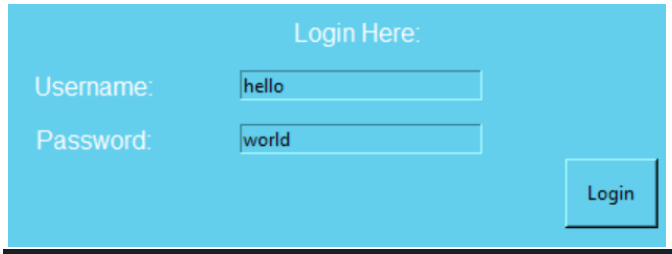
```
def existing_user_login(username,password):
    """Creates a user object from user database.
    Correct details are assured as username is unique(primary key)"""
    global total_users
    total_users = user_worksheet.get_all_records()
    #refreshes the local database from the online database
    current_user = False
    for profile in total_users:
        if profile["Username"] == username:#looks for user profile
            init_password = str.encode(password)
            #encodes the password in the same way so that they match if inputs match
            hashed_password = hashlib.sha224(init_password).hexdigest()
            if profile["Password"] == hashed_password:#if password is correct
                #obtains all the user information
                user_number = profile["User number"]
                name = profile["Name"]
                username = profile["Username"]
                password = profile["Password"]
                email = profile["Email"]
                date_joined = profile["Date joined"]
                started_TL = profile["Started tailored learning"]
                learning_path = profile["Learning path"]
                creating_user_object(name,username,password,email,
                user_number,date_joined,started_TL,learning_path)
                #creates a user object
                current_user = True
    return current_user
```

Description/justification:

Corrects problem with online database being out of date due to checking a local copy. Uses global keyword to obtain the total_users variable and replace it with a fresh copy imported from the online user database.

Testing for Creating User Interface – Part 4 – Iteration 3

Output:



Here in the output you can see the input of the username and password (the password will be starred out during the running of the program and is shown for observation only).

The program then accessed the details found in the online database with the password stored in its encrypted form.

```
user number: 2
name: [redacted]
username: hello
password: 6d1c301ca37b272fe11e939600ea3335d7f69abd0e7b25f5e1d792ee
date joined: 31/10/2020
email: [redacted]
started tailored learning? No
current learning path: None
```

Creating Tailored Learning - Part 1– Iteration 1

```
def create_random_practice_list():
    """creating a list of 10 random phrases with URL"""
    my_list = []
    for i in range(10):
        random_phrase = choice(total_vids)
        if random_phrase not in my_list:
            my_list.append(random_phrase)
    return my_list
```

```
def starting_consolidation():
    begin_text.value = "Beginning Random Consolidation"
    begin_practise_button.hide()
    consolidation_finish_text.hide()
    global consolidation_list# uses global list so that it can be accessed during iteration
    consolidation_list = create_random_practice_list()#creates a random list of 20 phrases
    iterate_consolidation_output()#first iteration initiallising all of the functionality
```

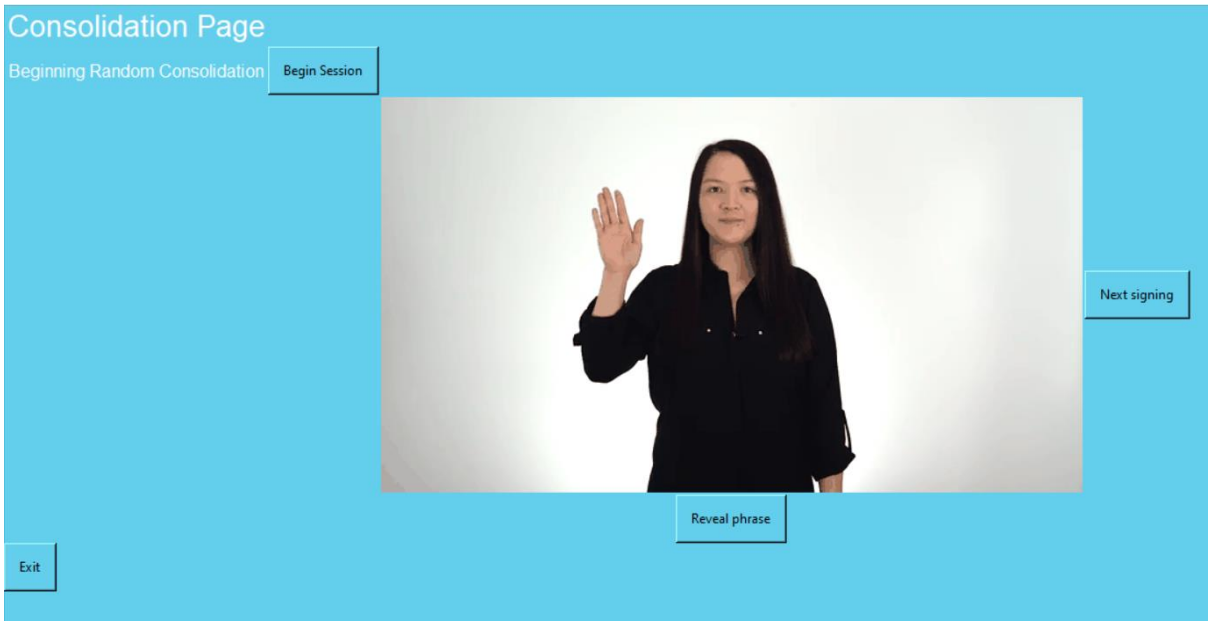
```
def iterate_consolidation_output():
    consolidation_gif_output.image = None
    consolidation_phrase_text.hide()
    consolidation_reveal_desc.hide()
    if len(consolidation_list)>1:
        consolidation_iterate_button.show()
        begin_practise_button.show()
    else:
        consolidation_iterate_button.hide()
        consolidation_finish_text.show()
    try:
        if not os.path.exists(consolidation_list[0]["Phrase"]+".gif"):
            #if video needs to be downloaded
            output_url = consolidation_list[0]["URL"]
            print(output_url)
            ytd = YouTube(output_url).streams.first().download(filename=consolidation_list[0]["Phrase"])
            print("Video downloaded to local directory")
            file_name = consolidation_list[0]["Phrase"] + ".mp4"
            clip = (VideoFileClip(file_name))#creates clip instance
            file_name = file_name.replace(".mp4",".gif")#changes the file type on string
            clip.write_gif(file_name)#converts .mp4 video to .gif
            consolidation_gif_output.image = file_name#outputs first value in list
            consolidation_gif_output.show()
            consolidation_reveal_desc.show()
            consolidation_phrase_text.value = "The Phrase is: " + consolidation_list[0]["Phrase"]
            del consolidation_list[0]#gets rid of the first value in list
        else:
            file_name = consolidation_list[0]["Phrase"]+".gif"
            consolidation_gif_output.image = file_name#outputs first value in list
            consolidation_gif_output.show()
            consolidation_reveal_desc.show()
            consolidation_phrase_text.value = "The Phrase is: " + consolidation_list[0]["Phrase"]
            del consolidation_list[0]#gets rid of the first value in list
    except:
        pass
```

Description:

This piece of code shows the procedures used to initialise the NON-tailored consolidation functionality (consolidation for a user that is not signed in – doesn't have an account – no data is saved). This works by creating a random list of phrases with the URL presenting the signing to the user. Once presented the user is allowed to reveal the description and confirm whether they interpreted the signing correctly. Within the GUI the user has the ability to initialise the consolidation program, iterate through the phrase and show the description for each phrase.

Testing for Creating Tailored Learning - Part 1– Iteration 1

Output:



This shows the signing presented to the user as a demonstration. The objective of this is to allow the user to see the phrase and copy the gesture. Once the user is ready, the program will present the description. This allows the user to attempt to interpret the signing before knowing what it means.

Creating Tailored Learning - Part 1– Iteration 2

```
def tailored_predict_desc():
    """This function is used when the user wants to see the actions/.gif
    and wants to guess the phrase/description.
    This functtion initialises this specific tailored learning"""
    tailored_phrase_then_desc.hide()
    tailored_desc_then_phrase.hide()
    tailored_consolidation_options.hide()
    global predict_desc
    predict_desc = True
    #this dictates to other functions which type of tailred learning is being done
    global tailored_list
    del tailored_list[:]#this is the list for practised phrases (queue)
    global template_list
    del template_list[:]#this is a template used to set the queue of phrases
    template_list = create_template_list()
    for value in template_list:
        #the queue of phrases with tailored repeats is made from the template
        #phrases that the user is less confident in have more repeats
        for i in range(5-int(value["Confidence"])):
            tailored_list.append(value)
    shuffle(tailored_list)#queue is shuffled
    tailored_consolidation()#starting the consolidation

def create_template_list():
    global user_object
    #checks whether the user has done tailored learning
    #if yes the user already has a list of phrases to use
    if user_object.started_TL == "Yes":
        temp_list = []
        for practised_phrase in user_object.learning_path:
            #learning path is the attribute of the obeject
            # that contains the user history
            if practised_phrase["Confidence"] != 5:
                #the number 5 dictates the number of points needed
                #to achieve "mastery"
                temp_list.append(practised_phrase)
                #only non mastered phrases are added
                #to be learnt in continuation
    else:
        #user needs a new set of phrases to learn
        temp_list = create_random_practice_list()
        for value in temp_list:
            value.update({"Confidence":0})
        user_object.started_TL = "Yes"
        user_object.learning_path = []
        for i in range(len(temp_list)):
            user_object.learning_path.append(temp_list[i])
    return temp_list
```

Description/Justification:

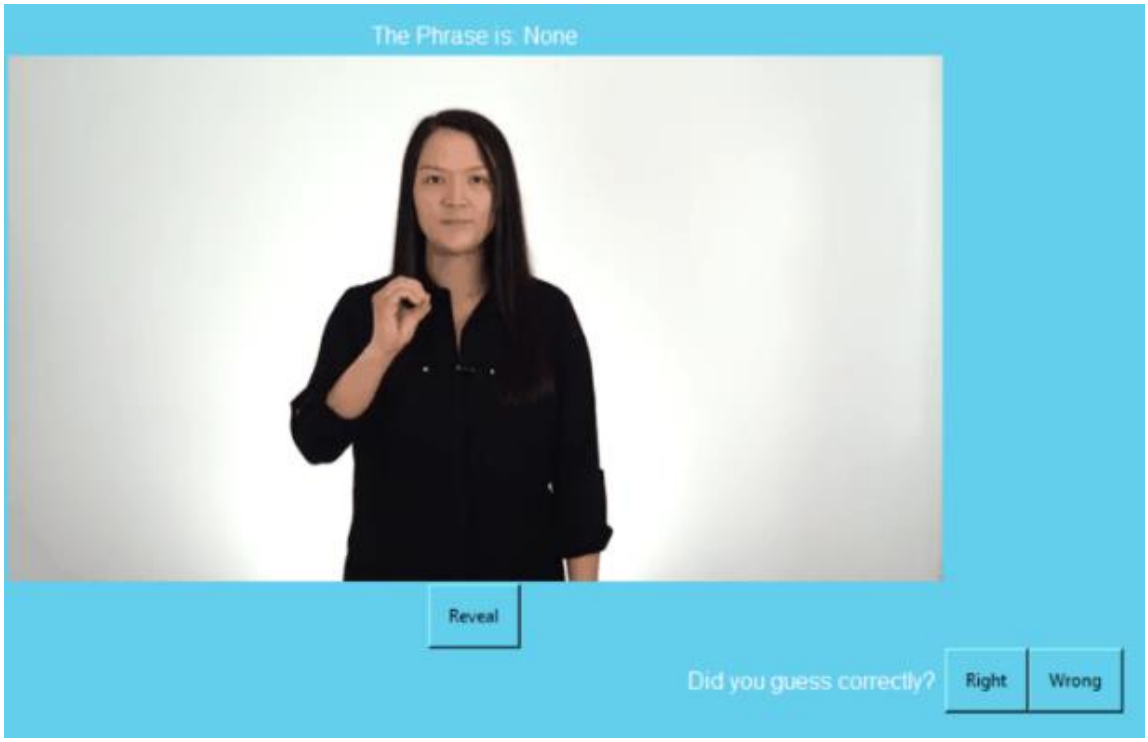
These pieces of code show the functions called when initiating the tailored learning for the user. The type of tailored learning here is from action/gif to description. This means that the user will be shown the gif of the signing and can guess the description of what it would translate to. Using the reveal button, the user would then be able to see the description and confirm/deny whether they guessed correctly. This data will then be stored.

The second function shows the creation of the template list. This list contains 10 different phrases which are currently being learnt by the user. This list then creates the queue of phrases testing the user with the respective frequency dependant on the user's previous confidence. Higher confidence means they occur less frequently.

The tailored_consolidation function called at the end of the first function would then display the .gif using the same function as the one seen in the previous iteration – "iterate_consolidation_output" (Creating Tailored Learning - Part 1– Iteration 1)

Testing for Creating Tailored Learning - Part 1– Iteration 2

Output:


The Phrase is: None
Reveal
Did you guess correctly?
Right
Wrong

Creating Tailored Learning - Part 1– Iteration 3

```
def correct_tailored_consolidation():
    global template_list
    global tailored_list
    global user_object
    user_object.total_right += 1
    for i in range(len(template_list)):
        if template_list[i]["Phrase"] == tailored_list[0]["Phrase"]:
            template_list[i]["Confidence"] += 1
            #increases confidence in template list for specific phrase that user got right
            if int(template_list[i]["Confidence"]) == 5:#if user has "mastered" the phrase
                removed = False
                while not removed:
                    #loops through and removes all other of that phrase from tailored_list (queue)
                    edited = False
                    for i in range(len(tailored_list)):
                        if tailored_list[i]["Phrase"] == template_list[i]["Phrase"]:
                            edited=True
                            del tailored_list[i]
                            break
                    if not edited:
                        removed = True
                temp_var = template_list[i]
                user_object.learning_path.append(temp_var)
                #appends mastered value to the user object to be stored
                del template_list[i]
                #deletes the value from the template list

                for k in range(len(total_vids)):#loops through the the video database
                    random_phrase = total_vids[k]
                    for complete_phrase in user_object.learning_path:
                        #loops through the users consolidation history
                        if complete_phrase["Phrase"] == random_phrase["Phrase"]:
                            #checks if the phrases are the same
                            if k == len(total_vids)-1:#if its the last phrase
                                random_phrase = choice(total_vids)
                                #uses a random phrase for the next phrase
                        else:
                            break#uses the next available phrase
                        break
                random_phrase.update({"Confidence":0})
                #inserts confidence key:value into the dictionary
                template_list.append(random_phrase)
                #adds the new phrase to be learnt to the template list
                user_object.learning_path.append(random_phrase)
                #adds the new phrase to be learnt to the user's history
                for i in range(5):
                    tailored_list.append(random_phrase)
                    #adds the new phrase to the practise list - to be practised 5 times
                shuffle(tailored_list)
                #shuffles the list so the same phrases aren't adjacent
            break
    tailored_feedback_text.show()
    tailored_correct_button.hide()
    tailored_incorrect_button.hide()
    tailored_feedback_text.hide()
    tailored_iteration_button.show()
    tailored_reveal.hide()
```

Description/Justification:

This piece of code is called when the user dictates that they guessed correctly. It increases the user's confidence rating found in the template. If a phrase is mastered (confidence=5) then the phrase is taken out of both the template and the queue (tailored_list) and appended to the user history (user.learning_path). A new phrase is then added to both the template and tailored lists.

Testing for Creating Tailored Learning - Part 1– Iteration 3

Output:

The GUI gets stuck on an output.

See testing and remedial actions.

Creating Tailored Learning - Part 1– Iteration 4

```
def correct_tailored_consolidation():
    global template_list
    global tailored_list
    global user_object
    user_object.total_right += 1
    for i in range(len(template_list)):
        if template_list[i]["Phrase"] == tailored_list[0]["Phrase"]:
            template_list[i]["Confidence"] += 1
            #increases confidence in template list for specific phrase that user got right
            if int(template_list[i]["Confidence"]) == 5:#if user has "mastered" the phrase
                removed = False
                while not removed:
                    #loops through and removes all other of that phrase from tailored_list (queue)
                    edited = False
                    for i in range(len(tailored_list)):
                        if tailored_list[i]["Phrase"] == template_list[i]["Phrase"]:
                            edited=True
                            del tailored_list[i]
                            break
                    if not edited:
                        removed = True
                temp_var = template_list[i]
                user_object.learning_path.append(temp_var)
                #appends mastered value to the user object to be stored
                del template_list[i]
                #deletes the value from the template list

                for k in range(len(total_vids)):#loops through the the video database
                    random_phrase = total_vids[k]
                    for complete_phrase in user_object.learning_path:
                        #loops through the users consolidation history
                        if complete_phrase["Phrase"] == random_phrase["Phrase"]:
                            #checks if the phrases are the same
                            if k == len(total_vids)-1:#if its the last phrase
                                random_phrase = choice(total_vids)
                                #uses a random phrase for the next phrase
                        else:
                            break#uses the next available phrase
                        break
                random_phrase.update({"Confidence":0})
                #inserts confidence key:value into the dictionary
                template_list.append(random_phrase)
                #adds the new phrase to be learnt to the template list
                user_object.learning_path.append(random_phrase)
                #adds the new phrase to be learnt to the user's history
                for i in range(5):
                    tailored_list.append(random_phrase)
                    #adds the new phrase to the practise list - to be practised 5 times
                shuffle(tailored_list)
                #shuffles the list so the same phrases aren't adjacent
            else:
                del tailored_list[0]
                #deletes the most phrase that was answered right from the queue
            break
    tailored_feedback_text.show()
    tailored_correct_button.hide()
    tailored_incorrect_button.hide()
    tailored_feedback_text.hide()
    tailored_iteration_button.show()
    tailored_reveal.hide()
```

Description/Justification:

The program corrects the problem where the GUI became stuck on a single phrase and would not iterate. This function iterates to the next phrase by deleting the first value in the tailored list (queue).

Testing for Creating Tailored Learning - Part 1– Iteration 4

Output:

```
    if tailored_list[i]["Phrase"] == template_list[i]["Phrase"]:
IndexError: list index out of range
```

When reaching a confidence value of 5 the program throws an error.See testing and remedial actions.

Creating Tailored Learning - Part 1– Iteration 5

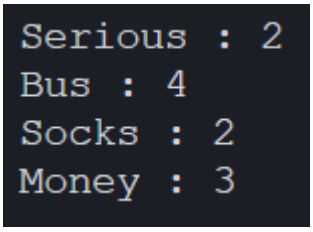
```
def correct_tailored_consolidation():
    global template_list
    global tailored_list
    for i in range(len(template_list)):
        if template_list[i]["Phrase"] == tailored_list[0]["Phrase"]:
            template_list[i]["Confidence"] += 1
            #increases confidence in template list for specific phrase that user got right
            if int(template_list[i]["Confidence"]) == 5:#if user has "mastered" the phrase
                removed = False
                while not removed:
                    #loops through and removes all other of that phrase from tailored_list (queue)
                    edited = False
                    for j in range(len(tailored_list)):
                        if tailored_list[j]["Phrase"] == template_list[i]["Phrase"]:
                            edited=True
                            del tailored_list[j]
                            break
                    if not edited:
                        removed = True
                temp_var = template_list[i]
                user_object.learning_path.append(temp_var)
                #appends mastered value to the user object to be stored
                del template_list[i]
                #deletes the value from the template list

                for k in range(len(total_vids)):#loops through the the video database
                    random_phrase = total_vids[k]
                    for complete_phrase in user_object.learning_path:
                        #loops through the users consolidation history
                        if complete_phrase["Phrase"] == random_phrase["Phrase"]:
                            #checks if the phrases are the same
                            if k == len(total_vids)-1:#if its the last phrase
                                random_phrase = choice(total_vids)
                                #uses a random phrase for the next phrase
                        else:
                            break#uses the next available phrase
                    break
                random_phrase.update({"Confidence":0})
                #inserts confidence key:value into the dictionary
                template_list.append(random_phrase)
                #adds the new phrase to be learnt to the template list
                user_object.learning_path.append(random_phrase)
                #adds the new phrase to be learnt to the user's history
                for i in range(5):
                    tailored_list.append(random_phrase)
                    #adds the new phrase to the practise list - to be practised 5 times
                shuffle(tailored_list)
                #shuffles the list so the same phrases aren't adjacent
            else:
                del tailored_list[0]
                #deletes the most phrase that was answered right from the queue
            break
    tailored_feedback_text.show()
    tailored_correct_button.hide()
    tailored_incorrect_button.hide()
    tailored_feedback_text.hide()
    tailored_iteration_button.show()
    tailored_reveal.hide()
```

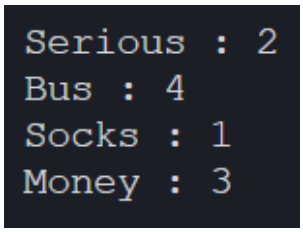
Description/Justification:

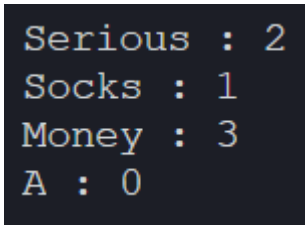
This piece of code corrects the problem where the index of a list would go out of range when a phrase attempted to be moved from the template list due to its confidence value of 5.

Testing for Creating Tailored Learning - Part 1– Iteration 5

Output:







These pictures show the confidence value for each phrase changes. As the confidence value of Bus becomes 5 it is removed from the template list and replaced by another phrase (confidence is 0 because it is new).

Creating Tailored Learning - Part 1– Iteration 6

```
def incorrect_tailored_consolidation():
    global template_list
    global tailored_list
    for value in template_list:
        if value["Phrase"] == tailored_list[0]["Phrase"] and value["Confidence"]!=0:
            #finds phrase in template list and only adjusts if confidence >0
            value["Confidence"]-= 1#adjusts the confidence value for the given phrase
            tailored_list.append(value)
            #adds same phrase to the back twice
            #due to decrease in confidence
            break
    del tailored_list[0]#allows iteration to next value
    tailored_list.append(value)
    shuffle(tailored_list)
    tailored_feedback_text.show()
    tailored_correct_button.hide()
    tailored_incorrect_button.hide()
    tailored_feedback_text.hide()
    tailored_iteration_button.show()
    tailored_reveal.hide()
```

Description/Justification:

Similar to the correct_tailored_consolidation function in the previous iteration, this function edits the template list by decreasing the confidence value of a given phrase. This function does not decrease the value if it is already 0 and instead adds the phrase to the back of the queue to be retried when the user gets to it. If the value is greater than 0, it is decreased by 1 and again added to the back of the queue. Since the value is decreased the frequency of phrases increased by adding the same phrase again to be practised more often. This allows user to have a fully tailored learning experience with a light version of spaced repetition.

Testing for Creating Tailored Learning - Part 1– Iteration 6

Output:

```
Who : 2
Waitrose : 2
[{'Phrase': 'Waitrose', 'URL': 'https://www.youtube.com/watch?v=N3JD9FF6P64', 'Confidence': 2}, {'Phrase':
 'Waitrose', 'URL': 'https://www.youtube.com/watch?v=N3JD9FF6P64', 'Confidence': 2}, {'Phrase': 'Waitrose'
, 'URL': 'https://www.youtube.com/watch?v=N3JD9FF6P64', 'Confidence': 2}, {'Phrase': 'Who', 'URL': 'https:
//www.youtube.com/watch?v=x_Svt0onT7A', 'Confidence': 2}, {'Phrase': 'Who', 'URL': 'https://www.youtube.co
m/watch?v=x_Svt0onT7A', 'Confidence': 2}, {'Phrase': 'Who', 'URL': 'https://www.youtube.com/watch?v=x_Svt0
onT7A', 'Confidence': 2}]

Who : 2
Waitrose : 1
[{'Phrase': 'Waitrose', 'URL': 'https://www.youtube.com/watch?v=N3JD9FF6P64', 'Confidence': 1}, {'Phrase':
 'Who', 'URL': 'https://www.youtube.com/watch?v=x_Svt0onT7A', 'Confidence': 2}, {'Phrase': 'Who', 'URL': '
https://www.youtube.com/watch?v=x_Svt0onT7A', 'Confidence': 2},{'Phrase': 'Waitrose', 'URL': 'https://www
.youtube.com/watch?v=N3JD9FF6P64', 'Confidence': 1}, {'Phrase': 'Waitrose', 'URL': 'https://www.youtube.co
m/watch?v=N3JD9FF6P64', 'Confidence': 1},{'Phrase': 'Who', 'URL': 'https://www.youtube.com/watch?v=x_Svt0
onT7A', 'Confidence': 2}, {'Phrase': 'Waitrose', 'URL': 'https://www.youtube.com/watch?v=N3JD9FF6P64', 'Co
nfidence': 1}]
```

Output shows the phrase next to its confidence value initially. It then shows the confidence of “Waitrose” decreasing and so (shown by the highlighting and numbering) the number of iterations of the given phrase increases from 3 to 4.

Creating Tailored Learning - Part 1– Iteration 7

```
def tailored_predict_phrase():
    """This function is used when the user wants to see the phrase/
    description and wants to guess the actions/.gif"""
    tailored_phrase_then_desc.hide()
    tailored_desc_then_phrase.hide()
    tailored_consolidation_options.hide()
    global predict_desc
    predict_desc = False
    #this dictates to other functions which type of tailred learning is being done
    global tailored_list
    del tailored_list[:]#this is the list for practised phrases (queue)
    global template_list
    del template_list[:]#this is a template used to set the queue of phrases
    template_list = create_template_list()
    for value in template_list:
        #the queue of phrases with tailored repeats is made from the template
        #phrases that the user is less confident in have more repeats
        for i in range(5-int(value["Confidence"])):
            tailored_list.append(value)
    shuffle(tailored_list)#queue is shuffled
    tailored_consolidation()#starting the consolidation
```

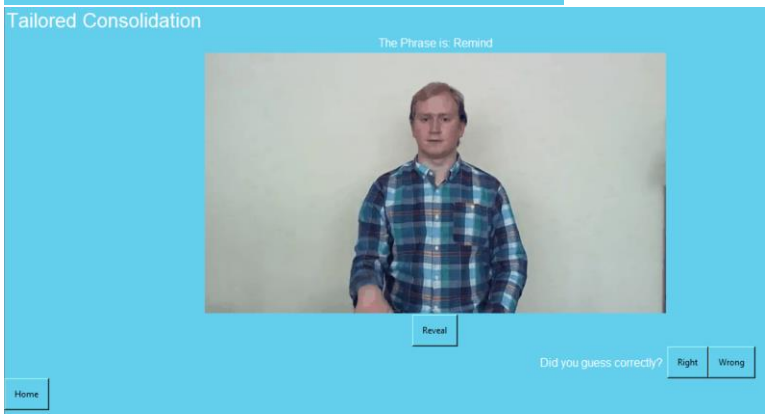
Description/Justification:

This piece of code is called when the user picks the “predict the phrase” option where they must use the description of the signing to guess the actions. This function is very similar to the tailored_predict_desc function used for the second learning option. The main difference is that the global predict_desc is set as false. This is used for future functions that need to know which output to display first, the description or the gif. Having this identifier means that this function only needs to hold the major initialisation processes such as creating the template list with the rest of the differentiating functionality being integrated into the main processing functions.

Testing for Creating Tailored Learning - Part 1– Iteration 7

Output:

This shows the initial stage where the description is shown and the feedback page where the user can enter whether they guessed the signing action correctly.



Creating Setting – Iteration 1

```
def loggedin_to_settings():
    home_loggedin_page.hide()
    settings_page.show()
    adjusted_name = user_object.name.title()
    settings_name.value = "Name: " + str(adjusted_name)
    settings_username.value = "Username: " + str(user_object.username)
    settings_date_joined.value = "Date joined: " + str(user_object.date_joined)
    settings_email.value = "Email: " + str(user_object.email)
    settings_total_right.value = "Total right answers: " + str(user_object.total_right)
    settings_total_wrong.value = "Total wrong answers: " + str(user_object.total_wrong)
```

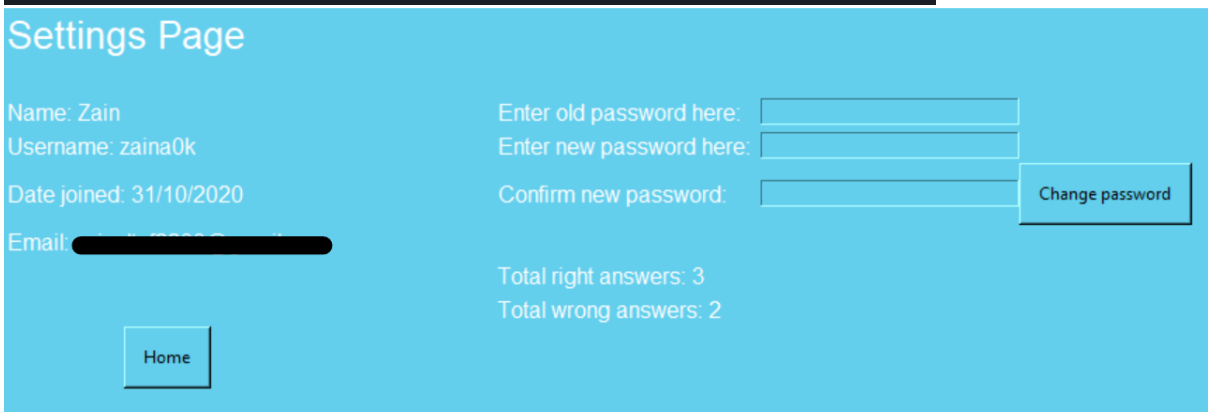
Description:

This piece of code uses the values from within the user object and displays them in the GUI for the user to see. This allows the user to understand what information is being stored about them and also allows them to see a rough idea of the progress they have made through the total right/wrong values.

Testing for Creating Setting – Iteration 1

Output:

```
user number: 2
name: zain
username: zaina0k
password: b033d770602994efa135c5248af300d81567ad5b59cec4bccbf15bcc
date joined: 31/10/2020
email: [redacted]
started tailored learning? Yes
current learning path:
total right: 3
total wrong: 2
```



Once again the personal email is blacked out. All the values are returned in the GUI in a well formatted manner.

Creating Setting – Iteration 2

```
def change_password():
    global total_users
    global user_object
    total_users = user_worksheet.get_all_records()
    password = settings_old_password_input.value
    init_password = str.encode(password)
    hashed_password = hashlib.sha224(init_password).hexdigest().strip("0")
    both_equal = False
    for i in range(len(total_users)):
        if total_users[i]["Password"] == hashed_password:
            if settings_new_password_input.value == settings_confirm_password_input.value:
                both_equal = True
                new_password = str.encode(settings_new_password_input.value)
                hashed_new_password = hashlib.sha224(new_password).hexdigest().strip("0")
                user_object.password = hashed_new_password
                updating_row()
                settings_old_password_input.value = ""
                settings_new_password_input.value = ""
                settings_confirm_password_input.value = ""
                settings_password_error.value = "Password changed successfully"

        elif i == len(total_users)-1:
            settings_password_error.value = "Please enter the correct original password"
    if not both_equal:
        settings_password_error.value = "Please enter the same new password"
```

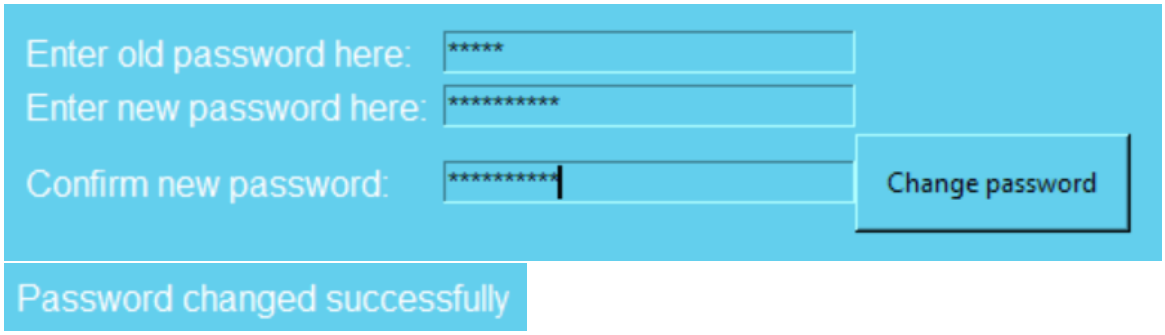
Description/Justification:

This piece of code is called when the user attempts to reset the password. In order to do this, they must know their old password and type in their new password in 2 separate boxes. This ensures that the user will not accidentally set their password with a mistyped string. Once confirmed with their details, the user's password is updated within the object in its encrypted form. The profile is then updated within the online database and ensures that the new password will be set for the next time the user attempts to login.

The function also returns specific error messages if the user has entered something incorrectly. This is useful since the textboxes are all starred out and therefore cannot be seen by the user.

Testing for Creating Setting – Iteration 2

Output:



3.3.1b Provide annotated evidence of prototype solutions justifying any decision made

To allow for better understanding of the prototypes the following steps will be taken:

- Descriptive identifiers
- Explanatory comments
- "spaced out code"
- Local inputs and outputs

Functional Chunk 1 - Searching specific phrases prototypes:

Back to contents page

Searching specific phrases prototype 1 – search conditioning

```
database_list = [["hello world",None,None],["good afternoon",None,None],["hello there",None,None]]
#database list format: [phrase, file path, visual data for recognition]


#input conditioning for search
user_input = "hello world!"
new_string = ""
for i in range(len(user_input)):
    #loops through the string and checks if every character is a letter
    if user_input[i].isalpha():
        new_string += user_input[i].lower()
        #only adds the letters to the final string

print(user_input)
print(new_string)
#this new phrase has no spaces like there will be in the database
```

This section of code shows the conditioning the raw user input will initially go through. This piece of code goes through each character entered by the user and isolating the alphabet characters from other special characters as well as numbers that cannot be searched. This new string can then be used to search for a specific phrase.

Output:

```
hello world!
helloworld
```

Searching specific phrases prototype 2 – Microphone input

```
import audiomath; audiomath.RequireAudiomathVersion( '1.12.0' )
import speech_recognition
from duck_typed_microphone import DuckTypedMicrophone

if __name__ == '__main__':#if program is being run internally
    import speech_recognition as sr
    r = sr.Recognizer()#initialises the recognizer by creating an object
    with DuckTypedMicrophone() as source:#uses the duck typed class as a microphone inputter
        print('\nSay something to the %s...' % source.__class__.__name__)#prints the name of the microphone class
        r.adjust_for_ambient_noise(source)#creates a tolerance for ambient noise in the audio source
        audio = r.listen(source)#converts the input from the microphone to audio data object
    print('Got it.')
    try:
        #performs a speech recognition on an audio data instance using Google Speech Recognition API
        print('\nUnderstood: "%s"\n' % r.recognize_google(audio))
    except Exception:
        #excepts unknown input that cannot be translated or any other errors such as no network connection
        print("sorry, unable to translate audio")
    if 0: # plot and/or play back captured audio
        s = audiomath.Sound(audio.get_wav_data(), fs=audio.sample_rate, nChannels=1)
        s.Play()
        s.Plot()
```

1

2

3

This prototype was developed using the Speech Recognition python module. This external module uses a variety of functions and algorithms to allow the user to use the microphone as an input.

The speech recognition module has built in modules that fully satisfy the amount of functionality I wished to gain from using it, however, while testing I found that the microphone module had bugs. To fix this I searched for an alternate method and came across a “duck typed” microphone module that could be used instead and offer the same functionality.

This substitute module was found on Stack overflow : (https://stackoverflow.com/questions/55984129/attributeerror-could-not-find-pyaudio-check-installation-cant-use-speech-re). Last accessed: 01/09/20.

Sections:

1. Imports for the speech recognition module as well as supporting modules
2. Microphone input as an audio file. Audio file is then converted into audio data
3. Google speech recognition API is used to process audio data and outputs processed text

This prototype was developed using the Speech Recognition python module. This external module uses a variety of functions and algorithms to allow the user to use the microphone as an input.

Output:

```
Say something to the DuckTypedMicrophone...
Got it.

Understood: "hello world"
```

Searching specific phrases prototype 2b – Duck typed microphone code

```
import audiomath; audiomath.RequireAudiomathVersion( '1.12.0' )
import speech_recognition

class DuckTypedMicrophone( speech_recognition.AudioSource ): # descent from AudioSource is required purely to pass an assertion in Recognizer.listen()
    def __init__( self, device=None, chunkSeconds=1024/44100.0 ):  # 1024 samples at 44100 Hz is about 23 ms
        self.recorder = None
        self.device = device
        self.chunkSeconds = chunkSeconds
    def __enter__( self ):
        self.nSamplesRead = 0
        self.recorder = audiomath.Recorder( audiomath.Sound( 5, nChannels=1 ), loop=True, device=self.device )
        # Attributes required by Recognizer.listen():
        self.CHUNK = audiomath.SecondsToSamples( self.chunkSeconds, self.recorder.fs, int )
        self.SAMPLE_RATE = int( self.recorder.fs )
        self.SAMPLE_WIDTH = self.recorder.sound.nbytes
        return self
    def __exit__( self, *blx ):
        self.recorder.Stop()
        self.recorder = None
    def read( self, nSamples ):
        sampleArray = self.recorder.ReadSamples( self.nSamplesRead, nSamples )
        self.nSamplesRead += nSamples
        return self.recorder.sound.dat2str( sampleArray )
    @property
    def stream( self ): # attribute must be present to pass an assertion in Recognizer.listen(), and its value must have a .read() method
        return self if self.recorder else None

# source must be an instance of a speech_recognition.AudioSource subclass
# source.stream must be non-None while the source is active
# source.CHUNK must be the (integer) number of samples per chunk
# source.SAMPLE_RATE must be the sampling rate
# source.SAMPLE_WIDTH must be the number of bytes per sample
# source.stream.read(numberOfSamples) must return raw single-channel audio data
```

Searching specific phrases prototype 3 –Top 20 recently searched phrases

```
sample_list = ["Lorem","ipsum", "dolor", "sit", "amet", "consectetur", "adipiscing", "elit", "eiusmod"]
#sample data
menu = True
queue = []#empty list to store the most recent strings
counter = 0#counter is used to identify an index of the sample list

while menu:
    if counter == len(sample_list):
        #when reaching the end of the list the counter resets
        counter = 0
        input()

    if len(queue) >= 4:#number 4 acts as the limit of the queue
        #if the queue is "full"
        queue.pop(-1)#gets rid of the last value in the list
        queue.insert(0,sample_list[counter])#adds new value to the front of the list
    else:
        queue.insert(0,sample_list[counter])#adds the new value to the front of the list

    counter +=1#next index
    print(queue)#output
```

This prototype showcases a queue where values are added by First In First Out rule. This prototype shows the concept of iteratively adding to an array which can then be displayed while continuously keeping the length less than a specified amount. In this case the specified amount is 4 but will be changed to 20 for the final program. The prototype also adds from the front and outputs from the back of the list. This is so that I can display the top 20 phrases in the natural order of the list with the most recent phrase being the first in line.

Output:

```
['Lorem']
['ipsum', 'Lorem']
['dolor', 'ipsum', 'Lorem']
['sit', 'dolor', 'ipsum', 'Lorem']
['amet', 'sit', 'dolor', 'ipsum']
['consectetur', 'amet', 'sit', 'dolor']
['adipiscing', 'consectetur', 'amet', 'sit']
['elit', 'adipiscing', 'consectetur', 'amet']
['eiusmod', 'elit', 'adipiscing', 'consectetur']
```

Functional Chunk 2 - Searching Through Database – Prototypes:

Back to Contents Page

Simple string checker using ==

```
database_list = [["hello world",None,None],["good afternoon",None,None],["hello there",None,None]]
#sample data for testing

user_input = "hello world"
data_result = None

for phrase in database_list:#iterated through the database list
    if phrase[0] == user_input:
        data_result = phrase
    #returns the list with the data for that specfic phrase
print(data_result)
```

Description:

This piece of code shows the process of searching through the database. It does this by comparing the user_input with the string value in the database. If it finds the right string it returns the list belonging to that string.

Output:

```
user input: hello world
database search result: ['hello world', None, None]
```

Comments:

The problem with this method is the use of == as a comparing method. It quickly become ineffective when little changes are made to the user input string. E.G. if the user entered an extra character or one letter uppercase the string would not be found and nothing would be returned.

```
user input: helloworld
database search result: None
```

```
user input: hello world3
database search result: None
```

Using conditioned strings to search

```
database_list = [["hello world",None,None],["good afternoon",None,None],["hello there",None,None]]
#sample data for testing

user_input = "helloworld134!"
new_string = ""
data_result = None

user_input = user_input.lower()
for i in range(len(user_input)):#iterates through user input
    if user_input[i] == " ":
        new_string+=user_input[i]#adds spaces to final string
    elif user_input[i].isalpha():
        new_string+=user_input[i]#adds letter characters

for phrase in database_list:#iterated through the database list
    data_string = ""
    raw_string = phrase[0]
    for i in range(len(raw_string)):#iterates through database phrase
        if raw_string[i] == " ":
            data_string += raw_string[i]#adds spaces to final string
        elif raw_string[i].isalpha():
            data_string += raw_string[i]#adds letter characters
    if data_string==new_string:
        data_result = phrase#checks the conditioned phrases against each other
        break
    #returns the list with the data for that specfic phrase

print()
print("user input: " + str(user_input))
print("database search result: " + str(data_result))
print()
```

Description:

This program differs from the “simple string checker using ==” as it includes some light input conditioning so that unwanted characters in the string are removed before checking against the database for a match. This allows the user to make some mistakes in terms of the input if they input numbers or special characters.

Output:

```
user input: helloworld134!
database search result: None
```

```
user input: hello world134!
database search result: ['hello world', None, None]
```

Comments:

The program works as expected and provides some extra “cushioning” for the user’s input before searching the database, however, it still does not provide any support to the user if they misspell a word or if they don’t place a space correctly.

Iterating through each string

```
database_list= [["good morning",None,None],["good evening",None,None],["rain",None,None],["raining",None,None]]
user_input = "rains78932"
new_string = ""
#input validation - removes numbers/special characters
user_input = user_input.lower()
for i in range(len(user_input)):#iterates through user input
    if user_input[i] == " ":
        new_string+=user_input[i]#adds spaces to final string
    elif user_input[i].isalpha():
        new_string+=user_input[i]#adds letter characters
user_input = new_string#sets user_input as the validated string

def phrase_finder(database,user_input,index):
    """iterates though each phrase in the database list letter by letter
    and checks agains the same index letter of the user input.
    returns the values that correspond"""
    temp_data_list = []
    for i in range(len(database)):#iterates through all the values of database
        data_string = database[i][0]#actual string value of each iteration
        if index<len(data_string):#in case the user input is longer than the the phrase
            phrase_letter = data_string[index]
            if phrase_letter == user_input[index]:
                temp_data_list.append(database[i])#adds corresponding phrases to be returned
    return temp_data_list

for i in range(len(user_input)):#loops through each character in the entered string
    holder_list  = phrase_finder(database_list,user_input,i)
    if len(holder_list) == 0:
        pass
    else:
        database_list = holder_list
    #updates database_list each iteration so that only the currently corresponding
    #values are added means that each letter iteration for the user input will be
    #searched against an increasingly condensed list of values
    #making it more efficient to find

print("values found from database: " + str(database_list))
```

Description:

Alongside the input validation developed in the previous prototype this prototype includes a letter by letter searching algorithm. This works by searching through the database to find phrases with the corresponding first letter and adding those corresponding phrases to a list to be returned. This function is carried out in a loop so that once the first letter has been checked and the list containing all phrases with the same first letter has been formed the list can be re-iterated through using the second letter to find phrases with the same second letter. This continues for the length of the phrase until all the values are found. Note: making the new database_list each time for the "new batch" of corresponding phrases makes sense as it shortens the length of the list so that the program has to iterate through less values and therefore makes it faster.

Output:

```
user input: rains78932
Values found from database: [['rain', None, None], ['raining', None, None]]
```

Comments:

The program works as expected and returns values that have the same first section of the user input. The main advantage using this process is the fact that the list prints out the last correlated value found in the list. This means that if mistakes are made in the string towards the end, it will still output the values that had the same beginning.

```
user input: graining78932
Values found from database: [['good morning', None, None], ['good evening', None, None]]
```

This screenshot shows that if the first part of the string is entered incorrectly the output is completely skewed.

3.3.2 Testing to inform development

3.3.2a Provide annotated evidence for testing at each stage justifying the reason for the test

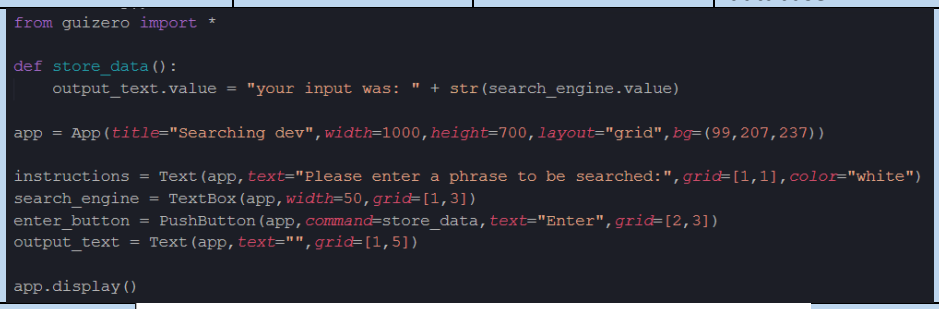
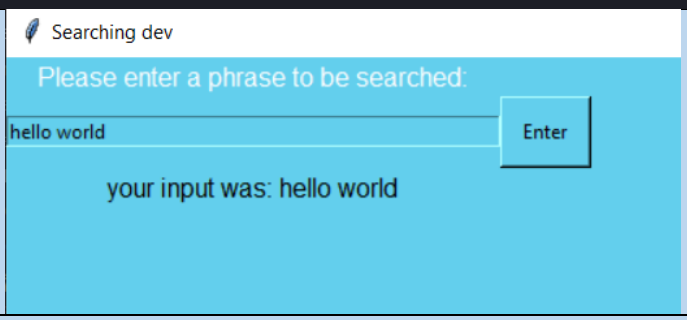
Functional Chunk 1 – Searching Specific Phrases

Back to Contents Page

Inputting a phrase

Part 1 - Iteration 1:

Inputting a Phrase – Part 1 - Iteration 1 Code

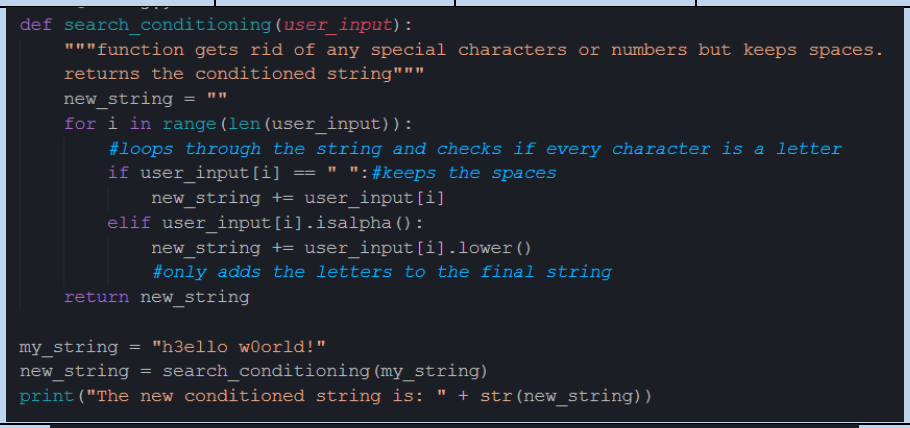
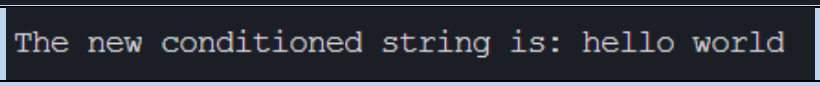
Test	Description	Test Data	Justification	Expected Output
Event Test 1	Create a basic graphical user interface that allows the user to input text as data.	"hello world"	Inputting data through a textbox will be one of the main sources of user input.	The string will be taken in by the program and stored to be searched through a database.
Test Code				
Result Screenshot				
Comments	Program works as expected			

```
from guizero import *

def store_data():
    output_text.value = "your input was: " + str(search_engine.value)

app = App(title="Searching dev",width=1000,height=700,layout="grid",bg=(99,207,237))

instructions = Text(app,text="Please enter a phrase to be searched:",grid=[1,1],color="white")
search_engine = TextBox(app,width=50,grid=[1,3])
enter_button = PushButton(app,command=store_data,text="Enter",grid=[2,3])
output_text = Text(app,text="",grid=[1,5])

app.display()
```

Input Conditioning

Iteration 1:

Input Conditioning – Iteration 1 Code

Test	Description	Test Data	Justification	Expected Output
Event Test 1	Function that takes in a flawed input – an input with numbers or special characters – and returns a valid output with only alphabet characters.	h3ello w0orld!	To allow the program to run without errors and to maximise the rate of quality outputs.	The output string should have no numbers or special characters – hello world.
Test Code				
Result Screenshot				
Comments	Program works as expected			

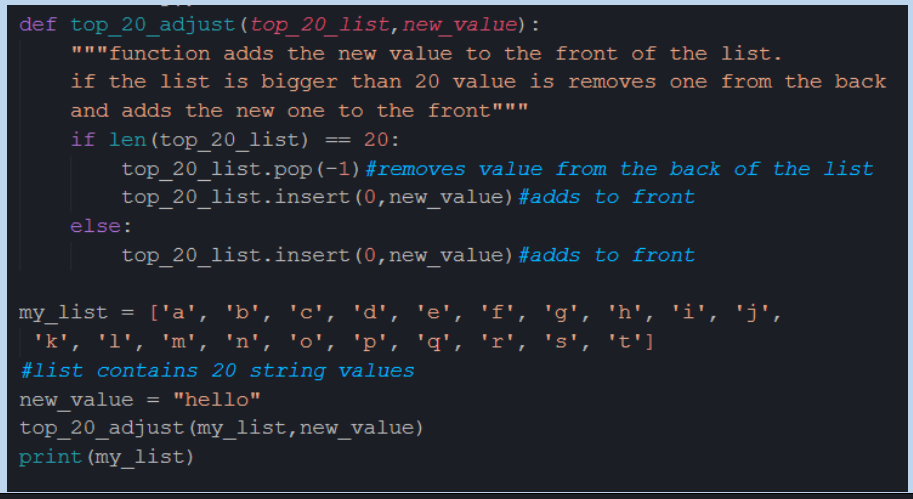
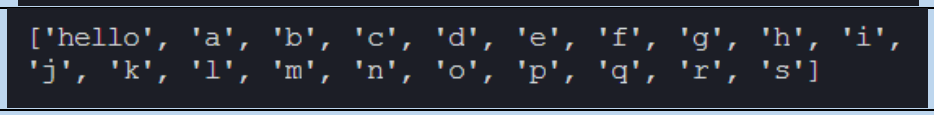
```
def search_conditioning(user_input):
    """function gets rid of any special characters or numbers but keeps spaces.
    returns the conditioned string"""
    new_string = ""
    for i in range(len(user_input)):
        #loops through the string and checks if every character is a letter
        if user_input[i] == " ":#keeps the spaces
            new_string += user_input[i]
        elif user_input[i].isalpha():
            new_string += user_input[i].lower()
            #only adds the letters to the final string
    return new_string

my_string = "h3ello w0orld!"
new_string = search_conditioning(my_string)
print("The new conditioned string is: " + str(new_string))
```

```
The new conditioned string is: hello world
```

Top 20 Recently Searched Phrases

Part 1 - Iteration 1:

Top 20 Recently Searched Phrases Code – Part 1 –Iteration 1

Test	Description	Test Data	Justification	Expected Output
Event Test 1	Function to maintain the size of the Top 20 list at 20 values. List would read with the most recent first.	List with 20 values already stored – use the function to add another value	The function needs to be able to manage the list when it reaches the point where it contains 20 value, but more values need to be added.	The list should be adjusted so that only the 20 most recent values are stored with the most recent first. Older values should be removed, and new values added once the list becomes full
Test Code				
Result Screenshot				
Comments	Program works as expected			

```
def top_20_adjust(top_20_list,new_value):
    """function adds the new value to the front of the list.
    if the list is bigger than 20 value is removes one from the back
    and adds the new one to the front"""
    if len(top_20_list) == 20:
        top_20_list.pop(-1)#removes value from the back of the list
        top_20_list.insert(0,new_value)#adds to front
    else:
        top_20_list.insert(0,new_value)#adds to front

my_list = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j',
 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't']
#list contains 20 string values
new_value = "hello"
top_20_adjust(my_list,new_value)
print(my_list)
```

```
['hello', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i',
'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's']
```

Part 2 – Iteration 1:

Top 20 Recently Searched Phrases code – Part 2 – Iteration 1

Test	Description	Test Data	Justification	Expected Output
Event Test 1	Program should process a list of varying size and return a 2d list with inner lists containing up to 4 values. Any extra values should be added to another internal list.	[“hello”,”this”,”is”, ”a”,”test”]	The string will already be split into its separate values and ready to be sorted. This test data is longer than 4 values and will showcase the program’s ability to sort the values.	[[“hello”,”this”,”is”, “a”],[“test”]]

Test Code

```
def top_20_format(my_list):
    """function formats the lists so that they can be more easily displayed.
    creates a 2d list with up to 4 values in each of the inner lists."""
    list_1 = []
    list_2 = []
    for i in range(len(my_list)):#loops through all the current top 20 values
        if len(list_1) == 4:#if there are 4 values in the holder list
            list_2.append([])#creates a new empty list
            for i in range(4):
                list_2[-1].append(list_1[i])#adds the 4 values to the empty list
            del list_1[:]#clears the holder list

        elif (i+1) == len(my_list):#if at the last index in list
            list_1.append(my_list[i])#adds the last value to holder list
            list_2.append([])#creates space for new values
            for i in range(len(list_1)):
                list_2[-1].append(list_1[i])#adds the rest of values to the empty list
        else:
            list_1.append(my_list[i])#adds value to holer list
```

Result Screenshot

```
[['hello']]
[['this', 'hello']]
[['is', 'this', 'hello']]
[['a', 'is', 'this', 'hello']]
[['test', 'a', 'is', 'this']]
```

Comments

Test failed: The result screenshot shows the printed output of the processed list after each iteration of adding an extra value to the top 20 list. The program successfully creates a 2d list but loses the “hello” value after the 5th value is added.

Additional testing - output

Test failed:

- In this test the inputted string is longer and so the code is run more times.
- Shows that the 5th value is not erased but instead missed when appending to the 2d list due to the fact after each iteration the top 20 2d list is remade.

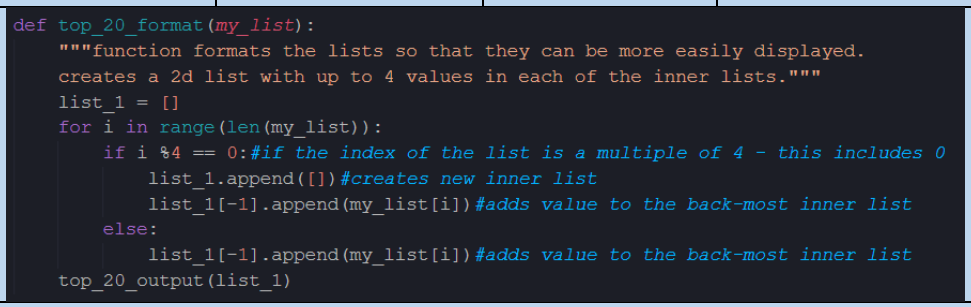
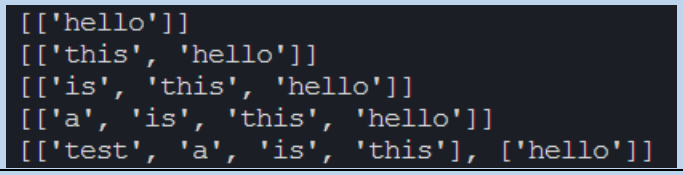
Test data: [“hello”, “this”, “is”, “a”, “longer”, “test”, “for”, “the”, “program”]

```
[['hello']]
[['this', 'hello']]
[['is', 'this', 'hello']]
[['a', 'is', 'this', 'hello']]
[['longer', 'a', 'is', 'this']]
[['test', 'longer', 'a', 'is'], ['hello']]
[['for', 'test', 'longer', 'a'], ['this', 'hello']]
[['the', 'for', 'test', 'longer'], ['is', 'this', 'hello']]
[['program', 'the', 'for', 'test'], ['a', 'is', 'this', 'hello']]
```

Remedial actions for Top 20 Recently Searched Phrases code – Part 2 – Iteration 1

Part 2 – Iteration 2:

Top 20 Recently Searched Phrases code – Part 2 – Iteration 2

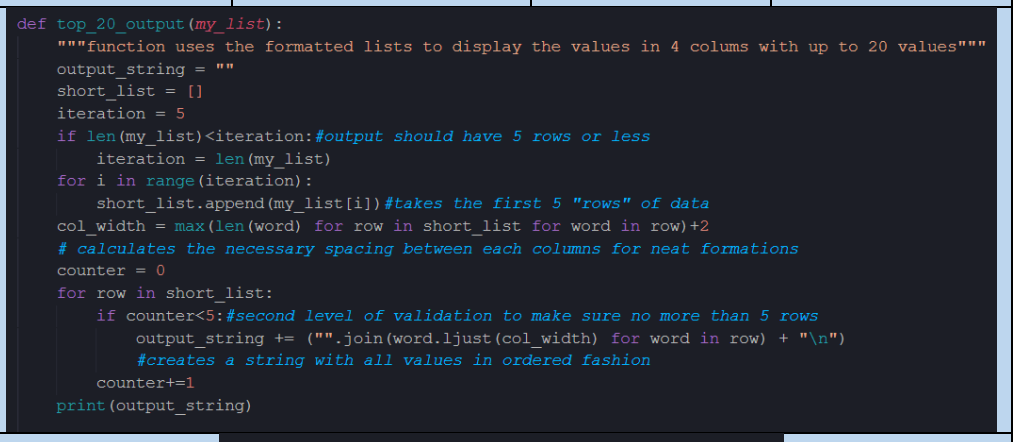
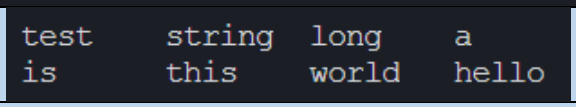
Test	Description	Test Data	Justification	Expected Output
Event Test 1	Program should process a list of varying size and return a 2d list with inner lists containing up to 4 values. Any extra values should be added to another internal list.	[“hello”,”this”,”is”, ”a”,”test”]	The string will already be split into its separate values and ready to be sorted. This test data is longer than 4 values and will showcase the program’s ability to sort the values.	[[“hello”,”this”,”is”, “a”],[“test”]]
Test Code				
Result Screenshot				
Comments	Program works as expected			

```
def top_20_format(my_list):
    """function formats the lists so that they can be more easily displayed.
    creates a 2d list with up to 4 values in each of the inner lists."""
    list_1 = []
    for i in range(len(my_list)):
        if i %4 == 0:#if the index of the list is a multiple of 4 - this includes 0
            list_1.append([])#creates new inner list
            list_1[-1].append(my_list[i])#adds value to the back-most inner list
        else:
            list_1[-1].append(my_list[i])#adds value to the back-most inner list
    top_20_output(list_1)
```

```
[['hello']]
[['this', 'hello']]
[['is', 'this', 'hello']]
[['a', 'is', 'this', 'hello']]
[['test', 'a', 'is', 'this'], ['hello']]
```

Part 3 – Iteration 1:

Top 20 Recently Searched Phrases code – Part 3 – Iteration 1

Test	Description	Test Data	Justification	Expected Output
Event Test 1	Program should process the normalised list and split the values into 4 separate columns with 5 rows. Each column should be spaced out.	[[“hello”,”world”,”this”, “is”],[”a”,”long”,”string”, “test”]]	The 4 columns and up to 5 rows allows for a compact form of information. It makes it easier to position in the GUI.	4 columns and 2 rows with the words in reverse order.
Test Code				
Result Screenshot				
Comments	Program works as expected			

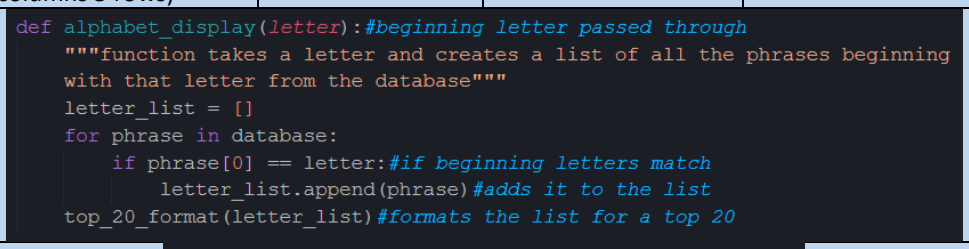
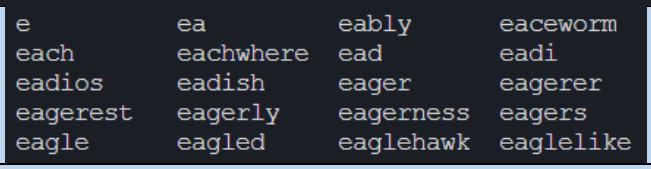
```
def top_20_output(my_list):
    """function uses the formatted lists to display the values in 4 colums with up to 20 values"""
    output_string = ""
    short_list = []
    iteration = 5
    if len(my_list)<iteration:#output should have 5 rows or less
        iteration = len(my_list)
    for i in range(iteration):
        short_list.append(my_list[i])#takes the first 5 "rows" of data
    col_width = max(len(word) for row in short_list for word in row)+2
    # calculates the necessary spacing between each columns for neat formations
    counter = 0
    for row in short_list:
        if counter<5:#second level of validation to make sure no more than 5 rows
            output_string += ("".join(word.ljust(col_width) for word in row) + "\n")
            #creates a string with all values in ordered fashion
        counter+=1
    print(output_string)
```

```
test    string  long    a
is      this    world   hello
```

Alphabet Line

Part 1 - Iteration 1:

Alphabet Line Code Part 1– Iteration 1

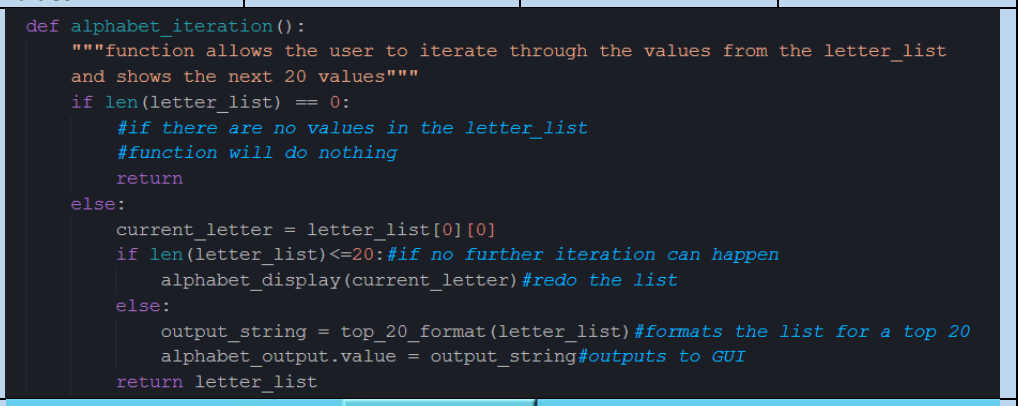
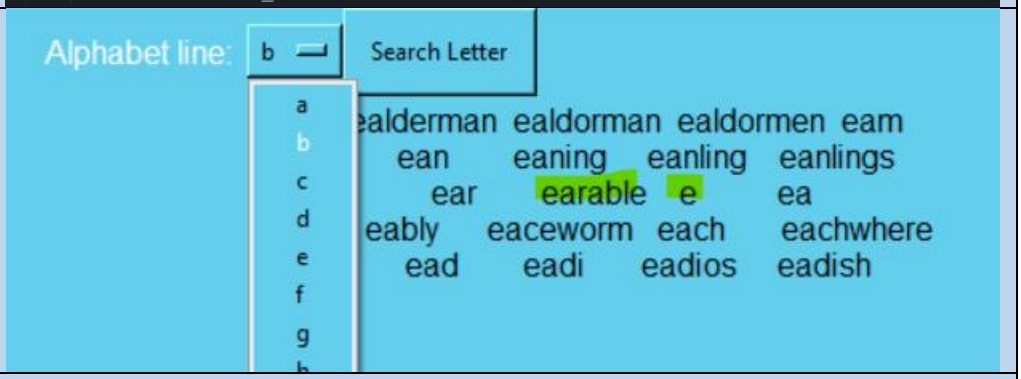
Test	Description	Test Data	Justification	Expected Output
Event Test 1	Program should allow user to pick a letter on an alphabet line. The program should then process the letter and find corresponding values from the word bank database and output them in the top 20 format (4 columns 5 rows)	"e"	The alphabet line acts an extra feature and allows another method of interactivity between the user and the program.	20 beginning values of the word bank database. All values should begin with the letter "e"
Test Code				
Result Screenshot				
Comments	Program works as expected			

```
def alphabet_display(letter):#beginning letter passed through
    """function takes a letter and creates a list of all the phrases beginning
    with that letter from the database"""
    letter_list = []
    for phrase in database:
        if phrase[0] == letter:#if beginning letters match
            letter_list.append(phrase)#adds it to the list
    top_20_format(letter_list)#formats the list for a top 20
```

```
e           ea          eably       eaceworm
each        eachwhere   ead         eadi
eadios      eadish      eager       eagerer
eagerest    eagerly     eagerness   eagers
eagle       eagled      eaglehawk   eaglelike
```

Part 2 – Iteration 1

Alphabet Line Code - Part 2 – Iteration 2

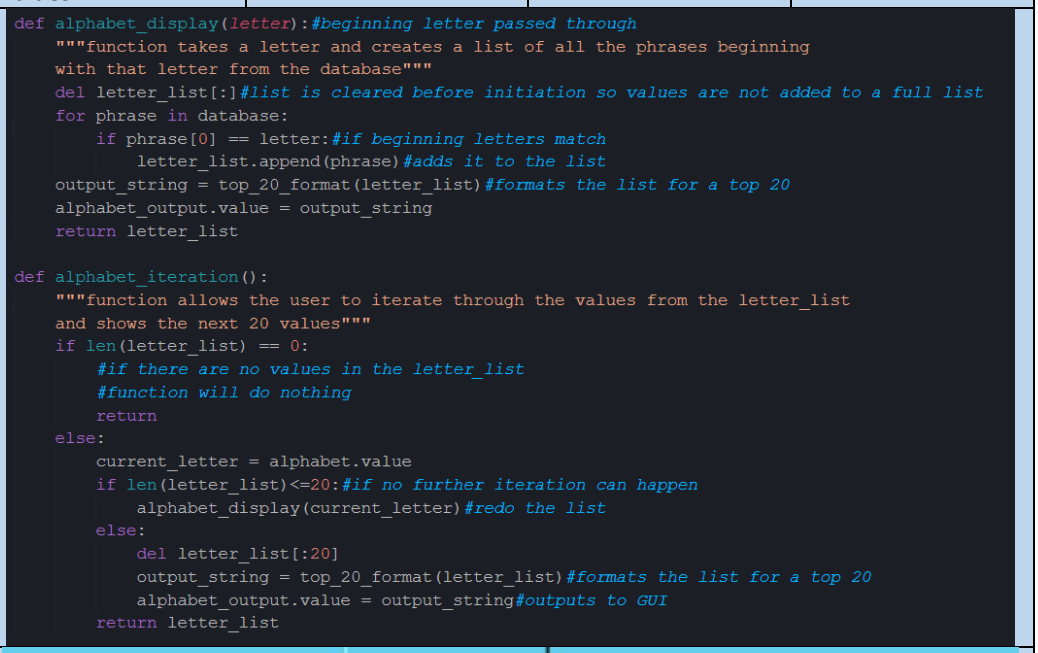
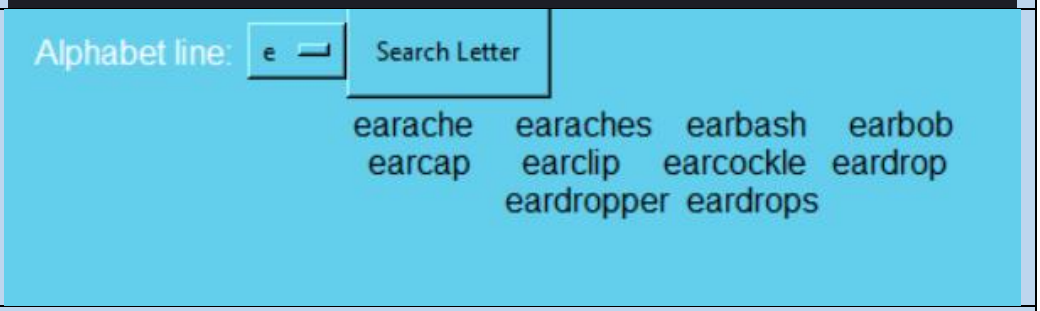
Test	Description	Test Data	Justification	Expected Output
Event Test 1	Program should be able to remove the first 20 elements from the list and update the output to the next 20 elements. If there are less than 20 elements list should redefine itself in order to obtain the original number of values.	Using a reduced list for the letter "e" with only 50 values. This will be achieved by creating the list normally then deleting all but 30 values (del letter_list[50:])	The program needs to be able to both refresh the list (present the next 20 values) as well as redo the list (re-create the initial letter list with all of the variables) once the list is finished.	Each time the button is pressed the output should change until the end of the list is reached. At this point the list should start all over again – showing the first values.
Test Code				
Result Screenshot				
Comments	Test failed: After reaching the end of the list the list re-initiates itself but does this in the same output block. This can be seen by "e" (first in list) coming after "earable" (last in list). A small change needs to be made so that the whole block resets once you have reached the end of the list. The second problem with this method is that when trying to change the letter (E.G. to letter "b") the program doesn't clear the current list as it should. It continues to display the current letter list until that list has been iterated all the way through. This is not such a problem with only 50 values (max 3 iterations) but will become a problem when the lists are unaltered and therefore much longer.			

```
def alphabet_iteration():
    """function allows the user to iterate through the values from the letter_list
    and shows the next 20 values"""
    if len(letter_list) == 0:
        #if there are no values in the letter_list
        #function will do nothing
        return
    else:
        current_letter = letter_list[0][0]
        if len(letter_list)<=20:#if no further iteration can happen
            alphabet_display(current_letter)#redo the list
        else:
            output_string = top_20_format(letter_list)#formats the list for a top 20
            alphabet_output.value = output_string#outputs to GUI
        return letter_list
```


Remedial Actions for Alphabet Line - Part 2 – Iteration 1

Part 2 – Iteration 2

Alphabet Line Code - Part 2 – Iteration 2

Test	Description	Test Data	Justification	Expected Output
Event Test 1	Program should be able to remove the first 20 elements from the list and update the output to the next 20 elements. If there are less than 20 elements list should redefine itself in order to obtain the original number of values.	Using a reduced list for the letter "e" with only 50 values. This will be achieved by creating the list normally then deleting all but 30 values (del letter_list[50:])	The program needs to be able to both refresh the list (present the next 20 values) as well as redo the list (re-create the initial letter list with all of the variables) once the list is finished.	Each time the button is pressed the output should change until the end of the list is reached. At this point the list should start all over again – showing the first values.
Test Code				
Result Screenshot				
Comments	Program works as expected – The program correctly displays what will be the last iteration of the letter list before refreshing the letter list.			

```
def alphabet_display(letter):#beginning letter passed through
    """function takes a letter and creates a list of all the phrases beginning
    with that letter from the database"""
    del letter_list[:]#list is cleared before initiation so values are not added to a full list
    for phrase in database:
        if phrase[0] == letter:#if beginning letters match
            letter_list.append(phrase)#adds it to the list
    output_string = top_20_format(letter_list)#formats the list for a top 20
    alphabet_output.value = output_string
    return letter_list

def alphabet_iteration():
    """function allows the user to iterate through the values from the letter_list
    and shows the next 20 values"""
    if len(letter_list) == 0:
        #if there are no values in the letter_list
        #function will do nothing
        return
    else:
        current_letter = alphabet.value
        if len(letter_list)<=20:#if no further iteration can happen
            alphabet_display(current_letter)#redo the list
        else:
            del letter_list[:20]
            output_string = top_20_format(letter_list)#formats the list for a top 20
            alphabet_output.value = output_string#outputs to GUI
        return letter_list
```

Part 2 – Iteration 3

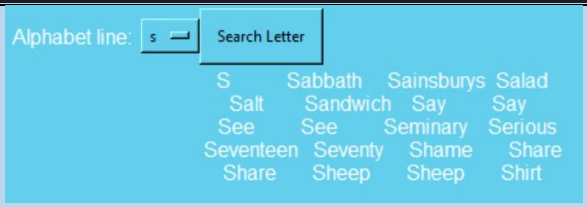
Alphabet Line Code – Part 2 – Iteration 3

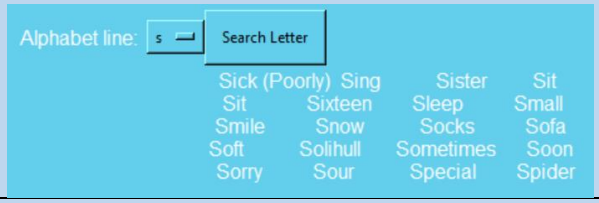
Test	Description	Test Data	Justification	Expected Output
Event Test 1	Program should use the phrases that it has available to be presented in signing form.	Using total_vids array that is obtained from the online database	Program can display suggestions for the user to search in the search engine that will output dedicated signings.	Program will output phrases from the online database of phrases. It should format these phrases in the exact way it had been previously.

Test Code

```
def alphabet_display(letter):#beginning letter passed through
    """function takes a letter and creates a list of all the phrases beginning
    with that letter from the database"""
    del letter_list[:]#list is cleared before initiation so values are not added to a full list
    counter = 0
    for word in total_vids:
        if word["Phrase"][0].lower() == letter:#if beginning letters match
            letter_list.append(word["Phrase"])#adds it to the list
    output_string = top_20_format(letter_list)#formats the list for a top 20
    alphabet_output.value = output_string
    return letter_list

def alphabet_iteration():
    """function allows the user to iterate through the values from the letter_list
    and shows the next 20 values"""
    if len(letter_list) == 0:
        #if there are no values in the letter_list
        #function will do nothing
        return
    else:
        current_letter = alphabet.value
        if len(letter_list)<=20:#if no further iteration can happen
            alphabet_display(current_letter)#redo the list
        else:
            del letter_list[:20]
            output_string = top_20_format(letter_list)#formats the list for a top 20
            alphabet_output.value = output_string#outputs to GUI
        return letter_list
```

Result Screenshot





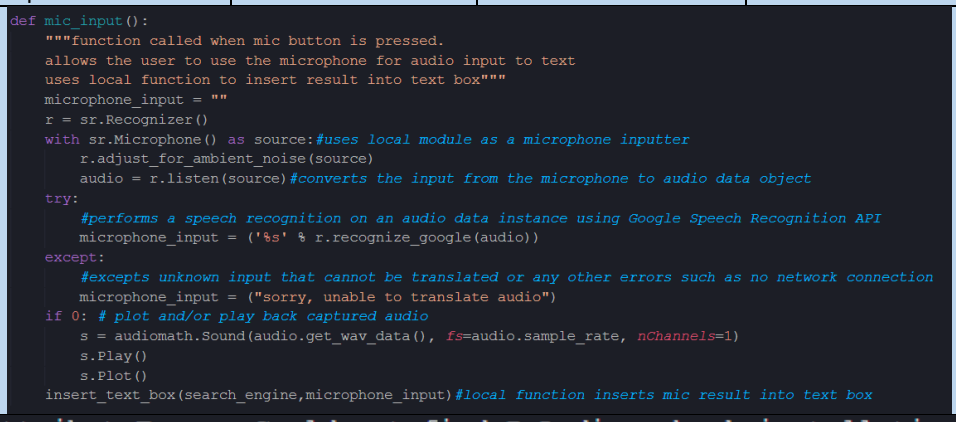
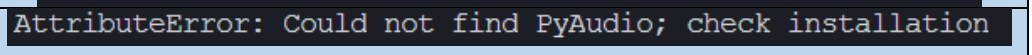
Comments

Program works as expected – Both output screenshots show the program correctly outputting the phrases in alphabetical order as well as in the correct format.

Microphone Input

Iteration 1:

Microphone Input Code – Iteration 1

Test	Description	Test Data	Justification	Expected Output
Event Test 1	Program should allow user to start microphone input via a button and speak into the microphone to input data.	Spoken: “hello world”	A simple phrase for the speech recognition to make sure it works	Text: “hello world”
Test Code				
Result Screenshot				
Comments	Test failed: as the program is run and the micophone functionality is tested there is an instant error that the PyAudio module is inaccessible. I proceeded to try and install PyAudio and then re-install the speech_recognition module. After doing this I re-ran the code and came to the same error.			

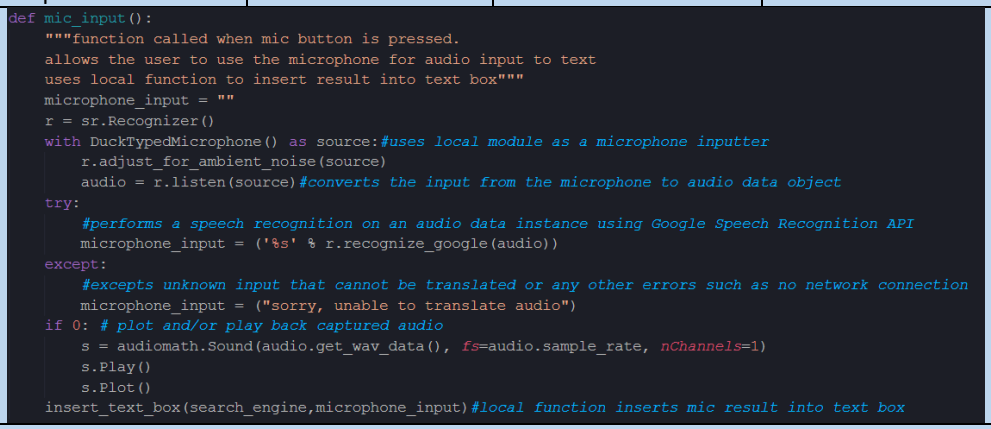
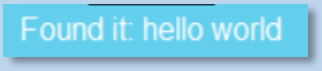
```
def mic_input():
    """function called when mic button is pressed.
    allows the user to use the microphone for audio input to text
    uses local function to insert result into text box"""
    microphone_input = ""
    r = sr.Recognizer()
    with sr.Microphone() as source:#uses local module as a microphone inputter
        r.adjust_for_ambient_noise(source)
        audio = r.listen(source)#converts the input from the microphone to audio data object
    try:
        #performs a speech recognition on an audio data instance using Google Speech Recognition API
        microphone_input = ('%s' % r.recognize_google(audio))
    except:
        #excepts unknown input that cannot be translated or any other errors such as no network connection
        microphone_input = ("sorry, unable to translate audio")
    if 0: # plot and/or play back captured audio
        s = audiomath.Sound(audio.get_wav_data(), fs=audio.sample_rate, nChannels=1)
        s.Play()
        s.Plot()
    insert_text_box(search_engine,microphone_input)#local function inserts mic result into text box
```

```
AttributeError: Could not find PyAudio; check installation
```

Remedial actions for Microphone Input – Iteration 1

Iteration 2:

Microphone Input Code – Iteration 2

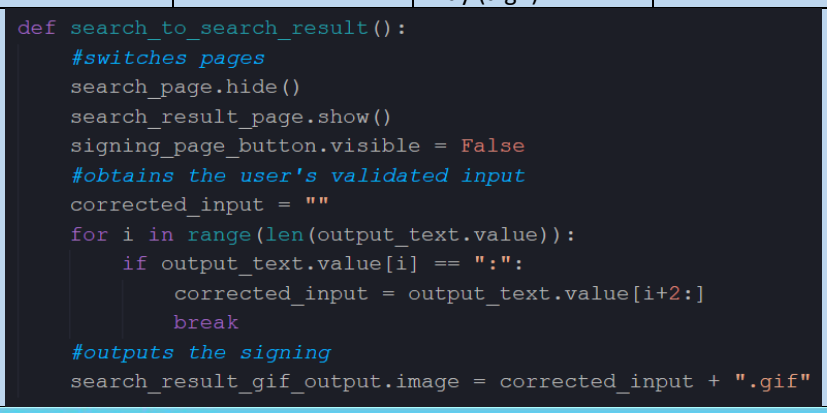
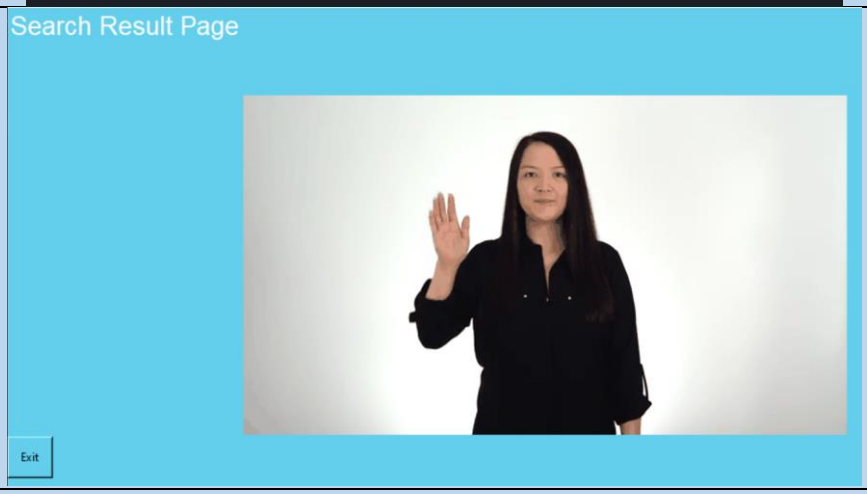
Test	Description	Test Data	Justification	Expected Output
Event Test 1	Program should allow user to start microphone input via a button and speak into the microphone to input data.	Spoken: “hello world”	A simple phrase for the speech recognition to make sure it works	Text: “hello world”
Test Code				
Result Screenshot				
Comments	Program works as expected			

```
def mic_input():
    """function called when mic button is pressed.
    allows the user to use the microphone for audio input to text
    uses local function to insert result into text box"""
    microphone_input = ""
    r = sr.Recognizer()
    with DuckTypedMicrophone() as source:#uses local module as a microphone inputter
        r.adjust_for_ambient_noise(source)
        audio = r.listen(source)#converts the input from the microphone to audio data object
    try:
        #performs a speech recognition on an audio data instance using Google Speech Recognition API
        microphone_input = ('%s' % r.recognize_google(audio))
    except:
        #excepts unknown input that cannot be translated or any other errors such as no network connection
        microphone_input = ("sorry, unable to translate audio")
    if 0: # plot and/or play back captured audio
        s = audiomath.Sound(audio.get_wav_data(), fs=audio.sample_rate, nChannels=1)
        s.Play()
        s.Plot()
    insert_text_box(search_engine,microphone_input)#local function inserts mic result into text box
```

Outputting a Phrase

Part 1 – Iteration 1

Outputting a Phrase code – Part 1 – Iteration 1

Test	Description	Test Data	Justification	Expected Output
Event Test 1	The program should output the corresponding BSL signing to the inputted phrase	"hello"	The program needs to be able to output the signing to the user in an understandable way (a gif)	Program should output the signing for "hello"
Test Code				
Result Screenshot				
Comments	The program works as expected – the program works as expected but only works with one phrase inputs. The program also needs to be able to tackle multiple word phrases such as "hello world"			

```
def search_to_search_result():
    #switches pages
    search_page.hide()
    search_result_page.show()
    signing_page_button.visible = False
    #obtains the user's validated input
    corrected_input = ""
    for i in range(len(output_text.value)):
        if output_text.value[i] == ":":
            corrected_input = output_text.value[i+2:]
            break
    #outputs the signing
    search_result_gif_output.image = corrected_input + ".gif"
```

Part 2 – Iteration 1

Outputting a Phrase code – Part 2 – Iteration 1

Test	Description	Test Data	Justification	Expected Output
Event Test 1	Program should allow the user to enter a phrase containing more than one word and output all the phrases.	"hello world"	The user needs to be able to translate a multitude of phrases at once.	Program should output a gif with a button that when pressed shows the next gif instead.

Test Code

```
def search_to_search_result():
    """switches to the search result page where the signings are presented to the user.
    this function finds the signings of the specific phrases in online database and
    outputs it to the user in the correct order via a .gif"""
    #switches pages
    search_page.hide()
    search_result_page.show()
    signing_page_button.visible = False
    #obtains corrected user input
    corrected_input = ""
    for i in range(len(output_text.value)):
        if output_text.value[i] == ":":
            corrected_input = output_text.value[i+2:]
            break
    #sets intro text value
    search_result_user_input.value = "Searching for: " + corrected_input
    clear_search_page()
    correct_input_list = corrected_input.split()#splits the user input into a list
    del search_output_file_names[:]#clears the global list
    #iterates through each word in the input
    for i in range(len(correct_input_list)):
        temp_file_name = correct_input_list[i] + ".gif"
        search_output_file_names.append(temp_file_name)
    iterate_search_result()#presents the result to the user

def iterate_search_result():
    """Outputs each signing and iterates to the next signing phrase in the list when the button pressed.
    The iteration button only appears if there is more than one phrase.
    Also updates the text at the top indicating what is currently being displayed."""
    #shows the iterate button if there are more than one phrases
    if len(search_output_file_names)>1:
        search_result_next_button.show()
    else:
        search_result_next_button.hide()
    try:
        #presents description of the current phrase being shown
        search_resut_current_output.value = "Currently showing: " + str(search_output_file_names[0]).replace(".gif","")
        search_result_gif_output.image = search_output_file_names[0]#outputs first value in list
        del search_output_file_names[0]#gets rid of the first value in list
    except:
        pass
```

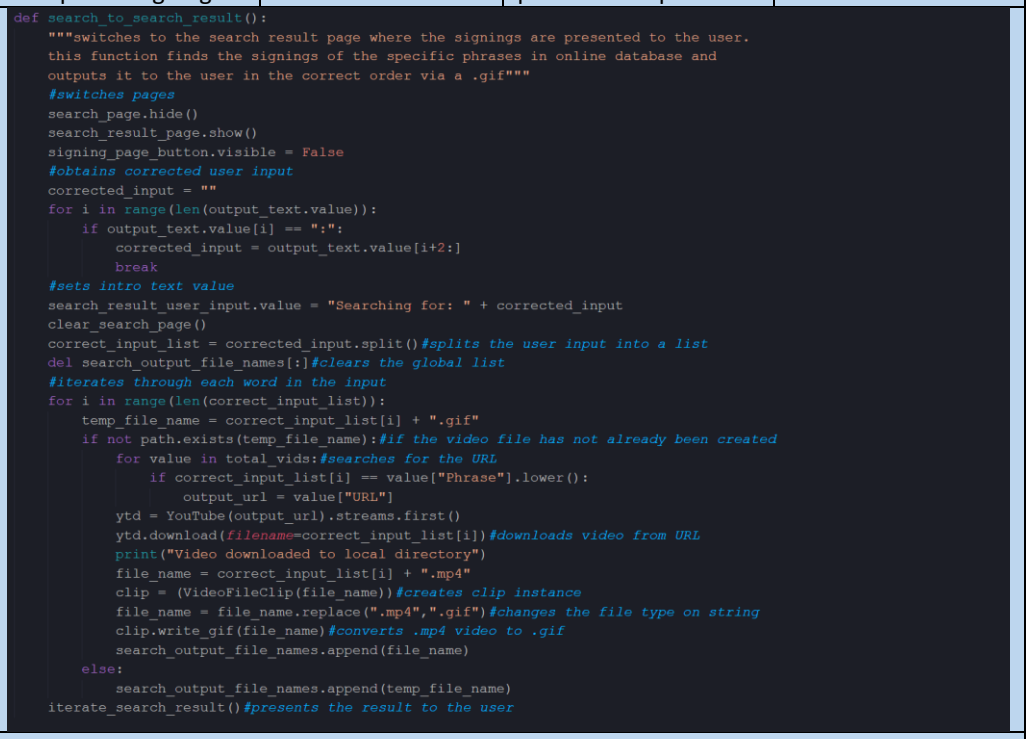
Result Screenshot

Comments

The program works as expected – program also edits the presence of iteration button so that if it is no longer needed (no more values to iterate through) then it will disappear.

Part 3 – Iteration 1

Outputting a Phrase code – Part 3 – Iteration 1

Test	Description	Test Data	Justification	Expected Output
Event Test 1	Program should use the BSL online database to obtain the correct URL in order to download the specific signing.	"captain hat"	The program needs to utilise the phrases within the online database to allow for a greater number of potential outputs.	Program should output a gif with a button that when pressed shows the next gif instead.
Test Code				
Result Screenshot				
Comments	Test failed – the test was not successful due to problems within the module (pytube) used to download the videos from youtube.			

```
def search_to_search_result():
    """switches to the search result page where the signings are presented to the user.
    this function finds the signings of the specific phrases in online database and
    outputs it to the user in the correct order via a .gif"""
    #switches pages
    search_page.hide()
    search_result_page.show()
    signing_page_button.visible = False
    #obtains corrected user input
    corrected_input = ""
    for i in range(len(output_text.value)):
        if output_text.value[i] == ":":
            corrected_input = output_text.value[i+2:]
            break
    #sets intro text value
    search_result_user_input.value = "Searching for: " + corrected_input
    clear_search_page()
    correct_input_list = corrected_input.split()#splits the user input into a list
    del search_output_file_names[:]#clears the global list
    #iterates through each word in the input
    for i in range(len(correct_input_list)):
        temp_file_name = correct_input_list[i] + ".gif"
        if not path.exists(temp_file_name):#if the video file has not already been created
            for value in total_vids:#searches for the URL
                if correct_input_list[i] == value["Phrase"].lower():
                    output_url = value["URL"]
            ytd = YouTube(output_url).streams.first()
            ytd.download(filename=correct_input_list[i])#downloads video from URL
            print("Video downloaded to local directory")
            file_name = correct_input_list[i] + ".mp4"
            clip = (VideoFileClip(file_name))#creates clip instance
            file_name = file_name.replace(".mp4",".gif")#changes the file type on string
            clip.write_gif(file_name)#converts .mp4 video to .gif
            search_output_file_names.append(file_name)
        else:
            search_output_file_names.append(temp_file_name)
    iterate_search_result()#presents the result to the user
```

Remedial actions for Outputting a Phrase – Part 3 – Iteration 1

Part 3 – Iteration 2

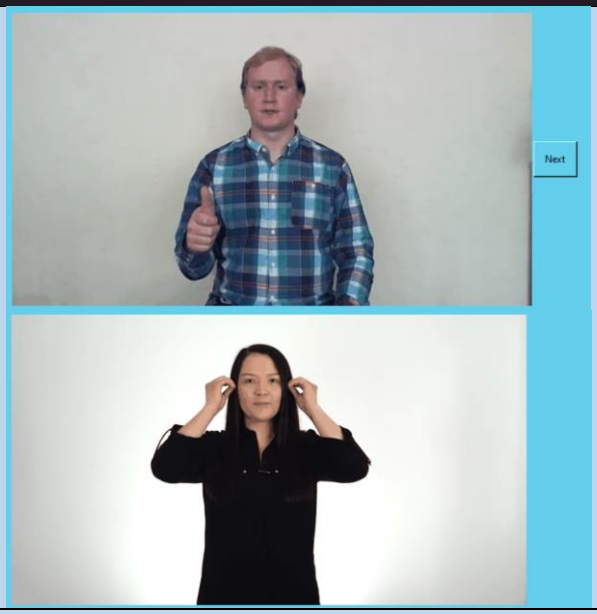
Outputting a Phrase code – Part 3 – Iteration 2

Test	Description	Test Data	Justification	Expected Output
Event Test 1	Program should use the BSL online database to obtain the correct URL in order to download the specific signing.	"captain hat"	The program needs to utilise the phrases within the online database to allow for a greater number of potential outputs.	Program should output a gif with a button that when pressed shows the next gif instead.

Test Code

```
def search_to_search_result():
    """switches to the search result page where the signings are presented to the user.
    this function finds the signings of the specific phrases in online database and
    outputs it to the user in the correct order via a .gif"""
    #switches pages
    search_page.hide()
    search_result_page.show()
    signing_page_button.visible = False
    #obtains corrected user input
    corrected_input = ""
    for i in range(len(output_text.value)):
        if output_text.value[i] == ":":
            corrected_input = output_text.value[i+2:]
            break
    #sets intro text value
    search_result_user_input.value = "Searching for: " + corrected_input
    clear_search_page()
    correct_input_list = corrected_input.split()#splits the user input into a list
    del search_output_file_names[:]#clears the global list
    #iterates through each word in the input
    for i in range(len(correct_input_list)):
        temp_file_name = correct_input_list[i] + ".gif"
        if not path.exists(temp_file_name):#if the video file has not already been created
            for value in total_vids:#searches for the URL
                if correct_input_list[i] == value["Phrase"].lower():
                    output_url = value["URL"]
            ytd = YouTube(output_url).streams.first()
            ytd.download(filename=correct_input_list[i])#downloads video from URL
            print("Video downloaded to local directory")
            file_name = correct_input_list[i] + ".mp4"
            clip = (VideoFileClip(file_name))#creates clip instance
            file_name = file_name.replace(".mp4",".gif")#changes the file type on string
            clip.write_gif(file_name)#converts .mp4 video to .gif
            search_output_file_names.append(file_name)
        else:
            search_output_file_names.append(temp_file_name)
    iterate_search_result()#presents the result to the user
```

Result Screenshot



Comments

The program works as expected – the program works well for one word phrases in the database. It also needs to be able to transcribe phrases that are not in the database using letters as well as multiple word phrases (e.g. good morning)

Part 4 – Iteration 1

Outputting a Phrase code – Part 4 – Iteration 1

Test	Description	Test Data	Justification	Expected Output
Event Test 1	Program should anticipate that there may be longer signings that can make up more than one word in the phrase. Words without signings should use the alphabet.	"I love you turing"	"I love you" is stored as a single signing alongside each individual word. "turing" is not a phrase stored and so will have to be made up with letters.	Program should find the most compatible signing for each phrase.

Test Code

```
def create_url_list(user_string):
    found_values = []#list of phrases that are in input
    user_input_list = user_string.split()
    for i in range(len(user_input_list)):
        #creates 2d list of input [user word,index of given phrase in found_values]
        user_input_list[i] = [user_input_list[i]]
        user_input_list[i].append(None)

    for value in total_vids:#loops through all phrases
        db_phrase = value["Phrase"].lower()#uses lowercase phrases
        if db_phrase in user_string:#if phrase present in input
            found_values.append(value["Phrase"])#appends formatted phrase to list
            for i in range(len(user_input_list)):#loops through each word of user input
                if db_phrase.split()[0] == user_input_list[i][0]:#lines up phrase and user input
                    for j in range(len(db_phrase.split())):#loops through length of phrase
                        if user_input_list[i+j][1] == None:#if there isnt anything found for a phrase
                            user_input_list[i+j][1] = len(found_values)-1
                            #current phrase set as value via pointer
                        elif len(db_phrase.split()) > len(found_values[user_input_list[i+j][1]]):
                            #if current phrase is bigger than previous phrase
                            user_input_list[i+j][1] = len(found_values)-1
                            #current phrase set at value via pointer

    for i in range(len(user_input_list)):#iterating through user input
        if user_input_list[i][1] == None:#find word that has no signings
            temp_list = []
            for j in range(len(user_input_list[i][0])):#loops through length of word
                for k in range(len(found_values)):#loops through the found values
                    if user_input_list[i][0][j] == found_values[k].lower():
                        #if current letter is the letter in found values
                        temp_list.append(k)#appends index of letter to temp list
                        break
            user_input_list[i][1] = temp_list
            #temp list is used in place where other words have a single phrase
```

Result Screenshot

```
        print(found_values)
        return user_input_list

    print(create_url_list("i love you turing"))
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\zaina\Documents\Computer Science\Zain Altaf 4132 OCR H446-0
n37/python.exe" "c:/Users/zaina/Documents/Computer Science/Zain Altaf 4
['G', 'I', 'I Love You', 'Love', 'N', 'R', 'T', 'U', 'You']
[['i', 2], ['love', 2], ['you', 2], ['turing', [6, 7, 5, 1, 4, 0]]]
PS C:\Users\zaina\Documents\Computer Science\Zain Altaf 4132 OCR H446-0
```

Comments

The program works as expected – program outputs each of the words in the phrase with a specific index for where the signing is in a separate list. these 2 lists will be used in conjunction when outputting to the user.

Part 4 – Iteration 2

Outputting a Phrase code – Part 4 – Iteration 2

Test	Description	Test Data	Justification	Expected Output
Event Test 1	Program should use the algorithm developed in iteration 1 inside of the GUI to output phrases correctly	"I love you turing"	"I love you" is stored as a single signing alongside each individual word. "turing" is not a phrase stored and so will have to be made up with letters.	Program should output each signing as a gif in the correct order with an exit button that resets the signing queue

Test Code

```
def gif_setup():
    """this function finds the signings of the specific phrases in online database"""
    #obtains corrected user input
    corrected_input = ""
    for i in range(len(output_text.value)):
        if output_text.value[i] == ":":
            corrected_input = output_text.value[i+2:]
            break
    #sets intro text value
    search_result_user_input.value = "Searching for: " + corrected_input
    clear_search_page()
    del search_output_file_names[:]#clears the global list
    user_input_list,found_values = create_url_list(corrected_input.lower())
    #obtains the total phrases found in the input and a 2d list containing
    #each word in the input and a pointer to a value in the found_values list
    temp_list = []
    counter = 0
    while counter < len(user_input_list):
        search_output_file_names.append(found_values[user_input_list[counter][1]])
        counter+=len(found_values[user_input_list[counter][1]].split())
    iterate_search_result()

def iterate_search_result():
    """Outputs each signing and iterates to the next signing phrase in the list when the button pressed.
    Downloads the next signing video phrase and converts it into .gif.
    The iteration button only appears if there is more than one phrase.
    Also updates the text at the top indicating what is currently being displayed."""
    search_result_gif_output.value = None
    #shows the iterate button if there are more than one phrases
    if len(search_output_file_names)>1:
        search_result_next_button.show()
    else:
        search_result_next_button.hide()
    try:
        #presents description of the current phrase being shown
        search_resut_current_output.value = "Currently showing: "+str(search_output_file_names[0])
        if not os.path.exists(search_output_file_names[0]+".gif"):
            #if video needs to be downloaded
            for value in total_vids:#obtains correct URL
                if search_output_file_names[0] == value["Phrase"]:
                    output_url = value["URL"]
                    break
            ytd = YouTube(output_url).streams.first().download(filename=search_output_file_names[0])
            print("Video downloaded to local directory")
            file_name = search_output_file_names[0] + ".mp4"
            clip = (VideoFileClip(file_name))#creates clip instance
            file_name = file_name.replace(".mp4",".gif")#changes the file type on string
            clip.write_gif(file_name)#converts .mp4 video to .gif

            search_result_gif_output.image = file_name#outputs first value in list
            del search_output_file_names[0]#gets rid of the first value in list
        else:

            file_name = search_output_file_names[0]+".gif"
            search_result_gif_output.image = file_name#outputs first value in list
            del search_output_file_names[0]#gets rid of the first value in list
    except:
        pass
```

Result Screenshot

```
    search_output_file_names.append(found_values[user_input_list[counter][1]])
TypeError: list indices must be integers or slices, not list
```

Comments

Test failed – program does not take into account the list within user_input_list. This is the list containing all the individual indexes for each of the letters for "turing"

Remedial actions for Outputting a Phrase code – Part 4 – Iteration 2

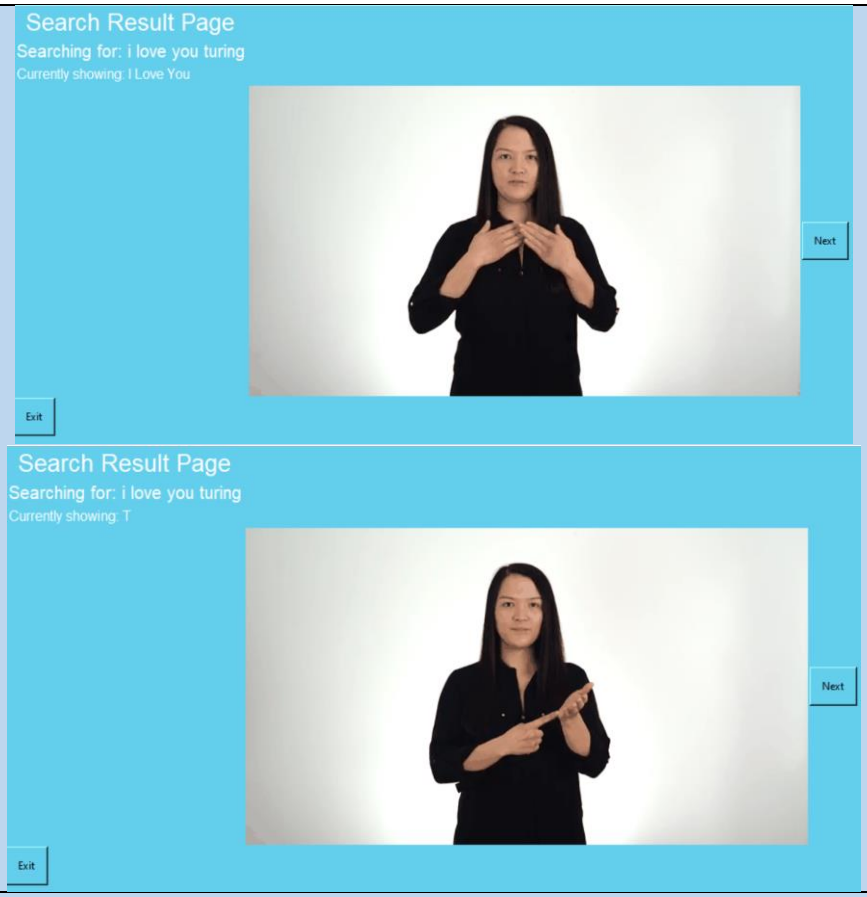
Part 4 – Iteration 3

Outputting a Phrase code – Part 4 – Iteration 3

Test	Description	Test Data	Justification	Expected Output
Event Test 1	Program should use the algorithm developed in iteration 1 inside of the GUI to output phrases correctly	"I love you turing"	"I love you" is stored as a single signing alongside each individual word. "turing" is not a phrase stored and so will have to be made up with letters.	Program should output each signing as a gif in the correct order with an exit button that resets the signing queue

Test Code

```
def gif_setup():
    """this function finds the signings of the specific phrases in online database"""
    #obtains corrected user input
    corrected_input = ""
    for i in range(len(output_text.value)):
        if output_text.value[i] == ":":
            corrected_input = output_text.value[i+2:]
            break
    #sets intro text value
    search_result_user_input.value = "Searching for: " + corrected_input
    clear_search_page()
    del search_output_file_names[:]#clears the global list
    user_input_list,found_values = create_url_list(corrected_input.lower())
    #obtains the total phrases found in the input and a 2d list containing
    #each word in the input and a pointer to a value in the found_values list
    temp_list = []
    counter = 0
    while counter < len(user_input_list):
        #this loops round the 2d list and abstracts all of the phrase descriptions
        #by looking at each of the indexes in found_values list
        if type(user_input_list[counter][1]) is list:
            for j in range(len(user_input_list[counter][1])):
                search_output_file_names.append(found_values[user_input_list[counter][1][j]])
            counter+=1
        else:
            search_output_file_names.append(found_values[user_input_list[counter][1]])
            counter+=len(found_values[user_input_list[counter][1]].split())
    iterate_search_result()#iterates once

def iterate_search_result():
    """Outputs each signing and iterates to the next signing phrase in the list when the button pressed.
    Downloads the next signing video phrase and converts it into .gif.
    The iteration button only appears if there is more than one phrase.
    Also updates the text at the top indicating what is currently being displayed."""
    #shows the iterate button if there are more than one phrases
    if len(search_output_file_names)>1:
        search_result_next_button.show()
    else:
        search_result_next_button.hide()
    try:
        #presents description of the current phrase being shown
        search_resut_current_output.value = "Currently showing: "+str(search_output_file_names[0])
        if not path.exists(search_output_file_names[0]+".gif"):
            #if video needs to be downloaded
            for value in total_vids:#obtains correct URL
                if search_output_file_names[0] == value["Phrase"]:
                    output_url = value["URL"]
                    break
            ytd = YouTube(output_url).streams.first()
            ytd.download(filename=search_output_file_names[0])#downloads video from URL
            print("Video downloaded to local directory")
            file_name = search_output_file_names[0] + ".mp4"
            clip = (VideoFileClip(file_name))#creates clip instance
            file_name = file_name.replace(".mp4",".gif")#changes the file type on string
            clip.write_gif(file_name)#converts .mp4 video to .gif
        else:
            file_name = search_output_file_names[0]+".gif"
        search_result_gif_output.image = file_name#outputs first value in list
        del search_output_file_names[0]#gets rid of the first value in list
    except:
        pass
```

Result Screenshot	
Comments	The program works as expected – program is able to integrate the online database and downloading functionality flawlessly. After continuous testing with exiting and re-entering a new input the program correctly outputs the signing necessary.

Functional Chunk 2 – Searching Through Database

Back to Contents Page

Optimised Database Search – Part 1 – Iteration 1

Optimised Database Search Code – Part 1 – Iteration 1

Test	Description	Test Data	Justification	Expected Output
Event Test 1	Program should validate the inputted phrase and use the validated string to search through a database with. Program should output the most recent corresponding phrases.	“rains78932”	Program needs to be able to take in values without causing errors due to invalid characters. Program needs to be able to return at least 1 value and the most recent corresponding value may be the intended value.	[[“rain”] ,[“raining”]]

Test Code

```
database_list= [["good morning",None,None],["good evening",None,None],["rain",None,None],["raining",None,None]]
user_input = "rains78932"
new_string = ""
#input validation - removes numbers/special characters
user_input = user_input.lower()
for i in range(len(user_input)):#iterates through user input
    if user_input[i] == " ":
        new_string+=user_input[i]#adds spaces to final string
    elif user_input[i].isalpha():
        new_string+=user_input[i]#adds letter characters

def phrase_finder(database,user_input,index):
    """iterates though each phrase in the database list letter by letter
    and checks agains the same index letter of the user input.
    returns the values that correspond"""
    temp_data_list = []
    for i in range(len(database)):#iterates through all the values of database
        data_string = database[i][0]#actual string value of each iteration
        if index<len(data_string):#in case the user input is longer than the the phrase
            phrase_letter = data_string[index]
            if phrase_letter == user_input[index]:
                temp_data_list.append(database[i])#adds corresponding phrases to be returned
    return temp_data_list

for i in range(len(new_string)):#loops through each character in the entered string
    holder_list  = phrase_finder(database_list,new_string,i)
    if len(holder_list) == 0:
        pass
    else:
        database_list = holder_list
    #updates database_list each iteration so that only the currently corresponding
    #values are added means that each letter iteration for the user input will be
    #searched against an increasingly condensed list of values
    #making it more efficient to find
print("user input: " + str(user_input))
print("Values found from database: " + str(database_list))
```

Result Screenshot

```
user input: rains78932
Values found from database: [['rain', None, None], ['raining', None, None]]
```

Comments

The program works as expected: Returns values that have the same first section of the user input. The main advantage using this process is the fact that the list prints out the last correlated value found in the list. This means that if mistakes are made in the string towards the end, it will still output the values that had the same beginning. However, using this method, the program begins to fail if there is a mistake towards the beginning of the input causing a massive difference between the input and output.

Optimised Database Search – Part 2 – Iteration 1

Optimised Database Search Code – Part 2 – Iteration 1

Test	Description	Test Data	Justification	Expected Output
Event Test 1	Program should take in a string from a user that can be made up of any number of words and search through the database and find the most similar phrase. This phrase should then be outputted to the user	"helllo worlds"	This test data showcases the main functional advantage of using the fuzzywuzzy module – the ability to return values that are similar to the input but not the same.	"hello world"

Test Code

```
from fuzzywuzzy import fuzz, process #'fuzzy' matching
import json
with open("Development\Chunk1_Searching\words_dictionary.json") as f:
    database = json.load(f)#loads the word bank database
f.close()

user_input = input("please enter the value you would like to search: ")
total_list = []
user_input = user_input.split()
return_list = []

for user_word in user_input:
    single_word_list = []#this list stores potential phrases for the user input
    for phrase in database:#loops through the entire database
        options_list = []#temp list to store each potential phrase
        percent_acc = fuzz.ratio(user_word,phrase)
        #uses fuzz.ratio to find how similar the input is to words in the database
        if percent_acc > 80:#if the word is relatively accurate
            options_list.append(phrase)#adds the phrase to the temp list
            options_list.append(percent_acc)#adds the level of accuracy to the temp list
            for i in range(len(single_word_list)):#loops through the entire selection of possibilities
                phrase_percent = int(single_word_list[i][1])
                if percent_acc>=phrase_percent:
                    single_word_list.insert(i,options_list)
                    #inserts the temp list with the newest phrase so that the final list is sorted
                    break#breaks out of the loop
                elif (i+1) == len(single_word_list):
                    single_word_list.insert(i+1,options_list)
                    #if the phrase is the least accurate it is inserted to the back of the list
                    break#breaks out of the loop
            if len(single_word_list) == 0:#if there are no values in the list
                single_word_list.append(options_list)
    total_list.append(single_word_list)#adds the list of all the potential phrases to the total list

for word_result in total_list:
    if len(word_result) == 0:#if no results were found
        return_list.append([])
        return_list[-1].append("no results found please try again")#error message
        return_list[-1].append(0)#identifier value that can be used to show no correlation
    else:
        return_list.append(word_result[0])#adds the most accurate value

#outputs to the user what the program thinks was meant by the user
print("\nSearch results: ")
for i in range(len(return_list)):
    if return_list[i][1] == 0:
        print("input '" + str(user_input[i]) + "' could not be understood. please try again.")
    elif return_list[i][1]<100:
        print("input '" + str(user_input[i]) + "' was not found. showing results for: " + str(return_list[i][0]))
    else:
        print("showing results for: " + str(return_list[i][0]))
```

Result Screenshot

```
please enter the value you would like to search: helllo worlds

Search results:
input 'helllo' was not found. showing results for: hello
showing results for: worlds
```

Comments

The program works as expected - the program does its job as it should but when searching for a similar value to "worlds" the program finds "worlds" to be more accurate to "world" which was what I intended to input. This is due to the fact that "worlds" was also in the database list but in having that there may not be a sign language phrase for "worlds". This will have to be observed later.

Optimised Database Search – Part 2 – Iteration 2

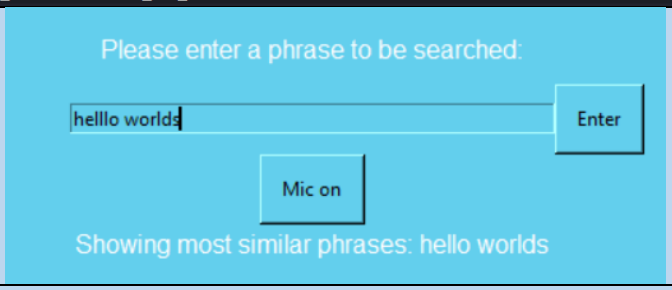
Optimised Database Search Code – Part 2 – Iteration 2

Test	Description	Test Data	Justification	Expected Output
Event Test 1	Program should take in a string from a user that can be made up of any number of words and search through the database and find the most similar phrase. This phrase should then be outputted to the user in a graphical user interface.	"helllo worlds"	This test data showcases the main functional advantage of using the fuzzywuzzy module – the ability to return values that are similar to the input but may not be the same. This silmultaneously has to update the top_20_list as well	"hello world"

Test Code

```
def searching_database(user_input,output_text):
    """takes user input as an attribute and searches through the database for it.
    this includes a fluffy search which outputs the most accurate word it can find in the database.
    this function also updates the top 20 list"""
    user_input_list = user_input.split()
    output_string = ""
    temp_list = []
    total_list = []
    for i in range(len(user_input_list)):#iterates through each inputted word
        single_word_list = fluffy_search(database,user_input_list[i],80)#creates a list of possible phrases
        total_list.append(single_word_list)#appends all possible phrases for each inputted word to total list
        if len(single_word_list) != 0:#if there were no found possibilities
            my_gui_string = top_20_adjust(top_20_list,total_list[i][0][0])
            #adds the searched phrase to the top 20 list and returns the new top 20 list to output
            output_string += total_list[i][0][0] + " " #adds new value to output to be printed

    if output_string == "":#if nothing was added to the string - if no words found in database
        output_text.value = "sorry. that couldn't be found, please try again"
    else:
        output_string = output_string.strip()#gets rid of any possible spaces on the ends
        correction = False#used to check if input needed to be corrected
        for word_list in total_list:
            if word_list[0][1] != 100:#if the accuracy value was not 100
                correction = True#phrase was corrected
        #different output message shown telling user if the phrase was corrected or not
        if correction == True:
            output_text.value = "Showing most similar phrases: " + str(output_string)
        else:
            output_text.value = "Found it: " + str(output_string)
        #presents the validated output to the user
    top_20_output_text.value = my_gui_string
```

Result Screenshot



Comments

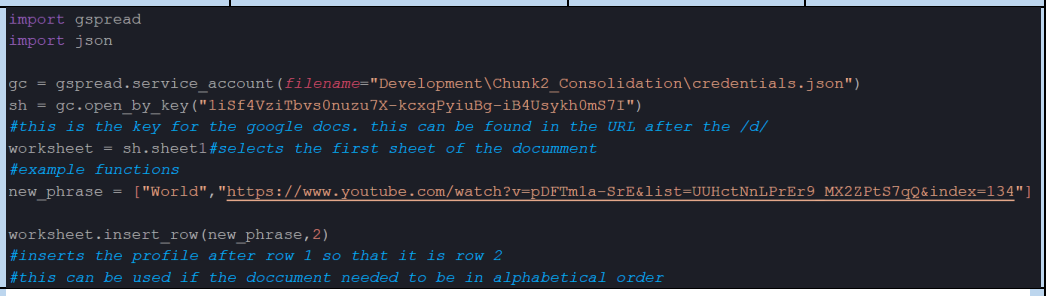
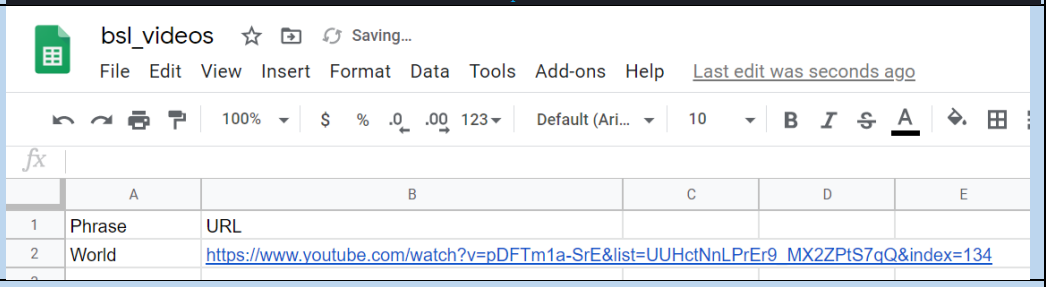
The program works as expected - the program does its job as it should but when searching for a similar value to "worlds" the program finds "worlds" to be more accurate to "world" which was what I intended to input. This is since "worlds" was also in the database list but in having that there may not be a sign language phrase for "worlds". This will have to be observed later.

Functional Chunk 3 – Consolidation of Learning

Return to Contents Page

Creating Online Database – Part 1 – Iteration 1

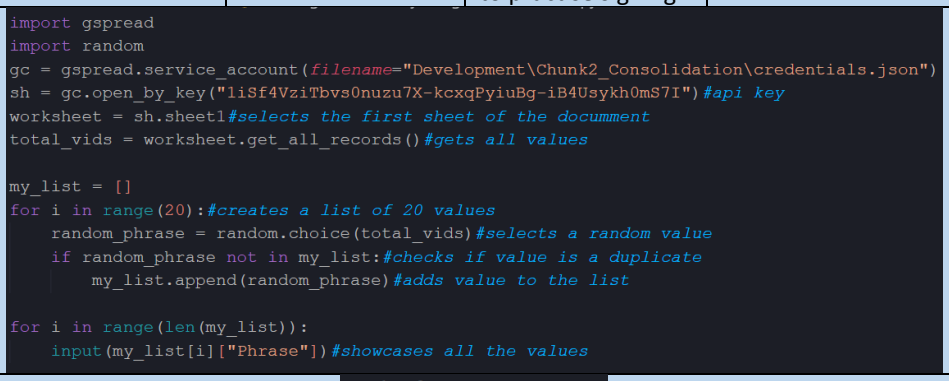
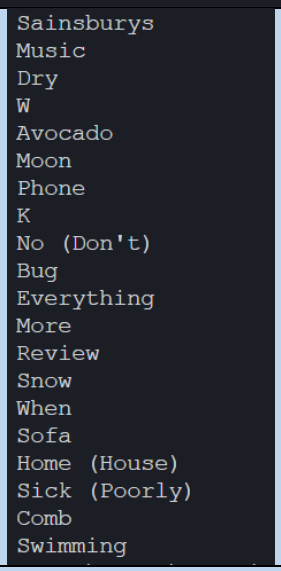
Creating Online Database Code – Part 1 – Iteration 1

Test	Description	Test Data	Justification	Expected Output
Event Test 1	The program should be able to edit an existing google sheets document in as a means of storing data online to be accessed later.	Phrase = "World" URL = "https://www.youtube.com/watch?v=pDFTm1a-SrE&list=UUHctNnLPrEr9_MX2ZPtS7qQ&index=134"	The program needs to be able to add specific phrases and their corresponding video in order to expand the database	Values should appear in the database in the corresponding columns
Test Code				
Result Screenshot				
Comments	The program works as expected			

```
import gspread
import json

gc = gspread.service_account(filename="Development\Chunk2_Consolidation\credentials.json")
sh = gc.open_by_key("1iSf4VziTbvs0nuzu7X-kcxqPyiuBg-iB4Usykh0mS7I")
#this is the key for the google docs. this can be found in the URL after the /d/
worksheet = sh.sheet1#selects the first sheet of the documment
#example functions
new_phrase = ["World","https://www.youtube.com/watch?v=pDFTm1a-SrE&list=UUHctNnLPrEr9_MX2ZPtS7qQ&index=134"]

worksheet.insert_row(new_phrase,2)
#inserts the profile after row 1 so that it is row 2
#this can be used if the doccument needed to be in alphabetical order
```

Creating Online Database – Part 2 – Iteration 1

Creating Online Database Code – Part 2 – Iteration 1

Test	Description	Test Data	Justification	Expected Output
Event Test 1	Program should use the online database as a source of data to create a random array of phrases for the user to practise.	Phrases from the bsl_videos database.	The program needs to be able to use the online database as the main source of information. The program also needs to be able to create a random array for the user to practise signing.	A list of 20 random phrases with corresponding URls.
Test Code				
Result Screenshot				
Comments	The program works as expected			

```
import gspread
import random
gc = gspread.service_account(filename="Development\Chunk2_Consolidation\credentials.json")
sh = gc.open_by_key("1iSf4VziTbvs0nuzu7X-kcxqPyiuBg-iB4Usykh0mS7I")#api key
worksheet = sh.sheet1#selects the first sheet of the documment
total_vids = worksheet.get_all_records()#gets all values

my_list = []
for i in range(20):#creates a list of 20 values
    random_phrase = random.choice(total_vids)#selects a random value
    if random_phrase not in my_list:#checks if value is a duplicate
        my_list.append(random_phrase)#adds value to the list

for i in range(len(my_list)):
    input(my_list[i]["Phrase"])#showcases all the values
```

```
Sainsburys
Music
Dry
W
Avocado
Moon
Phone
K
No (Don't)
Bug
Everything
More
Review
Snow
When
Sofa
Home (House)
Sick (Poorly)
Comb
Swimming
```

Creating Online Database – Part 3 – Iteration 1

Creating Online Database Code – Part 3 – Iteration 1

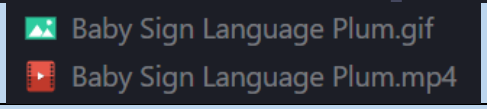
Test	Description	Test Data	Justification	Expected Output
Event Test 1	Program should take phrase input from the user and download the video from URL found in database. Video should be converted to .gif.	"Plum"	Program needs to be able to access video in .gif format to be outputted to user in GUIZERO	Videos of signing for given phrase in both .mp4 and .gif format.

Test Code

```
#imports
from pytube import YouTube, Playlist
from moviepy.editor import *

#funtions
def Search(search):
    #opens .csv file and reads in data
    database = open("Development\Chunk2_Consolidation\BSL_DATABASE.csv","r")
    database = database.readlines()
    #loops through each phrase
    for i in range(len(database)):
        #checks whether the user input string is in the database string
        if search in database[i]:
            #if the database string is found the URL is extracted
            result = database[i]
            result = result.replace(search, '')
            result = result.replace(',','')
            result = result.strip()
            #URL is then outputted
            return result

def Download(URL):
    #if no URL was found - if the phrase was not found
    if URL==None:
        print("ERROR")
    else:
        #video is downloaded to local drive
        ytd = YouTube(URL).streams.first().download()
        print("Video downloaded to local directory")
        print("file name:")
        VariableName = (YouTube(URL).streams[0].title)
        print(VariableName)
        file_name = ""
        for i in range(len(VariableName)):
            if VariableName[i] == " ":
                file_name += VariableName[i]
            elif VariableName[i].isalpha():
                file_name += VariableName[i]
        file_name += ".mp4"
        print(file_name)
        clip = (VideoFileClip(file_name))
        clip.write_gif("output.gif")

user_input = input("Enter a word: ")
#formats the user input to the correct format
user_input = user_input[0].upper() + user_input[1:].lower()
#finds the URL for the phrase
link = Search(user_input)
#downloads the video
download = Download(link)
```

Result Screenshot

```
Enter a word: Plum
Video downloaded to local directory
Current file name:
Baby Sign Language Plum.mp4
Converted file name:
Baby Sign Language Plum.gif
MoviePy - Building file Baby Sign Language Plum.gif with imageio.
```



Comments

The program works as expected – Output pictures show the video being downloaded from the URL and added to the local directory. These videos are then converted into .gif form so that they can be used in GUIZERO

Creating User Interface – Part 1 – Iteration 1

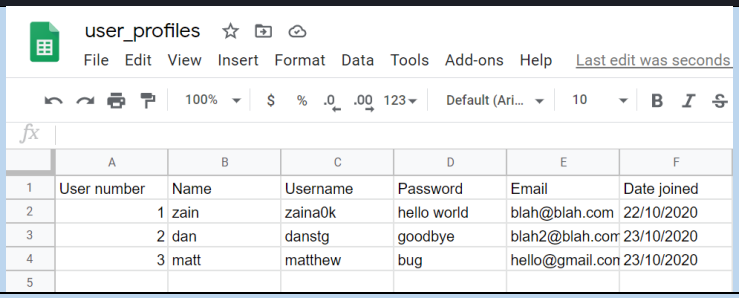
Creating User Interface Code – Part 1 – Iteration 1

Test	Description	Test Data	Justification	Expected Output
Event Test 1	Program should be able to create a user object with data which can then be outputted to the online db.	user_object = User("dan","danstg", "goodbye", "blah2@blah.com") Etc…	Program needs to update the online database when a user is added to the program so that it can be accessed later	Database updated with data in correct columns.

Test Code

```
class User:
    """class that creates user objects"""
    user_counter = 0
    def __init__(self,name,username,password,email):
        """initialising the class"""
        today = date.today()
        self.name = name
        self.date_joined = today.strftime("%d/%m/%Y")
        self.username = username
        self.password = password
        self.email = email
        self.user_number = User.user_counter + 1
        User.user_counter += 1
    def __str__(self):
        """print method shows all the values of the object"""
        print_str = ""
        print_str+= "user number: " + str(self.user_number) + "\n"
        print_str+= "name: " + str(self.name)+ "\n"
        print_str+= "username: " + str(self.username)+ "\n"
        print_str+= "password: " + str(self.password)+ "\n"
        print_str+= "date joined: " + str(self.date_joined)+ "\n"
        print_str+= "email: " + str(self.email)
        return print_str

def add_user_to_database(user_object):
    """function abstracts all of the data from the user object and appends the
    data to the database"""
    temp_list = []
    temp_list.append(user_object.user_number)
    temp_list.append(user_object.name)
    temp_list.append(user_object.username)
    temp_list.append(user_object.password)
    temp_list.append(user_object.email)
    temp_list.append(user_object.date_joined)
    worksheet.append_row(temp_list)
def set_user_counter():
    """method that sets the user counter as the most recent user number.
    This function should be called each time the program is run so that
    the user number can be set to the correct number."""
    User.user_counter = total_users[-1]["User number"]

set_user_counter()
user_object = User("dan","danstg","goodbye","blah2@blah.com")
user_object2 = User("matt","matthew","bug","hello@gmail.com")
add_user_to_database(user_object)
add_user_to_database(user_object2)
```

Result Screenshot



user_profiles

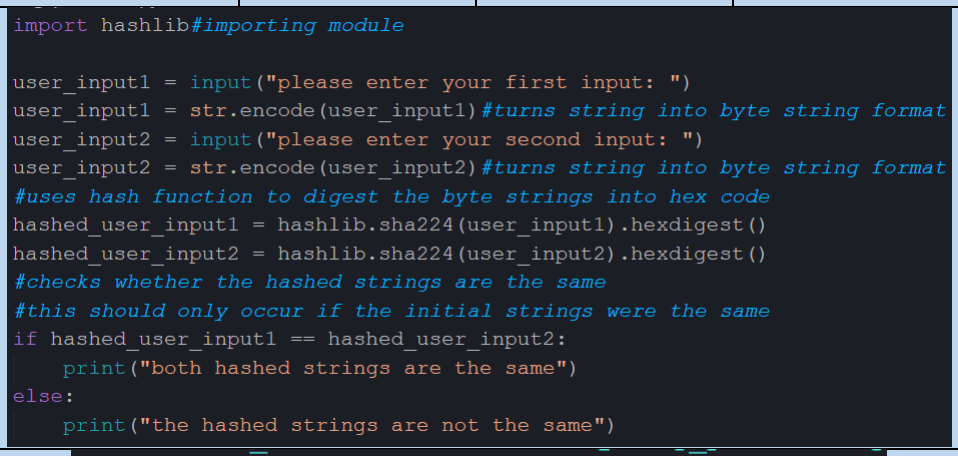
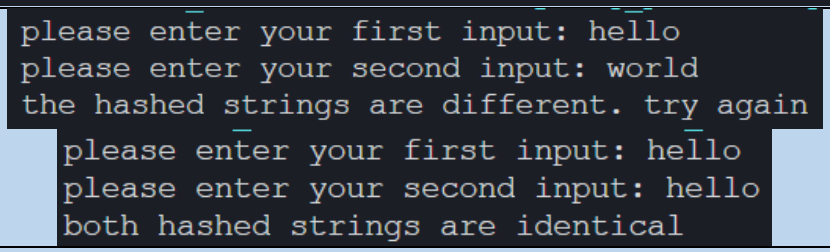
File Edit View Insert Format Data Tools Add-ons Help Last edit was seconds

	A	B	C	D	E	F
1	User number	Name	Username	Password	Email	Date joined
2	1	zain	zaina0k	hello world	blah@blah.com	22/10/2020
3	2	dan	danstg	goodbye	blah2@blah.com	23/10/2020
4	3	matt	matthew	bug	hello@gmail.com	23/10/2020
5						

Comments The program works as expected

Creating User Interface – Part 2 – Iteration 1

Creating User Interface Code – Part 2 – Iteration 1

Test	Description	Test Data	Justification	Expected Output
Event Test 1	Program should showcase the use of the hash function in python as a one-way encryption to store the user's password	"hello" and "world" Vs "hello" and "hello"	The program will encrypt the input from the user and check if the encrypted string matched the stored one to check whether the input was correct.	False for "hello" and "world. True for "hello" and "hello"
Test Code				
Result Screenshot				
Comments	The program works as expected			

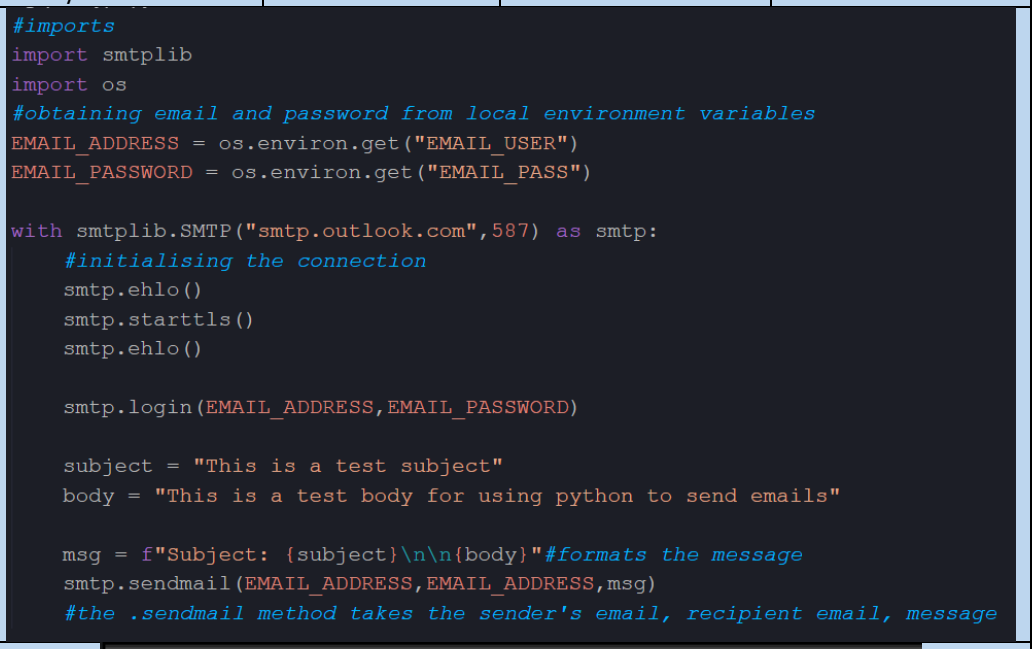
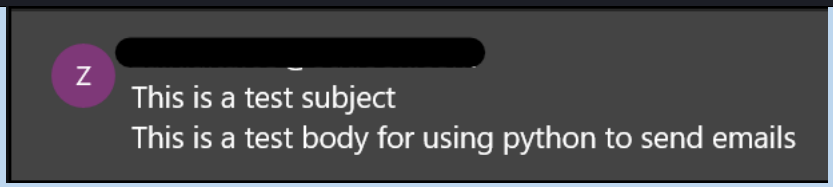
```
import hashlib#importing module

user_input1 = input("please enter your first input: ")
user_input1 = str.encode(user_input1)#turns string into byte string format
user_input2 = input("please enter your second input: ")
user_input2 = str.encode(user_input2)#turns string into byte string format
#uses hash function to digest the byte strings into hex code
hashed_user_input1 = hashlib.sha224(user_input1).hexdigest()
hashed_user_input2 = hashlib.sha224(user_input2).hexdigest()
#checks whether the hashed strings are the same
#this should only occur if the initial strings were the same
if hashed_user_input1 == hashed_user_input2:
    print("both hashed strings are the same")
else:
    print("the hashed strings are not the same")
```

```
please enter your first input: hello
please enter your second input: world
the hashed strings are different. try again
```

```
please enter your first input: hello
please enter your second input: hello
both hashed strings are identical
```

Creating User Interface – Part 3 – Iteration 1

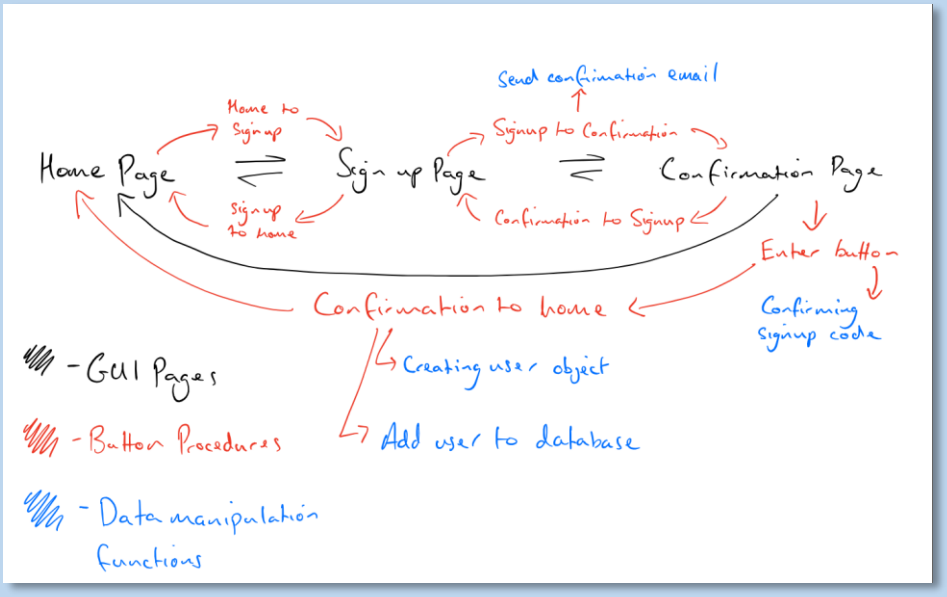
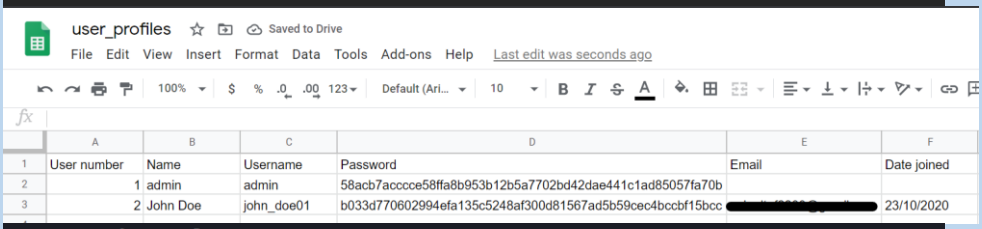
Creating User Interface Code – Part 3 – Iteration 1

Test	Description	Test Data	Justification	Expected Output
Event Test 1	The program should use the local environment variables to obtain an email and password used to send a message to the user via the smtplib library.	"This is a test subject" "This is a test body for using python to send emails"	This piece of code will be adapted into the project in order to send a code to the user to confirm the email/details for the account.	An email message should be sent via the local email to the user's email with the correct formatting.
Test Code				
Result Screenshot				
Comments	The program works as expected			

```python
#imports
import smtplib
import os
#obtaining email and password from local environment variables
EMAIL_ADDRESS = os.environ.get("EMAIL_USER")
EMAIL_PASSWORD = os.environ.get("EMAIL_PASS")

with smtplib.SMTP("smtp.outlook.com",587) as smtp:
    #initialising the connection
    smtp.ehlo()
    smtp.starttls()
    smtp.ehlo()

    smtp.login(EMAIL_ADDRESS,EMAIL_PASSWORD)

    subject = "This is a test subject"
    body = "This is a test body for using python to send emails"

    msg = f"Subject: {subject}\n\n{body}"#formats the message
    smtp.sendmail(EMAIL_ADDRESS,EMAIL_ADDRESS,msg)
    #the .sendmail method takes the sender's email, recipient email, message
```

Creating User Interface – Part 4 – Iteration 1

Creating User Interface Code – Part 4 – Iteration 1

Test	Description	Test Data	Justification	Expected Output
Event Test 1	This section encompasses the pieces of code used to create the user interface, such as creating a user object and sending an email to the user. This is all implemented into the GUI.	Name = John Doe Username = john_doe01 password = hello world email = admin email (will be blacked out)	Program must be able to take details from the user and check whether they are correct by validating them. Once validated the details should be stored in the online user database as well as in a global user object.	An email is received containing the confirmation code. Once entered the details are stored in the database and a user object is created.
Test Code				
Result Screenshot	This is a confirmation email for an account made for Sign Language Translator. Username: john_doe01 Please enter the following code to verify your account: 448395  user number: 2 name: John Doe username: john_doe01 password: b033d770602994efa135c5248af300d81567ad5b59cec4bccbf15bcc date joined: 23/10/2020 email: [redacted]			
Comments	The program works as expected – Output 1 shows the email sent to the user. Output 2 shows the updated online database. Output 3 shows the user object created in the program. The development of this algorithm was very problematic with the amounts of			

	attention to detail needed when coding the manipulation of the GUI objects. This cause many errors within the program, however, none were easily documentable due to the abstractness of the GUIZERO objects.

Creating User Interface – Part 4 – Iteration 2

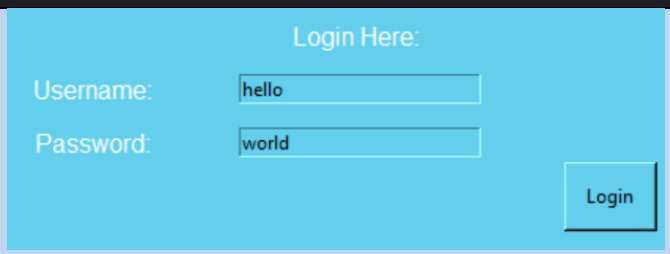
Creating User Interface Code – Part 4 – Iteration 2

Test	Description	Test Data	Justification	Expected Output
Event Test 1	This piece of code shows the function used to check whether there is a pre-existing user account that can be accessed by a returning user.	Username = hello Password = world	This is important as it allows user to end the program and save their progress to the online database and return to the program at a later time.	If there is a profile with the given user name and password the program should gather all the information for the profile and create a user object to store it.

Test Code

```
def existing_user_login(username,password):
    """Creates a user object from user database.
    Correct details are assured as username is unique(primary key)"""
    #refreshes the local database from the online database
    current_user = False
    for profile in total_users:
        if profile["Username"] == username:#looks for user profile
            init_password = str.encode(password)
            #encodes the password in the same way so that they match if inputs match
            hashed_password = hashlib.sha224(init_password).hexdigest().strip("0")
            if profile["Password"] == hashed_password:#if password is correct
                #obtains all the user information
                user_number = profile["User number"]
                name = profile["Name"]
                username = profile["Username"]
                password = profile["Password"]
                email = profile["Email"]
                date_joined = profile["Date joined"]
                started_TL = profile["Started tailored learning"]
                learning_path = profile["Learning path"]
                total_right = profile["Total right"]
                total_wrong = profile["Total wrong"]
                creating_user_object(name,username,password,email,
                user_number,date_joined,started_TL,learning_path)
                #creates a user object
                current_user = True
    return current_user
```

Result Screenshot



Comments

Extreme test failed – Program does not compensate for users that login in straight after signing up. This only causes and error if the user 1.signs up 2.logs out 3.tries to log back in. While the user's details have been added to the online database, the most recent version is not being accessed. Due to an outdated user database being used locally the program doesn't accept the user as it has no record of it.

Remedial Actions for Creating User Interface – Part 4 – Iteration 2

Creating User Interface – Part 4 – Iteration 3

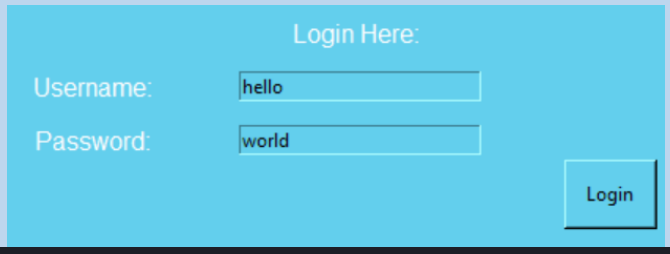
Creating User Interface Code – Part 4 – Iteration 3

Test	Description	Test Data	Justification	Expected Output
Event Test 1	This piece of code shows the function used to check whether there is a pre-existing user account that can be accessed by a returning user.	Username = hello Password = world	This is important as it allows user to end the program and save their progress to the online database and return to the program at a later time.	If there is a profile with the given user name and password the program should gather all the information for the profile and create a user object to store it.

Test Code

```
def existing_user_login(username,password):
    """Creates a user object from user database.
    Correct details are assured as username is unique(primary key)"""
    global total_users
    total_users = user_worksheet.get_all_records()
    #refreshes the local database from the online database
    current_user = False
    for profile in total_users:
        if profile["Username"] == username:#looks for user profile
            init_password = str.encode(password)
            #encodes the password in the same way so that they match if inputs match
            hashed_password = hashlib.sha224(init_password).hexdigest()
            if profile["Password"] == hashed_password:#if password is correct
                #obtains all the user information
                user_number = profile["User number"]
                name = profile["Name"]
                username = profile["Username"]
                password = profile["Password"]
                email = profile["Email"]
                date_joined = profile["Date joined"]
                started_TL = profile["Started tailored learning"]
                learning_path = profile["Learning path"]
                creating_user_object(name,username,password,email,
                user_number,date_joined,started_TL,learning_path)
                #creates a user object
                current_user = True
    return current_user
```

Result Screenshot




```
user number: 2
name: [redacted]
username: hello
password: 6d1c301ca37b272fe11e939600ea3335d7f69abd0e7b25f5e1d792ee
date joined: 31/10/2020
email: [redacted]
started tailored learning? No
current learning path: None
```

Comments

The program works as expected

Creating Tailored Learning - Part 1– Iteration 1

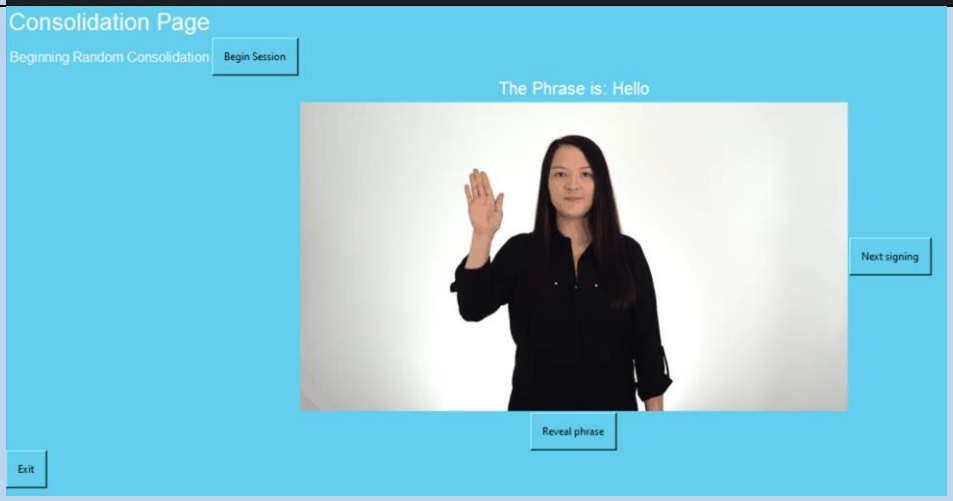
Creating Tailored Learning Code – Part 1 – Iteration 1

Test	Description	Test Data	Justification	Expected Output
Event Test 1	The program should create a random array of 10 phrases each time it is initialised. The program should iterate through and present each phrase and its description when needed.	BSL_database	The program needs to be able to present each phrase one at a time and allow the user to show the description at their discretion.	An array of 10 random phrases should be produced. Each phrase should be displayed alongside its description one by one.

Test Code

```
def iterate_consolidation_output():
    consolidation_gif_output.image = None
    consolidation_phrase_text.hide()
    consolidation_reveal_desc.hide()
    if len(consolidation_list)>1:
        consolidation_iterate_button.show()
        begin_practise_button.show()
    else:
        consolidation_iterate_button.hide()
        consolidation_finish_text.show()
    try:
        if not os.path.exists(consolidation_list[0]["Phrase"]+".gif"):
            #if video needs to be downloaded
            output_url = consolidation_list[0]["URL"]
            print(output_url)
            ytd = YouTube(output_url).streams.first().download(filename=consolidation_list[0]["Phrase"])
            print("Video downloaded to local directory")
            file_name = consolidation_list[0]["Phrase"] + ".mp4"
            clip = (VideoFileClip(file_name))#creates clip instance
            file_name = file_name.replace(".mp4",".gif")#changes the file type on string
            clip.write_gif(file_name)#converts .mp4 video to .gif
            consolidation_gif_output.image = file_name#outputs first value in list
            consolidation_gif_output.show()
            consolidation_reveal_desc.show()
            consolidation_phrase_text.value = "The Phrase is: " + consolidation_list[0]["Phrase"]
            del consolidation_list[0]#gets rid of the first value in list
        else:
            file_name = consolidation_list[0]["Phrase"]+".gif"
            consolidation_gif_output.image = file_name#outputs first value in list
            consolidation_gif_output.show()
            consolidation_reveal_desc.show()
            consolidation_phrase_text.value = "The Phrase is: " + consolidation_list[0]["Phrase"]
            del consolidation_list[0]#gets rid of the first value in list
    except:
        pass
```

Result Screenshot



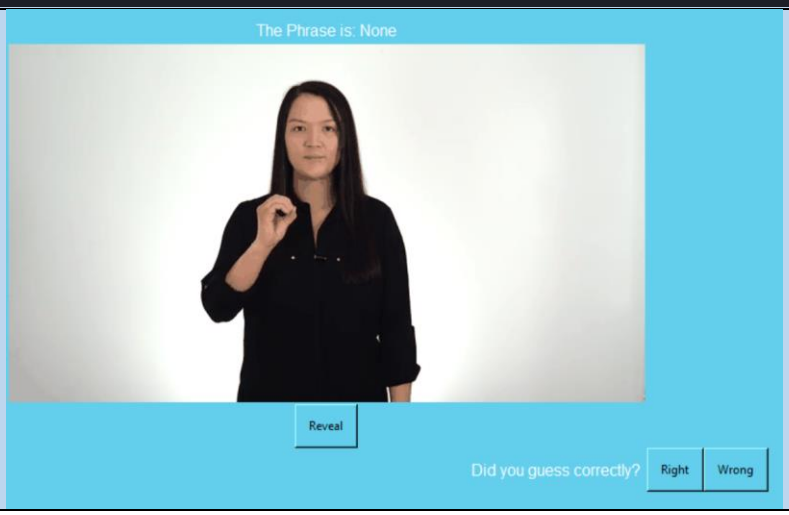
	[{'Phrase': 'Z', 'URL': 'https://www.youtube.com/watch?v=dZnUKw1vNK8&list=UUHctNnLPrEr9_MX2ZPtS7qQ&index=140'}, {'Phrase': 'Special', 'URL': 'https://www.youtube.com/watch?v=KKGPicFyWSE'}, {'Phrase': 'Through', 'URL': 'https://www.youtube.com/watch?v=MbdsyusVYTA'}, {'Phrase': 'Sister', 'URL': 'https://www.youtube.com/watch?v=vthJxWUMggw'}, {'Phrase': 'Wrong', 'URL': 'https://www.youtube.com/watch?v=5hMAtkizhHg'}, {'Phrase': 'Door Open', 'URL': 'https://www.youtube.com/watch?v=X4gvwSBRZOk'}, {'Phrase': 'Pants', 'URL': 'https://www.youtube.com/watch?v=bianxtwm_po'}, {'Phrase': 'Freckles', 'URL': 'https://www.youtube.com/watch?v=rNPALAURIxY'}, {'Phrase': 'White', 'URL': 'https://www.youtube.com/watch?v=I0fTR_0OCYQ'}, {'Phrase': 'Stand', 'URL': 'https://www.youtube.com/watch?v=hnrItxUcT5I'}, {'Phrase': 'Swimming', 'URL': 'https://www.youtube.com/watch?v=O2hWnnJxPuM'}, {'Phrase': 'O', 'URL': 'https://www.youtube.com/watch?v=GY
Comments	The program works as expected – output shows the GUI shown to the user as well as the array of 10 random phrases with the specific URL.

Creating Tailored Learning - Part 1– Iteration 2

Creating Tailored Learning Code - Part 1– Iteration 2

Test	Description	Test Data	Justification	Expected Output
Event Test 1	This piece of code should initialise the tailored learning by creating a list of phrases as a template in order to make a queue of phrases that are displayed.	An empty user object will be used showing the creation of a new template	The program must be able to create the template list in order to create the tailored list (queue of phrases). This would be copied from the user object if the user has previously done tailored learning.	A random set of 10 phrases should be made in a list. This should generate a list of phrases to be presented (queue) as a gif
Test Code				
Result Screenshot				

```
def tailored_predict_desc():
    """This function is used when the user wants to see the actions/.gif
    and wants to guess the phrase/description.
    This functtion initialises this specific tailored learning"""
    tailored_phrase_then_desc.hide()
    tailored_desc_then_phrase.hide()
    tailored_consolidation_options.hide()
    global predict_desc
    predict_desc = True
    #this dictates to other functions which type of tailred learning is being done
    global tailored_list
    del tailored_list[:]#this is the list for practised phrases (queue)
    global template_list
    del template_list[:]#this is a template used to set the queue of phrases
    template_list = create_template_list()
    for value in template_list:
        #the queue of phrases with tailored repeats is made from the template
        #phrases that the user is less confident in have more repeats
        for i in range(5-int(value["Confidence"])):
            tailored_list.append(value)
    shuffle(tailored_list)#queue is shuffled
    tailored_consolidation()#starting the consolidation
```



<table>
<tr><td></td><td>
<pre>
user number: 2
name: [redacted]
username: hello
password: 6d1c301ca37b272fe11e939600ea3335d7f69abd0e7b25f5e1d792ee
date joined: 31/10/2020
email: [redacted]
started tailored learning? Yes
current learning path: [{'Phrase': 'Me', 'URL': 'https://www.youtube.com/watch?v=ib5wls_aRbk', 'Confidence': 0}, {'Phrase': 'Waistcoat', 'URL': 'https://www.youtube.com/watch?v=3wZDsOn0lZM', 'Confidence': 0}, {'Phrase': 'Carrot', 'URL': 'https://www.youtube.com/watch?v=xwnywJ4C0fE', 'Confidence': 0}, {'Phrase': 'Quarter', 'URL': 'https://www.youtube.com/watch?v=CRLl6c94Mnw', 'Confidence': 0}, {'Phrase': 'Red', 'URL': 'https://www.youtube.com/watch?v=8r2mvKlM9Iw', 'Confidence': 0}, {'Phrase': 'None', 'URL': 'https://www.youtube.com/watch?v=z4xgh5UOikc', 'Confidence': 0}, {'Phrase': 'Poorly', 'URL': 'https://www.youtube.com/watch?v=qhYEfDK80tU', 'Confidence': 0}, {'Phrase': 'Airplane', 'URL': 'https://www.youtube.com/watch?v=gDsgZW0VogE', 'Confidence': 0}, {'Phrase': 'You', 'URL': 'https://www.youtube.com/watch?v=bkWuPat3mpc', 'Confidence': 0}, {'Phrase': 'Talk', 'URL': 'https://www.youtube.com/watch?v=AzBPbQo3FzA', 'Confidence': 0}]
</pre>
</td></tr>
<tr><td>Comments</td><td>The program works as expected - The first image shows the output of the phrase. This is similar to the usual consolidation page with the additional buttons for recording whether the user got it right or not (image shows the description revealed therefore showing the buttons to vote right or not). The second image shows the list created containing the 10 phrase values. Currently the list is stored as the user's learning path due to the template list being identical upon initiation.</td></tr>
</table>

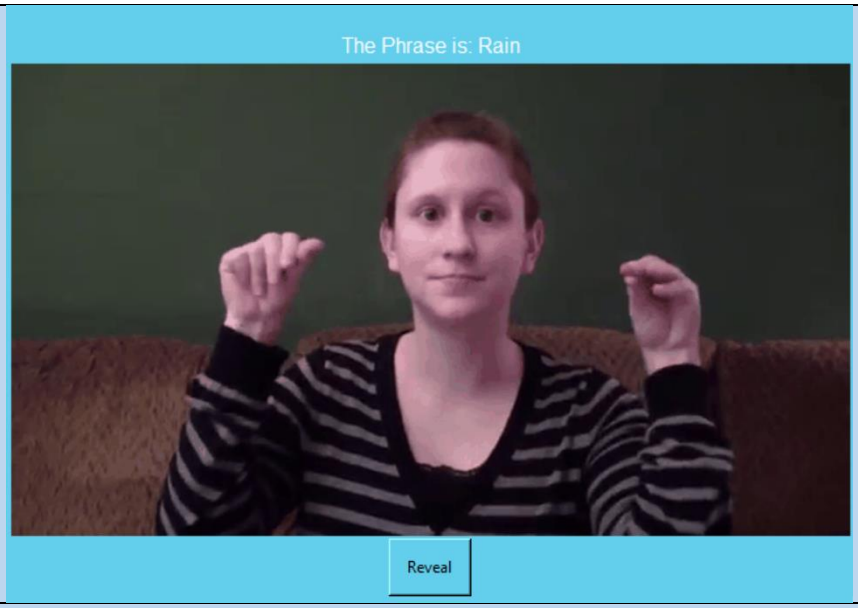
Creating Tailored Learning - Part 1– Iteration 3

Creating Tailored Learning Code - Part 1– Iteration 3

Test	Description	Test Data	Justification	Expected Output
Event Test 1	This piece of code should be able to add to the existing confidence system and determine when the user has "mastered" a phrase in order.	4 randomly selected phrases as a demo	Program needs to be able to adapt to the user in terms of them being able to consistently get a phrase right. This would imply they are confident with the phrase and therefore should be replaced with a new different phrase.	Confidence should change after each user input. When a phrase reaches a confidence of 5 it should be replaced with a new phrase

Test Code

```
def correct_tailored_consolidation():
    global template_list
    global tailored_list
    global user_object
    user_object.total_right += 1
    for i in range(len(template_list)):
        if template_list[i]["Phrase"] == tailored_list[0]["Phrase"]:
            template_list[i]["Confidence"] += 1
            #increases confidence in template list for specific phrase that user got right
            if int(template_list[i]["Confidence"]) == 5:#if user has "mastered" the phrase
                removed = False
                while not removed:
                    #loops through and removes all other of that phrase from tailored_list (queue)
                    edited = False
                    for i in range(len(tailored_list)):
                        if tailored_list[i]["Phrase"] == template_list[i]["Phrase"]:
                            edited=True
                            del tailored_list[i]
                            break
                    if not edited:
                        removed = True
                temp_var = template_list[i]
                user_object.learning_path.append(temp_var)
                #appends mastered value to the user object to be stored
                del template_list[i]
                #deletes the value from the template list

                for k in range(len(total_vids)):#loops through the the video database
                    random_phrase = total_vids[k]
                    for complete_phrase in user_object.learning_path:
                        #loops through the users consolidation history
                        if complete_phrase["Phrase"] == random_phrase["Phrase"]:
                            #checks if the phrases are the same
                            if k == len(total_vids)-1:#if its the last phrase
                                random_phrase = choice(total_vids)
                                #uses a random phrase for the next phrase
                        else:
                            break#uses the next available phrase
                    break
                random_phrase.update({"Confidence":0})
                #inserts confidence key:value into the dictionary
                template_list.append(random_phrase)
                #adds the new phrase to be learnt to the template list
                user_object.learning_path.append(random_phrase)
                #adds the new phrase to be learnt to the user's history
                for i in range(5):
                    tailored_list.append(random_phrase)
                    #adds the new phrase to the practise list - to be practised 5 times
                shuffle(tailored_list)
                #shuffles the list so the same phrases aren't adjacent
            break
    tailored_feedback_text.show()
    tailored_correct_button.hide()
    tailored_incorrect_button.hide()
    tailored_feedback_text.hide()
    tailored_iteration_button.show()
    tailored_reveal.hide()
```

Result Screenshot	
Comments	Test failed – test repeatedly shows the same phrase after pressing the correct button multiple times. The GUI seems stuck on the specific phrase

Remedial Actions for Creating Tailored Learning - Part 1– Iteration 3

Creating Tailored Learning - Part 1– Iteration 4

Creating Tailored Learning Code - Part 1– Iteration 4

Test	Description	Test Data	Justification	Expected Output
Event Test 1	This piece of code should be able to add to the existing confidence system and determine when the user has "mastered" a phrase in order.	4 randomly selected phrases as a demo	Program needs to be able to adapt to the user in terms of them being able to consistently get a phrase right. This would imply they are confident with the phrase and therefore should be replaced with a new different phrase.	Confidence should change after each user input. When a phrase reaches a confidence of 5 it should be replaced with a new phrase

Test Code

```
def correct_tailored_consolidation():
    global template_list
    global tailored_list
    global user_object
    user_object.total_right += 1
    for i in range(len(template_list)):
        if template_list[i]["Phrase"] == tailored_list[0]["Phrase"]:
            template_list[i]["Confidence"] += 1
            #increases confidence in template list for specific phrase that user got right
            if int(template_list[i]["Confidence"]) == 5:#if user has "mastered" the phrase
                removed = False
                while not removed:
                    #loops through and removes all other of that phrase from tailored_list (queue)
                    edited = False
                    for i in range(len(tailored_list)):
                        if tailored_list[i]["Phrase"] == template_list[i]["Phrase"]:
                            edited=True
                            del tailored_list[i]
                            break
                    if not edited:
                        removed = True
                temp_var = template_list[i]
                user_object.learning_path.append(temp_var)
                #appends mastered value to the user object to be stored
                del template_list[i]
                #deletes the value from the template list

                for k in range(len(total_vids)):#loops through the the video database
                    random_phrase = total_vids[k]
                    for complete_phrase in user_object.learning_path:
                        #loops through the users consolidation history
                        if complete_phrase["Phrase"] == random_phrase["Phrase"]:
                            #checks if the phrases are the same
                            if k == len(total_vids)-1:#if its the last phrase
                                random_phrase = choice(total_vids)
                                #uses a random phrase for the next phrase
                            else:
                                break#uses the next available phrase
                        break
                random_phrase.update({"Confidence":0})
                #inserts confidence key:value into the dictionary
                template_list.append(random_phrase)
                #adds the new phrase to be learnt to the template list
                user_object.learning_path.append(random_phrase)
                #adds the new phrase to be learnt to the user's history
                for i in range(5):
                    tailored_list.append(random_phrase)
                    #adds the new phrase to the practise list - to be practised 5 times
                shuffle(tailored_list)
                #shuffles the list so the same phrases aren't adjacent
            else:
                del tailored_list[0]
                #deletes the most phrase that was answered right from the queue
            break
    tailored_feedback_text.show()
    tailored_correct_button.hide()
    tailored_incorrect_button.hide()
    tailored_feedback_text.hide()
    tailored_iteration_button.show()
    tailored_reveal.hide()
```

Result Screenshot	`if tailored_list[i]["Phrase"] == template_list[i]["Phrase"]:` `IndexError: list index out of range`
Comments	Test failed – program fails to promote phrases to "mastered" (confidence level 5). On trying to do this the program throws an index out of range error.

Remedial Actions for Creating Tailored Learning - Part 1– Iteration 4

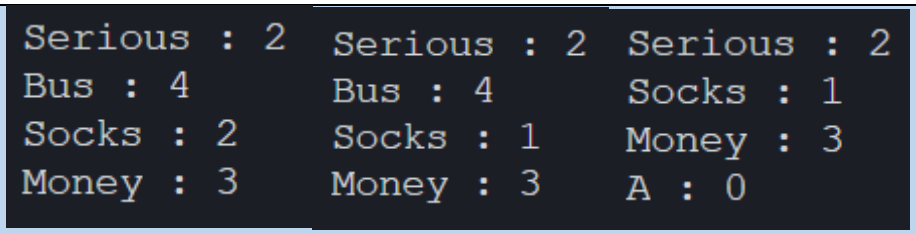
Creating Tailored Learning - Part 1– Iteration 5

Creating Tailored Learning Code - Part 1– Iteration 5

Test	Description	Test Data	Justification	Expected Output
Event Test 1	This piece of code should be able to add to the existing confidence system and determine when the user has "mastered" a phrase in order.	4 randomly selected phrases as a demo	Program needs to be able to adapt to the user in terms of them being able to consistently get a phrase right. This would imply they are confident with the phrase and therefore should be replaced with a new different phrase.	Confidence should change after each user input. When a phrase reaches a confidence of 5 it should be replaced with a new phrase

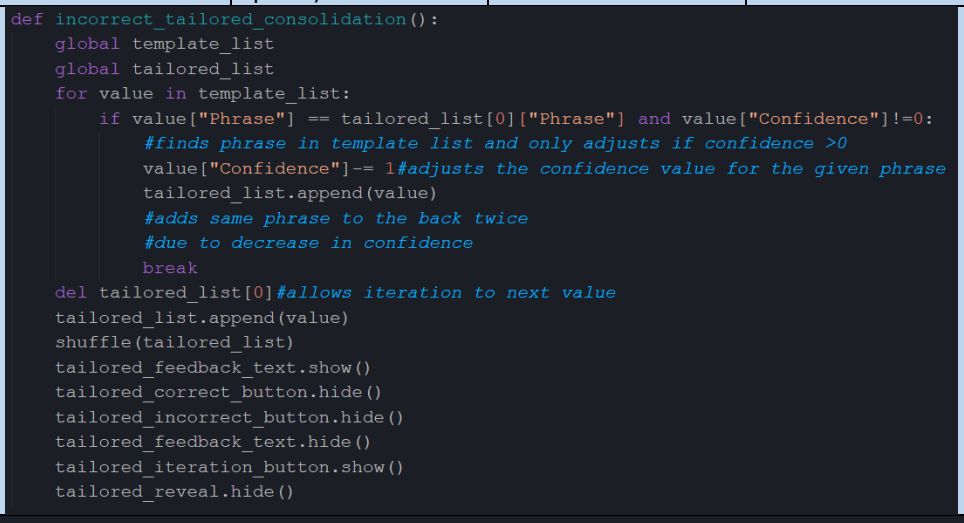
Test Code

```
def correct_tailored_consolidation():
    global template_list
    global tailored_list
    for i in range(len(template_list)):
        if template_list[i]["Phrase"] == tailored_list[0]["Phrase"]:
            template_list[i]["Confidence"] += 1
            #increases confidence in template list for specific phrase that user got right
            if int(template_list[i]["Confidence"]) == 5:#if user has "mastered" the phrase
                removed = False
                while not removed:
                    #loops through and removes all other of that phrase from tailored_list (queue)
                    edited = False
                    for j in range(len(tailored_list)):
                        if tailored_list[j]["Phrase"] == template_list[i]["Phrase"]:
                            edited=True
                            del tailored_list[j]
                            break
                    if not edited:
                        removed = True
                temp_var = template_list[i]
                user_object.learning_path.append(temp_var)
                #appends mastered value to the user object to be stored
                del template_list[i]
                #deletes the value from the template list

                for k in range(len(total_vids)):#loops through the the video database
                    random_phrase = total_vids[k]
                    for complete_phrase in user_object.learning_path:
                        #loops through the users consolidation history
                        if complete_phrase["Phrase"] == random_phrase["Phrase"]:
                            #checks if the phrases are the same
                            if k == len(total_vids)-1:#if its the last phrase
                                random_phrase = choice(total_vids)
                                #uses a random phrase for the next phrase
                        else:
                            break#uses the next available phrase
                    break
                random_phrase.update({"Confidence":0})
                #inserts confidence key:value into the dictionary
                template_list.append(random_phrase)
                #adds the new phrase to be learnt to the template list
                user_object.learning_path.append(random_phrase)
                #adds the new phrase to be learnt to the user's history
                for i in range(5):
                    tailored_list.append(random_phrase)
                    #adds the new phrase to the practise list - to be practised 5 times
                shuffle(tailored_list)
                #shuffles the list so the same phrases aren't adjacent
            else:
                del tailored_list[0]
                #deletes the most phrase that was answered right from the queue
            break
    tailored_feedback_text.show()
    tailored_correct_button.hide()
    tailored_incorrect_button.hide()
    tailored_feedback_text.hide()
    tailored_iteration_button.show()
    tailored_reveal.hide()
```

Result Screenshot	
Comments	The program works as expected – this output shows the three iterations of user input and the changes it has on the number values (the confidence rating of each phrase). The pictures show that only 1 numerical change can occur at one time. As “bus” goes from 4 to 5, it is replaced by “a” as a phrase with a confidence of 0.

Creating Tailored Learning - Part 1– Iteration 6

Creating Tailored Learning Code - Part 1– Iteration 6

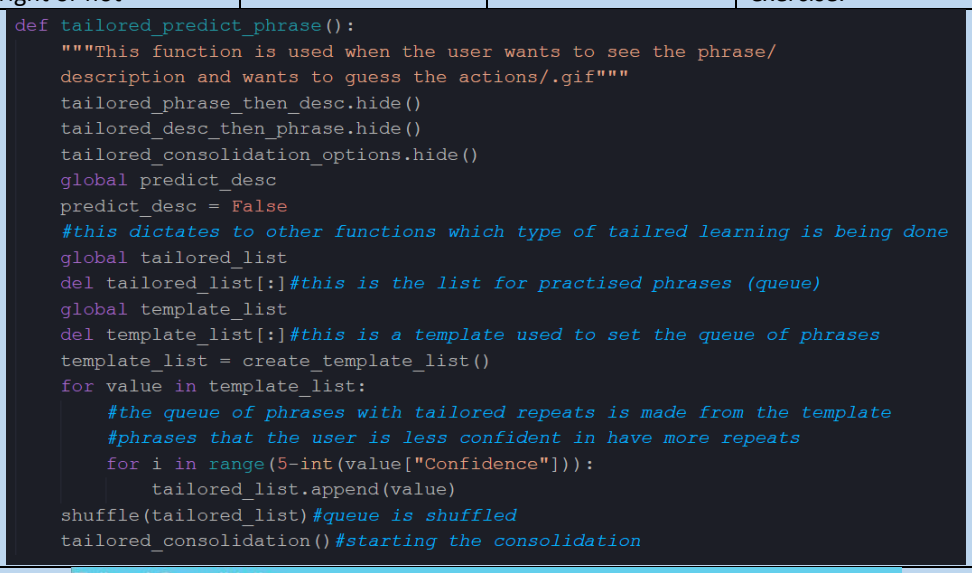
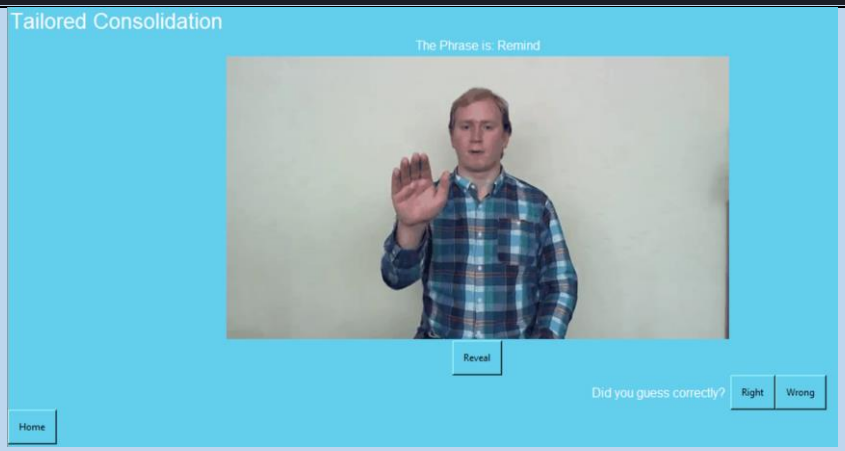
Test	Description	Test Data	Justification	Expected Output
Event Test 1	Program should edit the confidence value for a given phrase in the template list by reducing the value. This should change (increase) the number of that particular phrase presented to the user	Program will use 2 random phrases in a template list. a larger list containing multiple iterations of each phrase is created from the template (this is the tailored_list or queue)	The program will use 2 phrases in template instead of the usual 10 to demonstrate the change in the number of values in the tailored_list (queue).	Program should increase the number of the given phrase and decrease its confidence value.
Test Code				
Result Screenshot				
Comments	The program works as expected			

```
def incorrect_tailored_consolidation():
    global template_list
    global tailored_list
    for value in template_list:
        if value["Phrase"] == tailored_list[0]["Phrase"] and value["Confidence"]!=0:
            #finds phrase in template list and only adjusts if confidence >0
            value["Confidence"]-= 1#adjusts the confidence value for the given phrase
            tailored_list.append(value)
            #adds same phrase to the back twice
            #due to decrease in confidence
            break
    del tailored_list[0]#allows iteration to next value
    tailored_list.append(value)
    shuffle(tailored_list)
    tailored_feedback_text.show()
    tailored_correct_button.hide()
    tailored_incorrect_button.hide()
    tailored_feedback_text.hide()
    tailored_iteration_button.show()
    tailored_reveal.hide()
```

```
Who : 2
Waitrose : 2

[{'Phrase': 'Waitrose', 'URL': 'https://www.youtube.com/watch?v=N3JD9FF6P64', 'Confidence': 2}, {'Phrase': 'Waitrose', 'URL': 'https://www.youtube.com/watch?v=N3JD9FF6P64', 'Confidence': 2}, {'Phrase': 'Waitrose', 'URL': 'https://www.youtube.com/watch?v=N3JD9FF6P64', 'Confidence': 2}, {'Phrase': 'Who', 'URL': 'https://www.youtube.com/watch?v=x_Svt0onT7A', 'Confidence': 2}, {'Phrase': 'Who', 'URL': 'https://www.youtube.com/watch?v=x_Svt0onT7A', 'Confidence': 2}, {'Phrase': 'Who', 'URL': 'https://www.youtube.com/watch?v=x_Svt0onT7A', 'Confidence': 2}]

Who : 2
Waitrose : 1

[{'Phrase': 'Waitrose', 'URL': 'https://www.youtube.com/watch?v=N3JD9FF6P64', 'Confidence': 1}, {'Phrase': 'Who', 'URL': 'https://www.youtube.com/watch?v=x_Svt0onT7A', 'Confidence': 2}, {'Phrase': 'Who', 'URL': 'https://www.youtube.com/watch?v=x_Svt0onT7A', 'Confidence': 2}, {'Phrase': 'Waitrose', 'URL': 'https://www.youtube.com/watch?v=N3JD9FF6P64', 'Confidence': 1}, {'Phrase': 'Who', 'URL': 'https://www.youtube.com/watch?v=x_Svt0onT7A', 'Confidence': 2}, {'Phrase': 'Waitrose', 'URL': 'https://www.youtube.com/watch?v=N3JD9FF6P64', 'Confidence': 1}]
```

sCreating Tailored Learning - Part 1– Iteration 7

Creating Tailored Learning Code - Part 1– Iteration 7

Test	Description	Test Data	Justification	Expected Output
Event Test 1	Program will begin by showing the description of a phrase. Once the user has chosen to reveal the signing action (gif) the user will be able to input feedback as to whether they got it right or not	A random tailored_list (queue) is created. The shown phrase is rewind	The user can choose which method of practise they wish and it should work in the same way. This provides more freedom through more options.	The program should show the description of the phrase first and allow the user to reveal the gif. The functionality should be identical to the "Gif to description" consolidation exercise.
Test Code				
Result Screenshot				
Comments	The program works as expected			

```
def tailored_predict_phrase():
    """This function is used when the user wants to see the phrase/
    description and wants to guess the actions/.gif"""
    tailored_phrase_then_desc.hide()
    tailored_desc_then_phrase.hide()
    tailored_consolidation_options.hide()
    global predict_desc
    predict_desc = False
    #this dictates to other functions which type of tailred learning is being done
    global tailored_list
    del tailored_list[:]#this is the list for practised phrases (queue)
    global template_list
    del template_list[:]#this is a template used to set the queue of phrases
    template_list = create_template_list()
    for value in template_list:
        #the queue of phrases with tailored repeats is made from the template
        #phrases that the user is less confident in have more repeats
        for i in range(5-int(value["Confidence"])):
            tailored_list.append(value)
    shuffle(tailored_list)#queue is shuffled
    tailored_consolidation()#starting the consolidation
```

Creating Setting – Iteration 1

Creating Setting Code – Iteration 1

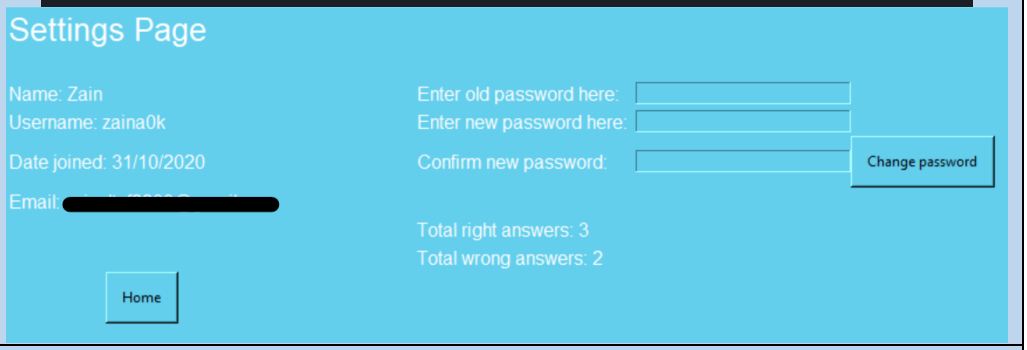
Test	Description	Test Data	Justification	Expected Output
Event Test 1	Program should take all values given to it during sign up and display them to the user. this should be displayed alongside the tailored learning stats.	A pre-made user object will be used with pre-established total right/wrong values.	The program should allow the user to see what about them is being stored.	Relevant data from the user object should be transferred to the GUI.

Test Code

```
def loggedin_to_settings():
    home_loggedin_page.hide()
    settings_page.show()
    adjusted_name = user_object.name.title()
    settings_name.value = "Name: " + str(adjusted_name)
    settings_username.value = "Username: " + str(user_object.username)
    settings_date_joined.value = "Date joined: " + str(user_object.date_joined)
    settings_email.value = "Email: " + str(user_object.email)
    settings_total_right.value = "Total right answers: " + str(user_object.total_right)
    settings_total_wrong.value = "Total wrong answers: " + str(user_object.total_wrong)
```

Result Screenshot

```
user number: 2
name: zain
username: zaina0k
password: b033d770602994efa135c5248af300d81567ad5b59cec4bccbf15bcc
date joined: 31/10/2020
email: [redacted]
started tailored learning? Yes
current learning path:
total right: 3
total wrong: 2
```



Comments The program works as expected

Creating Setting – Iteration 2

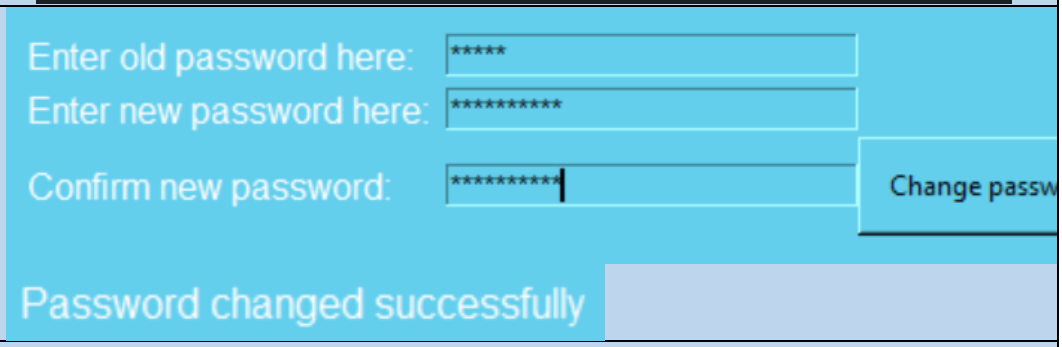
Creating Setting Code – Iteration 2

Test	Description	Test Data	Justification	Expected Output
Event Test 1	The program should allow the user to change their password by entering their old password and new password. The new password should be updated to the online user database.	Old password: world New password: helloworld	The user needs to be able to use the new password when accessing at a different time after being logged out. This means the new password must not be stored locally and therefore must be stored in the online database.	"Password changed successfully"

Test Code

```
def change_password():
    global total_users
    global user_object
    total_users = user_worksheet.get_all_records()
    password = settings_old_password_input.value
    init_password = str.encode(password)
    hashed_password = hashlib.sha224(init_password).hexdigest().strip("0")
    both_equal = False
    for i in range(len(total_users)):
        if total_users[i]["Password"] == hashed_password:
            if settings_new_password_input.value == settings_confirm_password_input.value:
                both_equal = True
                new_password = str.encode(settings_new_password_input.value)
                hashed_new_password = hashlib.sha224(new_password).hexdigest().strip("0")
                user_object.password = hashed_new_password
                updating_row()
                settings_old_password_input.value = ""
                settings_new_password_input.value = ""
                settings_confirm_password_input.value = ""
                settings_password_error.value = "Password changed successfully"

        elif i == len(total_users)-1:
            settings_password_error.value = "Please enter the correct original password"
    if not both_equal:
        settings_password_error.value = "Please enter the same new password"
```

Result Screenshot



Comments

The program works as expected

3.3.2b Provide annotated evidence of any remedial actions taken justifying the decision made

Functional Chunk 1 – Searching Specific Phrases

Return to Contents Page

Top 20 Recently Searched Phrases Remedial Action– Part 2 – Iteration 1:

```
my_list = ['hello', 'world', 'this', 'is', 'a', 'test']

def top_20_format(my_list):
    """function formats the lists so that they can be more easily displayed.
    creates a 2d list with up to 4 values in each of the inner lists."""
    list_1 = []
    for i in range(len(my_list)):
        if i %4 == 0:#if the index of the list is a multiple of 4 - this includes 0
            list_1.append([])#creates new inner list
            list_1[-1].append(my_list[i])#adds value to the back-most inner list
        else:
            list_1[-1].append(my_list[i])#adds value to the back-most inner list
    print(list_1)
```

Changes:

- 1 holder list instead of 2 – reduces error whilst transferring data
- Modulus function – used to dictate when to add the new list and value

Top 20 Recently Searched Phrases code – Part 2 – Iteration 1

Testing for Top 20 Recently Searched Phrases – Part 2 – Iteration 1

Alphabet Line Remedial Action - Part 2 – Iteration 1

```
def alphabet_display(letter):#beginning letter passed through
    """function takes a letter and creates a list of all the phrases beginning
    with that letter from the database"""
    del letter_list[:]#list is cleared before initiation so values are not added to a full list
    for phrase in database:
        if phrase[0] == letter:#if beginning letters match
            letter_list.append(phrase)#adds it to the list
    output_string = top_20_format(letter_list)#formats the list for a top 20
    alphabet_output.value = output_string
    return letter_list
```

Changes:

- Letter_list cleared before reinitialization of new letter list – happens at beginning of alphabet_display function from Part 1. The highlighted line of code allows new letter lists to be started from the beginning and therefore outputted correctly as well as allowing clearing the output before restarting a finished letter list

Testing for Alphabet Line – Part 2 – Iteration 1

Alphabet Line Code – Part 2 – Iteration 1

Microphone Input Remedial Action – Iteration 1:

```
import audiomath; audiomath.RequireAudiomathVersion( '1.12.0' )
import speech_recognition

class DuckTypedMicrophone( speech_recognition.AudioSource ):
    # descent from AudioSource is required purely to pass an assertion in Recognizer.listen()
    def __init__( self, device=None, chunkSeconds=1024/44100.0 ):
        # 1024 samples at 44100 Hz is about 23 ms
        self.recorder = None
        self.device = device
        self.chunkSeconds = chunkSeconds
    def __enter__( self ):
        self.nSamplesRead = 0
        self.recorder = audiomath.Recorder( audiomath.Sound( 5, nChannels=1 ), loop=True, device=self.device )
        # Attributes required by Recognizer.listen():
        self.CHUNK = audiomath.SecondsToSamples( self.chunkSeconds, self.recorder.fs, int )
        self.SAMPLE_RATE = int( self.recorder.fs )
        self.SAMPLE_WIDTH = self.recorder.sound.nbytes
        return self
    def __exit__( self, *blx ):
        self.recorder.Stop()
        self.recorder = None
    def read( self, nSamples ):
        sampleArray = self.recorder.ReadSamples( self.nSamplesRead, nSamples )
        self.nSamplesRead += nSamples
        return self.recorder.sound.dat2str( sampleArray )
    @property
    def stream( self ):
        # attribute must be present to pass an assertion in Recognizer.listen(),
        #and its value must have a .read() method
        return self if self.recorder else None

# source must be an instance of a speech_recognition.AudioSource subclass
# source.stream must be non-None while the source is active
# source.CHUNK must be the (integer) number of samples per chunk
# source.SAMPLE_RATE must be the sampling rate
# source.SAMPLE_WIDTH must be the number of bytes per sample
# source.stream.read(numberOfSamples) must return raw single-channel audio data
```

Changes:

- Microphone method of the speech_recognition module is swapped out for DuckTypedMicrophone code – this code works as the original microphone code is supposed to. This substitute module was found on Stack overflow : (https://stackoverflow.com/questions/55984129/attributeerror-could-not-find-pyaudio-check-installation-cant-use-speech-re). Last accessed: 01/09/20.
- More about these changes can be found in the prototyping section here: Microphone Prototype

Microphone Input Code – Iteration 1

Testing for Microphone Input – Iteration 1

Outputting a Phrase Remedial Action – Part 3 – Iteration 1:

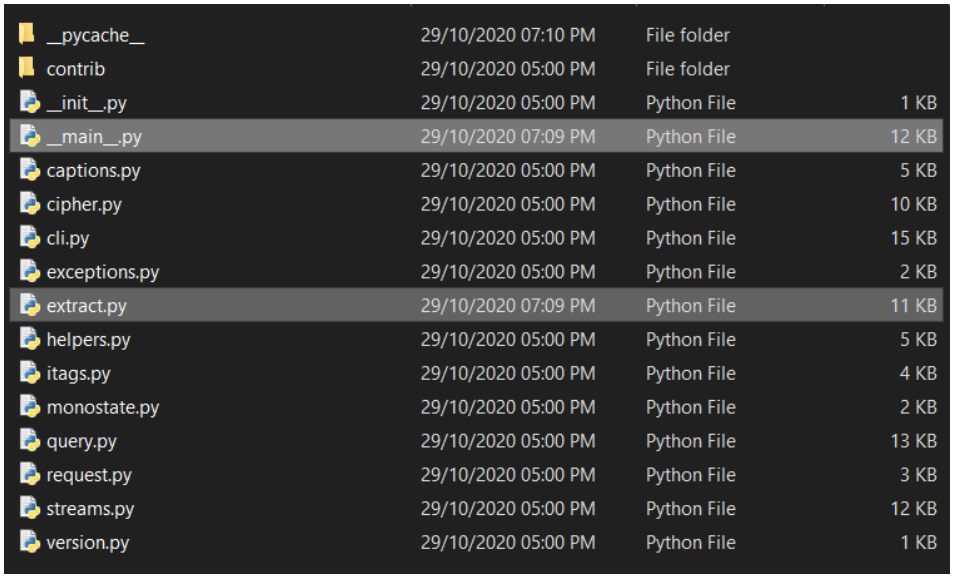
The pytube library used to download the video from YouTube has been working intermittently due to problems within the module. Upon further inspection the following error message is returned by the program.

```
    youtube = pytube.YouTube(URL)
  File "C:\Users\zaina\AppData\Roaming\Python\Python37\site-packages\pytube\__main__.py", line 91, in __init__
    self.prefetch()
  File "C:\Users\zaina\AppData\Roaming\Python\Python37\site-packages\pytube\__main__.py", line 183, in prefetch
    self.js_url = extract.js_url(self.watch_html)
  File "C:\Users\zaina\AppData\Roaming\Python\Python37\site-packages\pytube\extract.py", line 143, in js_url
    base_js = get_ytplayer_config(html)["assets"]["js"]
KeyError: 'assets'
```

By researching the error, I found that others had also experienced this problem and that the issue was with the pytube module and not my programming project. My main sources of information regarding fixing the issue came from stack overflow and GitHub. Results from stackoverflow initially called for changes within the extract.py file within the file containing the library. However, this proved to be inconsequential as the problems persisted. Other people also detailed that the fix hadn't entirely worked and so a link was provided to the GitHub page containing a more reliable temporary fix. This fix also requires adjustments to be made to the extract.py file.

The first step was to uninstall the pytube module and instead use the pytube3 module. This is a fork of the pytube module and will eventually be reintegrated together to form 1 library.

Once this was complete the developer posted the changes they had made to the individual modules in the library. In order to obtain the changes, I visited the GitHub code and replaced the local files of the library with the edited ones. This included the __main__.py and extract.py modules.



__pycache__	29/10/2020 07:10 PM	File folder	
contrib	29/10/2020 05:00 PM	File folder	
__init__.py	29/10/2020 05:00 PM	Python File	1 KB
__main__.py	29/10/2020 07:09 PM	Python File	12 KB
captions.py	29/10/2020 05:00 PM	Python File	5 KB
cipher.py	29/10/2020 05:00 PM	Python File	10 KB
cli.py	29/10/2020 05:00 PM	Python File	15 KB
exceptions.py	29/10/2020 05:00 PM	Python File	2 KB
extract.py	29/10/2020 07:09 PM	Python File	11 KB
helpers.py	29/10/2020 05:00 PM	Python File	5 KB
itags.py	29/10/2020 05:00 PM	Python File	4 KB
monostate.py	29/10/2020 05:00 PM	Python File	2 KB
query.py	29/10/2020 05:00 PM	Python File	13 KB
request.py	29/10/2020 05:00 PM	Python File	3 KB
streams.py	29/10/2020 05:00 PM	Python File	12 KB
version.py	29/10/2020 05:00 PM	Python File	1 KB

This was described by the creator as a temporary fix until the edits are implemented into the main pytube library during integration of the fork programs.

Outputting Phrase Code – Part 3 – Iteration 1

Testing for Outputting Phrase – Part 3 – Iteration 1

Outputting a Phrase Remedial Action – Part 4 – Iteration 2:

```
def gif_setup():
    """this function finds the signings of the specific phrases in online database"""
    #obtains corrected user input
    corrected_input = ""
    for i in range(len(output_text.value)):
        if output_text.value[i] == ":":
            corrected_input = output_text.value[i+2:]
            break
    #sets intro text value
    search_result_user_input.value = "Searching for: " + corrected_input
    clear_search_page()
    del search_output_file_names[:]#clears the global list
    user_input_list,found_values = create_url_list(corrected_input.lower())
    #obtains the total phrases found in the input and a 2d list containing
    #each word in the input and a pointer to a value in the found_values list
    temp_list = []
    counter = 0
    while counter < len(user_input_list):
        #this loops round the 2d list and abstracts all of the phrase descriptions
        #by looking at each of the indexes in found_values list
        if type(user_input_list[counter][1]) is list:
            for j in range(len(user_input_list[counter][1])):
                search_output_file_names.append(found_values[user_input_list[counter][1][j]])
            counter+=1
        else:
            search_output_file_names.append(found_values[user_input_list[counter][1]])
            counter+=len(found_values[user_input_list[counter][1]].split())
    iterate_search_result()#iterates once
```

Changes:

The problem with the second iteration was that it did not take into account the list containing multiple indexes for each letter within the user_input_list. A separate if statement checking if the value is a list first allows the program to extract each of the indexes and process them in the same way as it does the others. The else statement then leads the program if the value is for a single phrase.

Outputting a Phrase Code – Part 4 – Iteration 2

Testing for Outputting a Phrase – Part 4 – Iteration 2

Functional Chunk 2 – Searching Through Database

Back to Contents Page

Functional Chunk 3 – Consolidation of Learning

Back to Contents Page

Creating User Interface Remedial Action – Part 4 – Iteration 2

```
def existing_user_login(username,password):
    """Creates a user object from user database.
    Correct details are assured as username is unique(primary key)"""
    global total_users
    total_users = user_worksheet.get_all_records()
    #refreshes the local database from the online database
    current_user = False
    for profile in total_users:
        if profile["Username"] == username:#looks for user profile
            init_password = str.encode(password)
            #encodes the password in the same way so that they match if inputs match
            hashed_password = hashlib.sha224(init_password).hexdigest()
            if profile["Password"] == hashed_password:#if password is correct
                #obtains all the user information
                user_number = profile["User number"]
                name = profile["Name"]
                username = profile["Username"]
                password = profile["Password"]
                email = profile["Email"]
                date_joined = profile["Date joined"]
                started_TL = profile["Started tailored learning"]
                learning_path = profile["Learning path"]
                creating_user_object(name,username,password,email,
                user_number,date_joined,started_TL,learning_path)
                #creates a user object
                current_user = True
    return current_user
```

Changes:

Here the main change made to allow the program to log in to the given profile was to re-obtain the user data from the online database. To ensure that this data was then up to date everywhere else in the program and NOT stored locally within the function, the “global” keyword is used. This means that the total_users list is up to date for the entire program at that point and allows the program to successfully find recently created users.

Creating User Interface Code – Part 4 – Iteration 2

Testing for Creating User Interface – Part 4 – Iteration 2

Creating Tailored Learning Remedial Action- Part 1– Iteration 3

```
def correct_tailored_consolidation():
    global template_list
    global tailored_list
    global user_object
    user_object.total_right += 1
    for i in range(len(template_list)):
        if template_list[i]["Phrase"] == tailored_list[0]["Phrase"]:
            template_list[i]["Confidence"] += 1
            #increases confidence in template list for specific phrase that user got right
            if int(template_list[i]["Confidence"]) == 5:#if user has "mastered" the phrase
                removed = False
                while not removed:
                    #loops through and removes all other of that phrase from tailored_list (queue)
                    edited = False
                    for i in range(len(tailored_list)):
                        if tailored_list[i]["Phrase"] == template_list[i]["Phrase"]:
                            edited=True
                            del tailored_list[i]
                            break
                    if not edited:
                        removed = True
                temp_var = template_list[i]
                user_object.learning_path.append(temp_var)
                #appends mastered value to the user object to be stored
                del template_list[i]
                #deletes the value from the template list

                for k in range(len(total_vids)):#loops through the the video database
                    random_phrase = total_vids[k]
                    for complete_phrase in user_object.learning_path:
                        #loops through the users consolidation history
                        if complete_phrase["Phrase"] == random_phrase["Phrase"]:
                            #checks if the phrases are the same
                            if k == len(total_vids)-1:#if its the last phrase
                                random_phrase = choice(total_vids)
                                #uses a random phrase for the next phrase
                            else:
                                break#uses the next available phrase
                        break
                random_phrase.update({"Confidence":0})
                #inserts confidence key:value into the dictionary
                template_list.append(random_phrase)
                #adds the new phrase to be learnt to the template list
                user_object.learning_path.append(random_phrase)
                #adds the new phrase to be learnt to the user's history
                for i in range(5):
                    tailored_list.append(random_phrase)
                    #adds the new phrase to the practise list - to be practised 5 times
                shuffle(tailored_list)
                #shuffles the list so the same phrases aren't adjacent
            else:
                del tailored_list[0]
                #deletes the most phrase that was answered right from the queue
            break
    tailored_feedback_text.show()
    tailored_correct_button.hide()
    tailored_incorrect_button.hide()
    tailored_feedback_text.hide()
    tailored_iteration_button.show()
    tailored_reveal.hide()
```

Changes:

The change made to the code was adding an else statement after the condition on whether the confidence value was equal to 5. This will most of the time not be the case and so to allow the list to iterate to the next value, the first value of the tailored_list (the value just shown) is deleted. This was at first thought to be a problem within the presentation of the gif in the GUI.

Creating Tailored Learning Code - Part 1– Iteration 3

Testing for Creating Tailored Learning - Part 1– Iteration 3

Creating Tailored Learning Remedial Action- Part 1– Iteration 4

```
def correct_tailored_consolidation():
    global template_list
    global tailored_list
    for i in range(len(template_list)):
        if template_list[i]["Phrase"] == tailored_list[0]["Phrase"]:
            template_list[i]["Confidence"] += 1
            #increases confidence in template list for specific phrase that user got right
            if int(template_list[i]["Confidence"]) == 5:#if user has "mastered" the phrase
                removed = False
                while not removed:
                    #loops through and removes all other of that phrase from tailored_list (queue)
                    edited = False
                    for j in range(len(tailored_list)):
                        if tailored_list[j]["Phrase"] == template_list[i]["Phrase"]:
                            edited=True
                            del tailored_list[j]
                            break
                    if not edited:
                        removed = True
                temp_var = template_list[i]
                user_object.learning_path.append(temp_var)
                #appends mastered value to the user object to be stored
                del template_list[i]
                #deletes the value from the template list

                for k in range(len(total_vids)):#loops through the the video database
                    random_phrase = total_vids[k]
                    for complete_phrase in user_object.learning_path:
                        #loops through the users consolidation history
                        if complete_phrase["Phrase"] == random_phrase["Phrase"]:
                            #checks if the phrases are the same
                            if k == len(total_vids)-1:#if its the last phrase
                                random_phrase = choice(total_vids)
                                #uses a random phrase for the next phrase
                        else:
                            break#uses the next available phrase
                        break
                random_phrase.update({"Confidence":0})
                #inserts confidence key:value into the dictionary
                template_list.append(random_phrase)
                #adds the new phrase to be learnt to the template list
                user_object.learning_path.append(random_phrase)
                #adds the new phrase to be learnt to the user's history
                for i in range(5):
                    tailored_list.append(random_phrase)
                    #adds the new phrase to the practise list - to be practised 5 times
                shuffle(tailored_list)
                #shuffles the list so the same phrases aren't adjacent
            else:
                del tailored_list[0]
                #deletes the most phrase that was answered right from the queue
            break
    tailored_feedback_text.show()
    tailored_correct_button.hide()
    tailored_incorrect_button.hide()
    tailored_feedback_text.hide()
    tailored_iteration_button.show()
    tailored_reveal.hide()
```

Changes:

The change was made to the looping mechanism when hitting the if statement for a phrase with a confidence value of 5. Here a for loop is used. In the previous iteration the for loop uses the variable i to loop. This clashes with the variable i used to iterate through the template list. To solve this a different variable must be used, allowing the program to loop through the tailored list (queue) and remove all of the values of the phrase that has just been mastered. This means that the mastered phrase will not come up for the user due to their confidence in it.

Creating Tailored Learning Code - Part 1– Iteration 4

Testing for Creating Tailored Learning - Part 1– Iteration 4

3.4 Evaluation

3.4.1 Testing to inform evaluation

3.4.1a Provide annotated evidence of testing the solution of robustness at the end of the development process

This section will focus on testing the program against the success criteria made at the beginning of the document in 3.1.4b

Tests in this section:

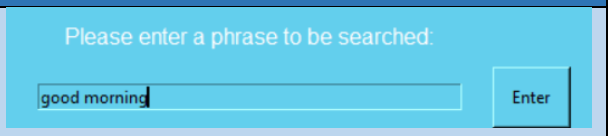
- SC 1 – Searching a phrase
- SC 2 – Searching a sentence
- SC 3 – Inputting invalid phrases
- SC 4 – Creating a user profile
- SC 5 – Voice search
- SC 6 – Check daily progress
- SC 7 – Change the time frame on progress achieved
- SC 8 – Attempting to create a second user with the same details as another

SC 1

Number	Success criteria	Justification	Test
SC 1	User will be able to search a phrase and get the result for that phrase.	It is important part of the program that the user is able to search for phrases that they do not know how to sign.	Search the phrase "good morning". This should output the corresponding signing action labelled "good morning".

This success criteria ensures that the user is able to search for a phrase to sign and obtains a viable result.

1. Can the user enter a phrase?
2. Does the program return an output of the signing?
3. How does the program process inaccurate inputs?

Number	Test	Expected Result	Actual Result
SC 1.1	Checks whether the program has an appropriate way for the user to input a phrase – input "good morning"	The program should provide user with a text box in order to input their phrase into.	
Test Passed?	Test Passed		

Number	Test	Expected Result	Actual Result
SC 1.2	Checks whether the program returns a signing phrase – "good morning"	The program should present the user with a gif representing the signing action for the given phrase	See Video SC 1-2
Test Passed?	Test Passed		

Number	Test	Expected Result	Actual Result
SC 1.3	Checks how the program responds to a slightly incorrect value inputted as a phrase – "good morning" (slightly misspelt)	Any slight abnormalities will cause the program to present the next most applicable phrase to be searched.	See Video 1-3
Test Passed?	Test Passed		

SC 2

Number	Success criteria	Justification	Test
SC 2	User will be able to search for a sentence and get the result for the sentence	It is important part of the program that the user is able to search for sentence that they do not know how to sign and receive the signing for the full sentence	Search the sentence: "good morning hello world". This should output the corresponding signing for the whole sentence – all of the sentence that can be found.

This success criteria tests that the user can search for a sentence as an input. A sentence should be interpreted as a sequence of phrases and so the user should be shown a sequence of signings in the same order.

1. Can the user enter a sentence – is it validated
2. Does the program output the sentence in the same sequence it was entered?

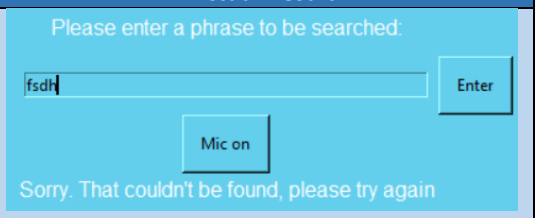
Number	Test	Expected Result	Actual Result
SC 2.1	Checks whether the user can enter a sentence as a sequence of phrases – "good morning hello world"	The program should present the user with the most high level phrases possible to present the sentence in the least number of signings (this should show "good morning" as 1 output)	See Video SC 2-1
Test Passed?	Test Passed		

Number	Test	Expected Result	Actual Result
SC 2.2	Checks whether the program outputs the sentence as a sequence of signing in the corresponding order	Program should present to the user the highest level (least number of gifs) signing outputs for the sentence. The sequence of the words should remain unaltered.	See Video SC 2-2
Test Passed?	Test Passed		

SC 3

Number	Success criteria	Justification	Test
SC 3	Phrases that cannot be found should output an error message.	There may be phrases that are not in the database. On the other hand, the phrase may not be able to be found due to miss spelling, etc.	Search the string “fsdh”. This should output an error message telling the user it could not be found

This success criteria tests that the user cannot input a random string of letters and receive an output.

Number	Test	Expected Result	Actual Result
SC 3.1	Checks whether the program will reject a random string of letters inputted to obtain a phrase – “fsdh”	Program should present an error message to the user if a non-standard word is inputted into the search engine.	
Test Passed?	Test Passed		

SC 4

Number	Success criteria	Justification	Test
SC 4	User will be able to create a user profile with a username and password.	To ensure that the user's data can be stored and kept for the next time.	Create the following user: First name: "John" Last name: "Doe" Username: "john_doe01" Password: "Hello_world12" Fluency: "beginner" After making the user, try and log out then log back in again. Check to make sure all the details are the same.

This success criteria tests that the user will be able to create a user profile within the program. This user profile will be able to store and easily access data about the user.

1. Can the user enter all of their details?
2. Is the user's password encrypted?
3. Can the user log out and log back in?

Number	Test	Expected Result	Actual Result
SC 4.1	Checks whether the program allows the user to enter all of the shown details in order to create a user profile within the program – First name: "John" Last name: "Doe" Username: "john_doe01" Password: "Hello_world12" Fluency: "beginner"	Program will take the values: first name, last name, username, password and fluency and create a user profile with it. The user will then be able to log into that profile whenever they wish.	Program will not take all of the shown values. This is due to different values being taken by the program. The user will be asked to input a: name, username, password and email for confirmation. 
Test Passed?	Test Passed		

Number	Test	Expected Result	Actual Result

SC 4.2	Checks whether the program encrypts the user's password – "Hello_world12" as user's password.	The program should store the users password in an encrypted form using a 1-way encryption.	Users password is stored using 1-way encryption.
			`user number: 3`
			`name: John Doe`
			`username: john_doe01`
			`password: db5d19b11af850c14c5e95a7b6a50146fee811c934c519ba9256b273`
			`date joined: 01/11/2020`

Test Passed?	Test Passed

Number	Test	Expected Result	Actual Result
SC 4.3	Checks whether the user can log back into a premade account	Program should take the username and password and check whether they correspond to a premade account. If the account is found the user is logged in and their data is retrieved.	See video SC 4-3
Test Passed?	Test Passed		

SC 5

Number	Success criteria	Justification	Test
SC5	User will be able to conduct a voice search.	Being able to search for a phrase via spoken English was a key piece of functionality and served as the basis of the program which was built upon. Being able to search via voice directly ties into the user's ability to directly link a signing action with the audio counterpart of that phrase. This is all in the service of further the learning of the user.	Click the microphone icon to start the voice search and say "good morning" into the microphone. The search result should display the corresponding signing action for the phrase labelled "good morning"

This success criteria tests that the user can input a phrase into the program via speaking. This allows the user to input phrases via a mic instead of typing.

Number	Test	Expected Result	Actual Result
SC 5.1	Checks whether the program allows the user to input a phrase via the mic.	The program listens to user input via the mic and translates the audio into text to be searched via the search engine.	See video SC 5-1
Test Passed?	Test Passed		

SC 6

Number	Success criteria	Justification	Test
SC 6	User should be able to check their daily… progress that they made.	It is important for users to understand their progress for new skills as well as old skills. Showing progress not only reinforces the user's achievements but also shows them where they may need improvement. Furthermore, a progress map also encourages users to continue using the resource in order to keep a daily progress and keep improving.	Using the "john_doe01" user, attempt the "learning option" and randomly click "I got it right" and "I got it wrong" options for the shown phrases. Once satisfied, navigate to the "daily progress" page to view the progress that was completed. This should be an easy to read source of information in the form of a graph/chart.

This success criteria tests that the user can check their daily progress to see how well they are progressing. This would use the data from the user's profile, collected during practise sessions.

Number	Test	Expected Result	Actual Result
SC 6.1	Checks whether the program allows the user to see the progress made on a daily basis	Program should allow the user to view data stored in their user profile based on data collected from the consolidation of learning section. This data should be presented in graph form.	This was not implemented into the program due to a continuous looping mechanism used instead. Here the program uses a confidence system with spaced repetition to provide the user with a greater learning experience. This however does not take any values daily from the consolidation process. The program keeps a total figure of the number of right and number of wrong answers.
Test Passed?	Test Failed		

SC 7

Number	Success criteria	Justification	Test
SC 7	Users will be able to change the time frame of the progress data that they wish to see.	Adding on to SC 8 the user also needs to be able to change the time frame allowing them to see a greater view of their progress. This is important in all users but more useful to users that use the program more over a longer period of time.	Again, using the "john_doe01" user, navigate to the progress page. Switch between the time frame settings from "daily" to "weekly" and "monthly". There should not be any data here yet, but some data should be shown from the progress made beforehand.

This success criteria tests that the user can check their daily progress to see how well they are progressing. This would use the data from the user's profile, collected during practise sessions.

Number	Test	Expected Result	Actual Result
SC 7.1	Checks whether the program allows the user to change the time frame of progress so that they can see their progress on a larger scale	Program should use a slider or mechanism of sorts to allow the user to view their progress data over a varying period of time.	This was not implemented into the project due to the fact that the data was not date stamped when stored. Instead a total value for number of right answers and number of wrong answers were stored.
Test Passed?	Test Failed		

SC 8

Number	Success criteria	Justification	Test
SC 8	Users should not be able to create two accounts with the same username but the program should still accept users with the same first and last name.	Two users may try and create an account with the same username. This may cause confusion or even errors in the program and so should be prohibited. However, the program should still accept users with the same first and last name by using the username as the unique attribute.	Attempt to make the exact same user that was made SC 4 again. The program should output that the username is already taken. With all the details the same, attempt to remake the SC 4 user with the username "john_doe02" instead. The program should allow a brand-new user to be created.

This success criteria tests whether a user can input details into the signup that are identical to another user's details. This tests the robustness of the program and how it responds to data that could cause problems within the program if not validated correctly.

1. Can the user enter the same details as another user?
2. Which details need to be different between users?

Number	Test	Expected Result	Actual Result
SC 8.1	Checks whether the program allows the user to enter the same details of another user/themselves.	Program should check the other users and see whether the values correspond to another user. If they do the program should output an error message.	See video 8-1
Test Passed?	Test Passed		

Number	Test	Expected Result	Actual Result
SC 8.2	Checks what pieces of data need to be adjusted to allow for a similar user to be made	Program should allow a user to be made with any name and password as long as the username and email are unique	See video 8-2
Test Passed?	Test Passed		

3.4.1b Provide annotated evidence of usability testing (user feedback)

3.4.2 Success of the Solution

3.4.2a Use the test evidence from the development and post development process to evaluate the solution against the success criteria from the analysis

Using the success criteria in 3.4.1a I found that my final project had met 6 out of the 8 success criteria successfully. While this is fairly successful in its own right, it is important to note that the success criteria were created before the development of the program and were therefore subject to be overruled as less important aspects to the overall program than other pieces of functionality that took higher priority. The success criteria also focus largely on the initial parts of the project and therefore do not cover the full extent of the solution to the problem itself. In the following table I will explore each success criteria individually and in more detail.

Number	Success criteria	Evaluation of development	Further comments – What to improve and how…
SC 1	User will be able to search a phrase and get the result for that phrase.	While this seems initially simple, the difficulty of reaching this criterion came from the framework of functionality built around it. This included creating the GUI with the multiple interacting object. The hardest part of development came from using an online database to store the phrases and links for each phrase. This required me to find, understand and use multiple external modules.	This was an essential part of the dictionary functionality within the program. The ambiguity within this criterion means that while it has been met, there are many things that could make it better such as having multiple sign language databases. This could be done by having a separate online database with phrases and URLs for a different sign language. While this was thought of originally, it requires a significant amount of time to create a sizable database for another sign language.
SC 2	User will be able to search for a sentence and get the result for the sentence	Once the first success criterion was met this criterion was much easier since a lot of the code could be modified to be reused for this situation. The hardest part of this criteria was developing the algorithm to find the highest level phrases within the sentence so that the least number of outputs needed to be displayed. E.g. “good morning” would find the phrase “good morning” instead of showing “good” and “morning” separately	This worked well in the final program and continues the same pattern of SC 1 with its complexity able to change with the frameworks of the program. While the algorithm itself is very efficient the process of forming a sentence could be improved with phrases in the database being longer and more high level. E.g having more phrases with 2+ words such as “good morning”.
SC 3	Phrases that cannot be found should output an error message.	In order to do this the user’s input went through a process of its viability. Once non alphabet characters were gone, the input was comparted (word by word) against an entire dictionary of phrases for the most similar one. This meant that any minor inaccuracies may be solved. However, if the input was not similar enough to anything in the dictionary an error message would be displayed.	Using a word bank of phrases allowed the program to incorporate a light autocorrect feature. This could be made better by having a larger wordbank (possibly multiple languages) in order to process a greater number of inputs.

SC 4	User will be able to create a user profile with a username and password.	This was possible through the use of OOP. Once the local user object had been created, the hard part of development came from uploading the data stored within the object to the online user database and back whilst keeping the formatting the same. During development it was decided that the nature of the final program did not require a “fluency” value to be stored and was therefore removed from signup.	This again follows the same trend of complexity, with its added complexity coming from the online upload and email confirmation. Currently a trial version of both these modules are being used and are therefore limited. During development and personal use this has been fine but if more requests are demanded a higher subscription level would be necessary.
SC 5	User will be able to conduct a voice search.	This piece of functionality was the first hard part of development due to its problems within the module used. This required me to work out how to solve the problem, in this case a local replacement function.	This piece of functionality was static in its complexity once its problems were solved. However, the viability of its use was limited to how quite a room was in order for it to work properly. The user can improve the use of this function by using a headset microphone in a quiet room. An imporovement to the transcription accuracy would require me to find a different speech recognition API other than google’s.
SC 6	User should be able to check their daily… progress that they made.	This success criterion was not reached during the development process. This was due to the fact that other pieces of code took higher priority.	After development I still believe this to be a viable piece of functionality and with more time would be implemented into the project
SC 7	Users will be able to change the time frame of the progress data that they wish to see	“	“
SC 8	Users should not be able to create two accounts with the same username but the program should still accept users with the same first and last name.	The development of this piece of functionality was quite simple once the framework of the user system had been developed. The signup process also includes validation from input within the GUI in a few ways including email verification code and checking for repeat username and emails.	The idea that the username would be a unique identifier between users was an idea kept throughout development.

Works and is implemented well

Does not work/not included

3.4.3 Describe the final project

3.4.3a Provide annotated evidence of the usability features from the design, commenting on their effectiveness

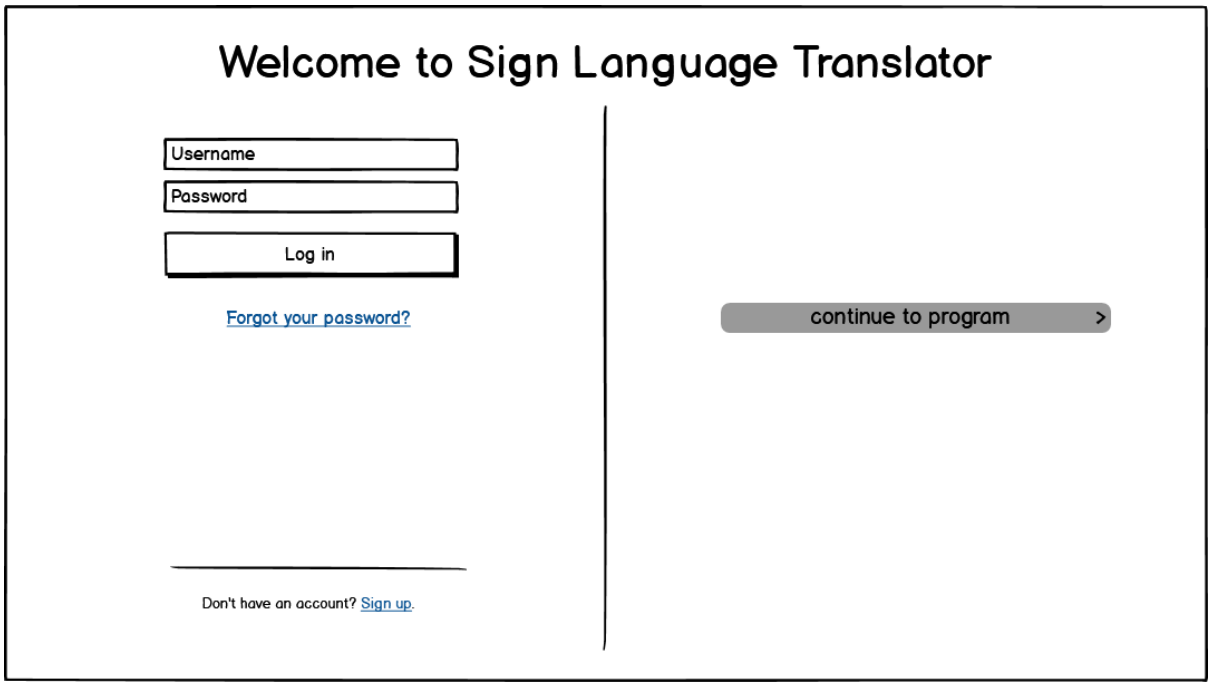
My program contains many points of interaction within the GUI and with therefore be organised by the page in which it is found. This will be compared to the pages found in section 3.2.2c.

The developed pages include:

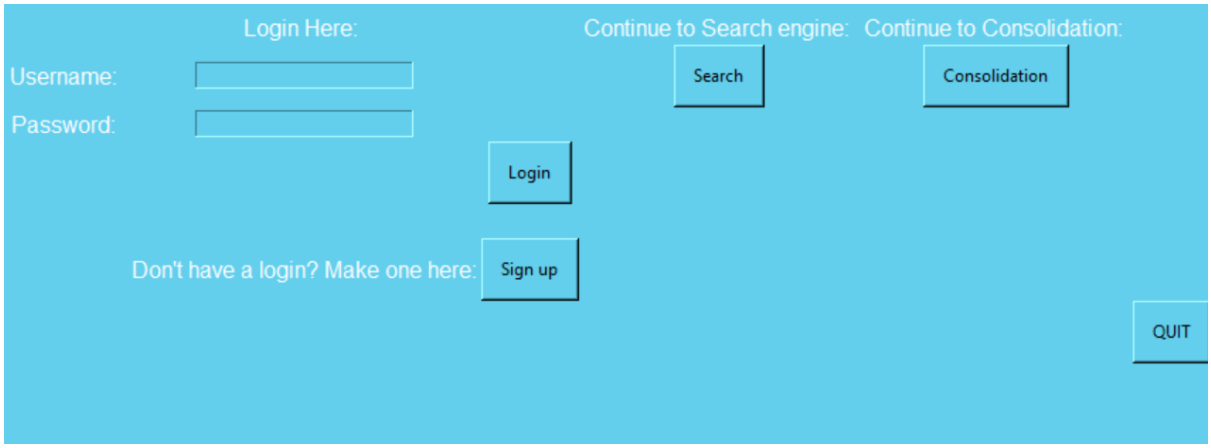
- Home Page
- Logged in Home page
- Sign up page
- Search page
- Consolidation page
- Tailored consolidation page
- Settings page

Home page

Initial Design:



Actual Design:



Both designs boast a formal and tidy design, however, the initial design shows a more appealing simple design with access to different types of buttons and design aspects that guizero cannot provide.

Username and password:

The design between the initial and actual pages are quite similar here. Both have text boxes for the user to enter data. Both have a button to press in order to initiate the login. All of the objects are clearly labelled for the user allowing for ease of use in both designs.

Signup option:

Both initial and actual designs contain the signup button which allows the user to create a user account. Neither of the designs allow the user to input their details to sign up on the home page but instead provide a button to another page in which the signup would occur. This is effective as it saves there from being more clutter while offering the functionality.

Consolidation and search buttons:

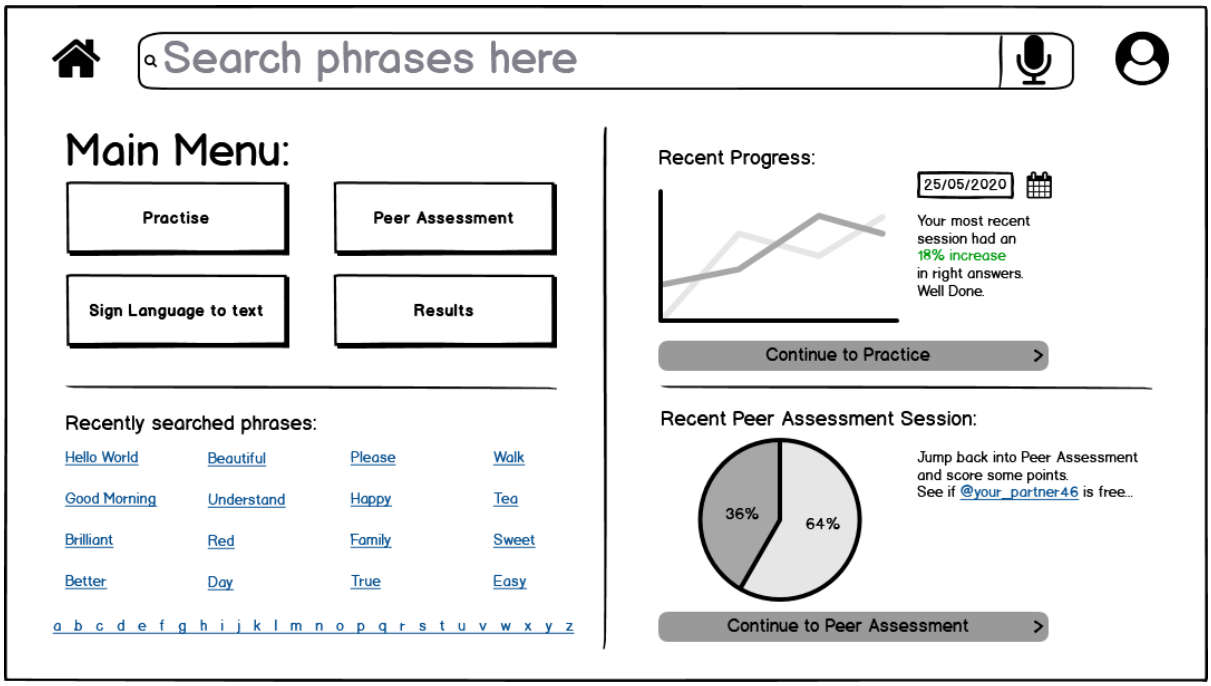
These button were placed on the home page with the knowledge that any user that has not signed in would be able to use the features. This effectively provides easier access to the functionality is removes a page of buffer from the initial design where the buttons would have been found on the next page.

Quit button:

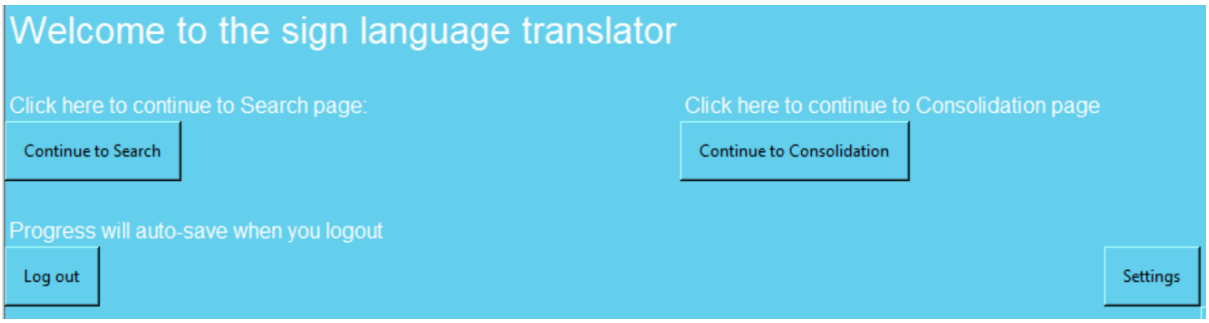
The actual design incorporates a quit button within the program for the purpose of exiting the program from the home page. This is effective as it establishes a base point for where the user starts and ends their journey in the program.

Logged in Home page

Initial Design:



Actual Design:



This page shows the biggest difference between the initial aesthetic and the actual design. This is mainly due to the lack of further functionality shown in the initial design.

Search button:

Button and description are simple and effective at directing user to the search page.

Consolidation:

Button and description are simple and effective at directing user to the consolidation page.

Logout:

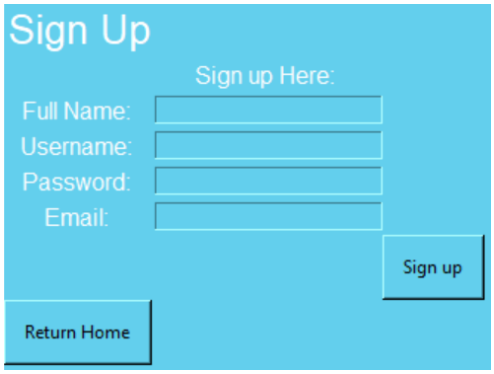
Logout button alongside information for the user saying their progress will be saved automatically when logging out is overall simple and effective.

Settings:

Button and description are simple and effective at directing user to the settings page.

Sign up page

Sign up - Actual Design



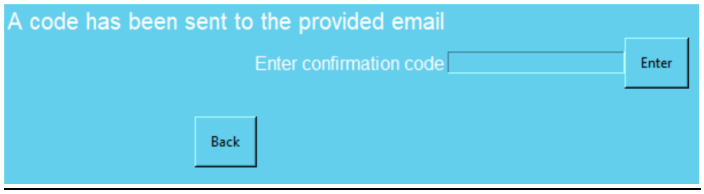
User input:

The 4 textboxes each have their own descriptor telling the user what value needs to go where. A simple and effective system that allows the user to input their data and sign up.

Home button:

A home button is added just in case a user mis-clicks and does not mean to create an account.

Confirmation - Actual Design

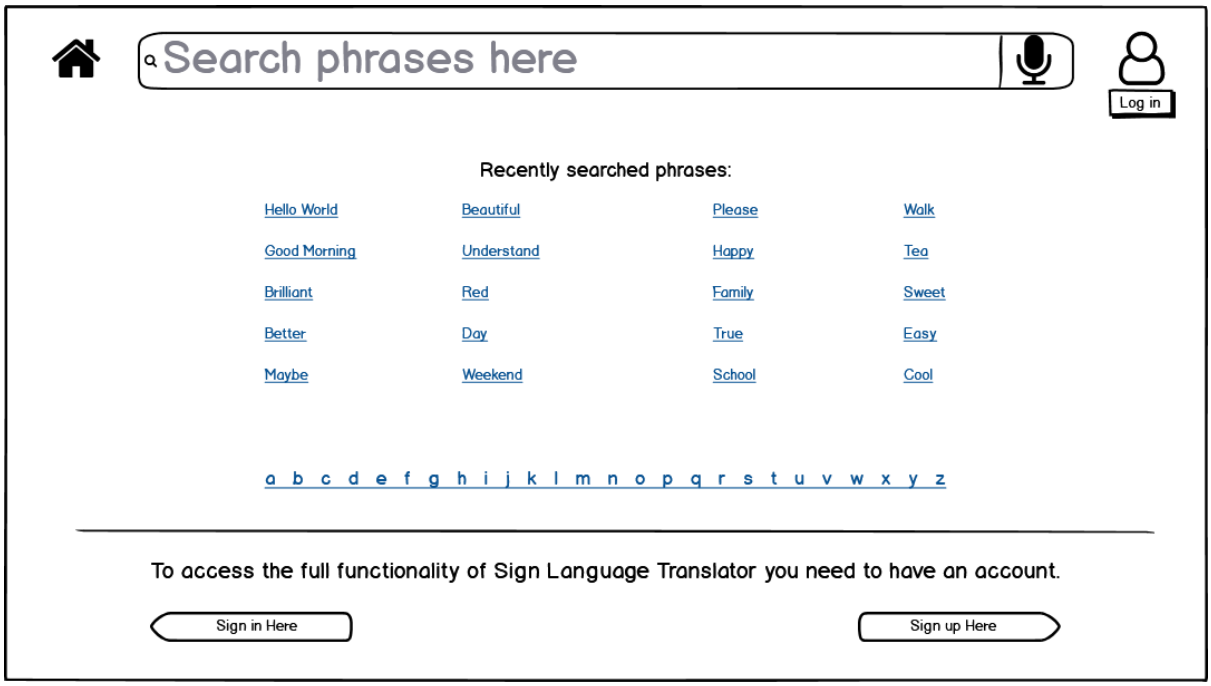


Confirmation:

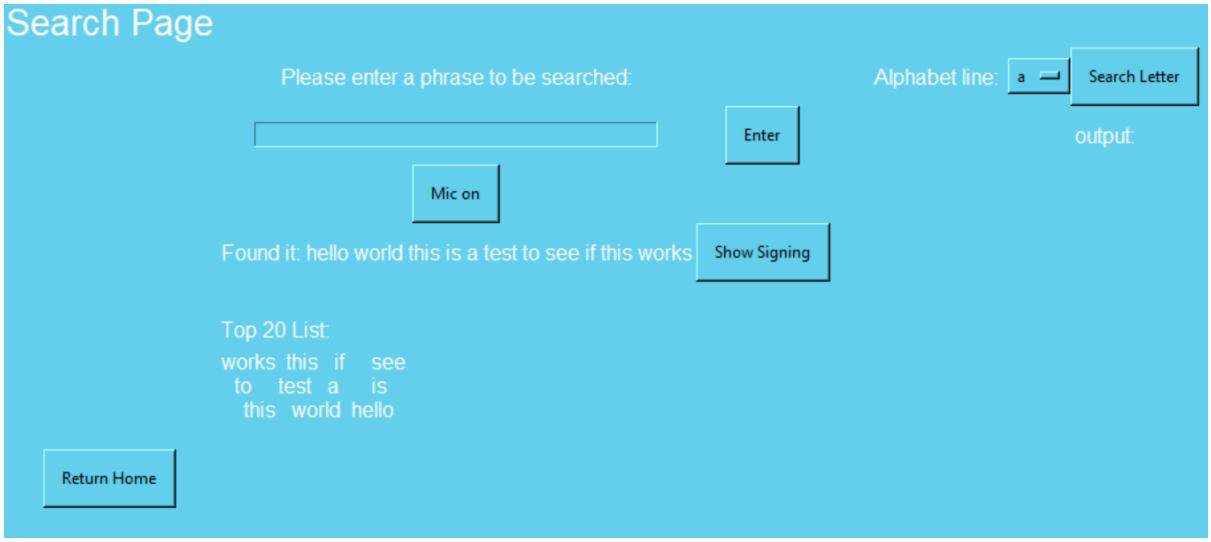
once the user has continued to sign up they are sent an email with a verification code. This code would be entered here as described by the instructions. on pressing enter the program will show the button to return to the homepage if the code is correct. This is effective as it only shows the exit button once the requirements are met.

Search page

Initial Design



Final Design



Search options:

Both designs offer a search bar for the text search and a button for the microphone input. These are both simple and effective designs.

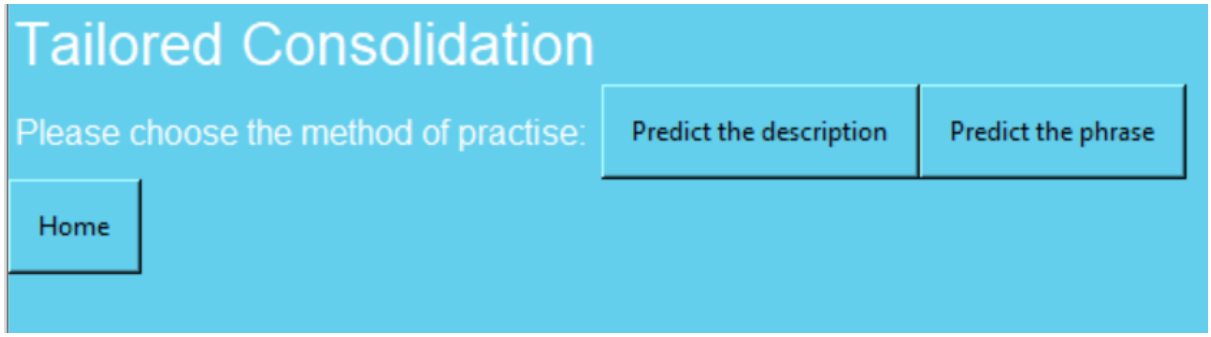
Alphabet line:

The format for the alphabet line is different for the final design as it does not incorporate a line of hyperlinked letters. Instead the alphabet is presented in a dropdown box which then presents the phrases that begin with the selected letter. While this method is effective it doesn't allow the user to click on the phrase and see it. The user must enter the desired phrase and search for it.

Top 20 list:

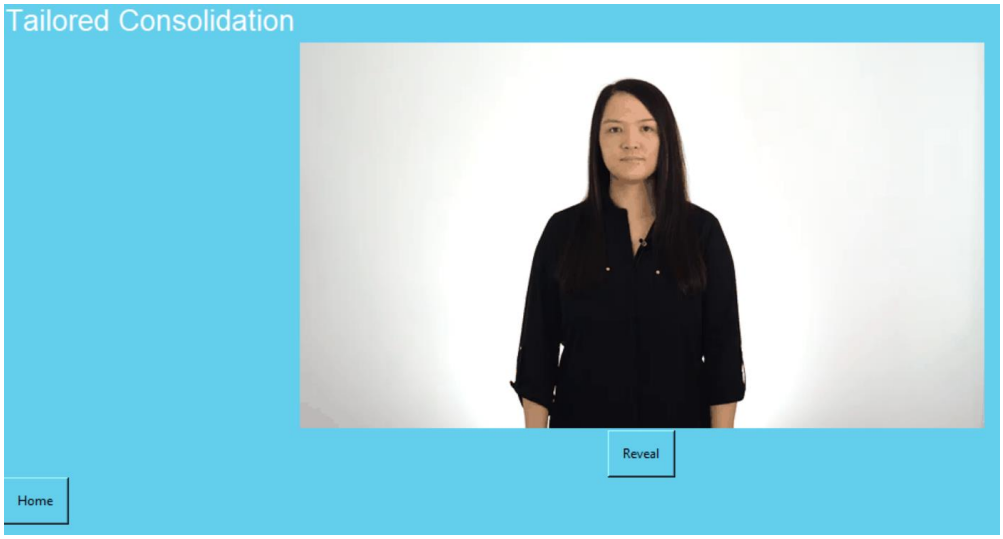
The top 20 list has a similar approach as the alphabet line where the phrases are presented identically however, they are not hyperlinked. The user must type the phrase which is a slightly less effective solution.

Tailored consolidation page



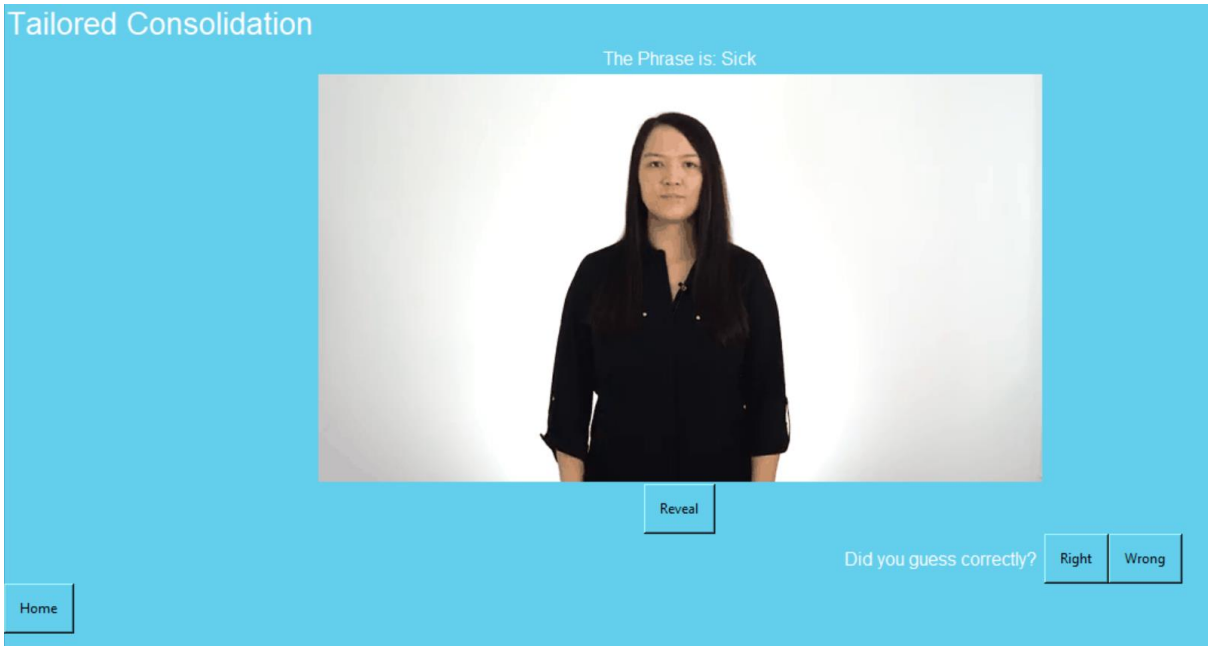
Methods of practise

The methods of practise are presented to the user by a choice of 2 buttons. Each of which would initiate the given method of learning. This is simple and effective.



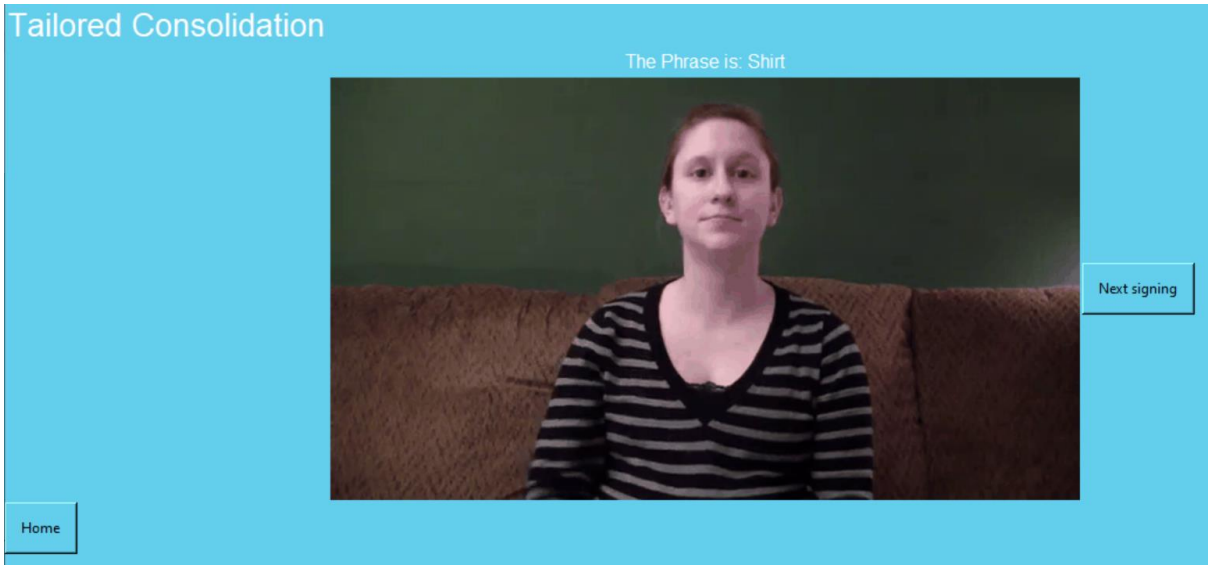
Reveal button:

A reveal button shows the description of the phrase if the “predict the description” method had been chosen. Conversely the reveal button would display the gif if the “predict the phrase” button had been clicked. Simple and effective.



Feedback buttons:

Two buttons are provided to the user to dictate whether they attempted the signing correctly or not. The lack of other options and apt descriptions makes this simple and effective.



Next signing button:

Serves as an iteration button to present the next phrase. This is the only option for the user as the other options are hidden after being pressed. This simplifies the options making it effective.

3.4.4 Maintenance and development

3.4.4a Discuss the maintainability of the solution

When designing the project, it was important to decompose each problem into its smaller chunks in order to better develop a solution, by first focusing on each individual chunk. Doing this throughout the project has created a modular design that is vital to the maintainability of the solution. Maintaining the solution includes keeping the libraries up to date, fixing errors or developing more features into the program.

Some of the methods I used to streamline these processes in the future for me or other developers are:

Having a modular approach:

As mentioned before in the project, the data manipulating parts of the program are all contained within separate functions. this modularity allowed the program to utilise any given function anywhere in the program and allowed the program to be more efficient with less bloat code. Each of these functions were kept within the same program for the sake of the project code listing, but could easily be split into separate files increasing its modularity for other users.

The project boasts 33 data manipulating functions, each of which could potentially be accessed and used by another project or person. This makes the access to the program much more available and understandable making the chances of major bug being solved much quicker.

Using external modules/data:

A total of 15 external modules are utilised in this programming project. using external modules means that tasks will be completed with greater efficiency alongside the ability to offer functionality that would otherwise take much more work and a lot more time to produce. External modules also mean that other developers looking at the code will better understand certain methods that they may already be familiar with, instead of trying to understand self-developed code that would do the same thing.

Saving data to an external file such as the online database for both BSL phrases and User profiles allows trusted developers to access this data which they would not be able to do if the data was stored locally. Having the data in a database also allows the data to be presented in a more understandable way than a long printed string.

Well commented/written code:

Having descriptive identifiers is essential to the ability of a developer to distinguish the processes within an algorithm. This, alongside descriptive comments and concise doctrings allow for a more streamlined approach to understand the processes created by a function and best way to adapt or fix it. This is vital to the maintainability of the program especially for the data manipulating functions as they will contain the manipulation of similar data sets.

```
> def add_user_to_database(user_object):···

> def updating_row():···

> def existing_user_login(username,password):···

> def create_random_practice_list():···

> def starting_consolidation():···
```

Here the functions are given descriptive identifiers to give the developer a brief understanding of what was contained within the function.

```
def add_user_to_database(user_object):
    """function abstracts all of the data from the user object and appends the
    data to the database. This data is added to the online database"""
    temp_list = []
    #values are added to an array
    temp_list.append(user_object.user_number)
    temp_list.append(user_object.name)
    temp_list.append(user_object.username)
    temp_list.append(user_object.password)
    temp_list.append(user_object.email)
    temp_list.append(user_object.date_joined)
    temp_list.append(user_object.started_TL)
    temp_list.append(str(user_object.learning_path))
    #user's consolidation data saved as a list within a string which is then extracted on retrieval
    #array added to the online user database in a row
    user_worksheet.append_row(temp_list)
```

Here you can see a docstring showing a description of what the entire function does as well as comments describing the processes within the function.

Parameterisation

Some aspects of the program are used multiple times for slightly different reasons. Parameters are passed through functions when they are called so that different pieces of data from different places can be manipulated in the same way.

```
def creating_user_object(name,username,password,email,user_number=None,
date_joined=None,started_TL=None,learning_path=None,total_right=None,total_wrong=None):
    """creates and returns a user object using values from signup page"""
    global user_object
    user_object = User(name,username,password,email,user_number,date_joined,started_TL,
    learning_path,total_right,total_wrong)#creates user object
```

This function is used in order to create a user object for the signed in user. This can be used after signing up or when logging into the program. At different points in the program the user will have different pieces of data and therefore the use of parameterisation requires the essential data to be accepted and the extra data to be passed through if it is present.

3.4.4b Discuss potential further development of the solution

When beginning the project, it was important to have an idea that was easily expandable in what could be implemented into the program. With the sign language translator there were many places where more time could be spent to develop more pieces of functionality in order to bring about a broader and more intuitive program. Some of these extra pieces of functionality were incorporated into the final program such as email confirmation and multiple forms of tailored consolidation of learning. The final program serves a solution to the problem proposed at the beginning of the project as it provides a learning platform for people to learn sign language. However, it misses out on pieces of functionality that were mentioned which could still be implemented.

Without the time and money limitations on the project more features could be added to the program in order to make it more well-rounded.

During a second phase of development more time could be spent on developing the following features:

Peer assessment functionality:

This piece of functionality would use a partner system to gamify the learning of sign language with a friend. Using an emails system and storing the username of the buddy within the user profiles themselves, the peer assessment functionality was within arm's reach for this program since a lot of the framework had already been built. Due to time limitations however, I wasn't able to begin the integration and testing process.

Sign language to text functionality:

Originally the most interesting part of the project turned out to also be the most difficult. Sign language recognition requires very specific and complex algorithms that may include machine learning. This would require me to create an entire frame work with a module that allowed camera access by the program in order to obtain the relevant data from the user. one way of doing this better is with an inferred camera which are fairly expensive in order to create more accurate algorithms. In a second phase of development it was be vital to begin by researching existing modules and libraries that may help, given the overwhelming complexity of it.

More types of sign language:

Similar to the peer assessment functionality this extension to the project was within arm's reach also. With the online databases already set up in a standardised way it would be simple to implement a new database for a separate type of sign language. The problem I ran into with this was the acquirement of sign language videos to put into the database. Without creating your own videos, sign language videos are few and far between making them extremely hard to normalise into a database. I experienced this while creating my BSL database which has roughly 300 phrases currently. During a second phase I would spend time creating more databases with more phrases in order to accommodate a wider range of expressions.

Appendix

Functional Chunk Code Listings:

Throughout the functional chunks more code may be added to the same program file. In this case the code will be highlighted to show the changes made to the entirety of the project. This clarifies the differences between each section of development within the contained code. I have made the decision that the majority of the code will be coded within the same file but the major pieces of functionality split into functions and procedures. I think this is the best and easiest way to view and understand a large portion of code by being able to see where specific functions are being called actually are and being able to follow the processing path.

Functional Chunk 1 – Full Code Listing

Return to Contents Page

```
#imports
import audiomath; audiomath.RequireAudiomathVersion( '1.12.0' )
import speech_recognition as sr
from duck_typed_microphone import DuckTypedMicrophone
from guizero import *
import json

#accessing local word bank as refrence data
#creating globals
with open("Development\Chunk1_Searching\words_dictionary.json") as f:
    database = json.load(f)
f.close()
top_20_list = []
letter_list = []

#functions

def mic_input():
    """function called when mic button is pressed.
    allows the user to use the microphone for audio input to text
    uses local function to insert result into text box"""
    microphone_input = ""
    r = sr.Recognizer()
    with DuckTypedMicrophone() as source:#uses local module as a microphone
inputter
        r.adjust_for_ambient_noise(source)
        audio = r.listen(source)#converts the input from the microphone to
audio data object
    try:
        #performs a speech recognition on an audio data instance using Google Speech Recogn
ition API
        microphone_input = ('%s' % r.recognize_google(audio))
    except:
        #excepts unknown input that cannot be translated or any other errors such as no net
work connection
        microphone_input = ("sorry, unable to translate audio")
```

```
    if 0: # plot and/or play back captured audio
        s = audiomath.Sound(audio.get_wav_data(), fs=audio.sample_rate, nChannels=1)
        s.Play()
        s.Plot()
    insert_text_box(search_engine,microphone_input)#local function inserts mic result into
text box



def search_conditioning(user_input):
    """function gets rid of any special characters or numbers but keeps spaces.returns the
conditioned string"""
    new_string = ""
    for i in range(len(user_input)):
        #loops through the string and checks if every character is a letter
        if user_input[i] == " ":#keeps the spaces
            new_string += user_input[i]
        elif user_input[i].isalpha():
            new_string += user_input[i].lower()
            #only adds the letters to the final string
    return new_string

def multi_value_input(user_input):
    """function for if more than 1 word is inputted into the text box"""
    user_input = user_input.split()#splits the string by each spaceinto a list
    output_string = ""
    for i in range(len(user_input)):#loops through each word
        for phrase in database:#loops through the database
            if user_input[i] == phrase:
                my_gui_string = top_20_adjust(top_20_list,phrase)
                #adds each of the words found from string to top 20
individually
                output_string += phrase + " "#adds to the string
                break
    if output_string == "":#if nothing was added to the string - if no words
found in database
        output_text.value = "sorry. that couldn't be found, please try again"
    else:
        output_string = output_string.strip()
        output_text.value = "Found it: " + str(output_string)
        #presents the validated output to the user
    return my_gui_string

def search_phrase():
    """function is called when enter button is pressed.
    starts by conditioning string and inserting validated string in the text
box"""
    user_input = search_conditioning(search_engine.value)
    search_engine.value = user_input#inserts validated string in text box
```

```
    searching_database(user_input,output_text)#checks the validated string
against database
def insert_text_box(text_box,text):
    """function used to replace tech in text box"""
    #function used in mic input function
    text_box.value = text

def searching_database(user_input,output_text):
    """takes user input as an attribute and searches through the
database for it"""
    if " " in user_input.strip():#if there is more than one word uses multi
function
        my_gui_string = multi_value_input(user_input)
        output_text.value = my_gui_string
    else:
        found_value = False
        for word in database:#searches through the database
            if word == user_input:
                output_text.value = "Found it: " + str(word)#outputs to the
user the phrase
                my_gui_string = top_20_adjust(top_20_list,word)#adds the
searched phrase to the top 20 list
                output_text.value = my_gui_string
                found_value = True
        if found_value == False:
            output_text.value = "sorry. that couldn't be found, please try
again"

def top_20_adjust(top_20_list,new_value):
    """function adds the new value to the front of the list.
    if the list is bigger than 20 value is removes one from the back
    and adds the new one to the front"""
    if len(top_20_list) == 20:
        top_20_list.pop(-1)#removes value from the back of the list
        top_20_list.insert(0,new_value)#adds to front
    else:
        top_20_list.insert(0,new_value)#adds to front
    output_string = top_20_format(top_20_list)
    return output_string
def top_20_format(my_list):
    """function formats the lists so that they can be more easily
displayed.creates a 2d list with up to 4 values in each of the inner
lists."""
    list_1 = []
    for i in range(len(my_list)):
        if i %4 == 0:#if the index of the list is a multiple of 4 - this
includes 0
            list_1.append([])#creates new inner list
```

```
            list_1[-1].append(my_list[i])#adds value to the
back-most inner list
        else:
            list_1[-1].append(my_list[i])#adds value to the
back-most inner list
    output_string = top_20_output(list_1)
    return output_string

def top_20_output(my_list):
    """function uses the formatted lists to display the values in
    4 colums with up to 20 values"""
    output_string = ""
    short_list = []
    iteration = 5
    if len(my_list)<iteration:#output should have 5 rows or less
        iteration = len(my_list)
    for i in range(iteration):
        short_list.append(my_list[i])#takes the first 5 "rows" of data
    col_width = max(len(word) for row in short_list for word in row)+2
    # calculates the necessary spacing between each columns for neat formation
    counter = 0
    for row in short_list:
        if counter<5:#second level of validation to make sure no more than
5 rows
            output_string += ("".join(word.ljust(col_width) for word in row) + "\n")
            #creates a string with all values in ordered fashion
        counter+=1
    return output_string

def alphabet_display(letter):#beginning letter passed through
    """function takes a letter and creates a list of all the phrases beginning
    with that letter from the database"""
    del letter_list[:]#list is cleared before initiation so values are not
added to a full list
    for phrase in database:
        if phrase[0] == letter:#if beginning letters match
            letter_list.append(phrase)#adds it to the list
    output_string = top_20_format(letter_list)#formats the list for a top 20
    alphabet_output.value = output_string
    return letter_list

def alphabet_iteration():
    """function allows the user to iterate through the values from the letter_list
    and shows the next 20 values"""
    if len(letter_list) == 0:
        #if there are no values in the letter_list
        #function will do nothing
        return
```

```
    else:
        current_letter = alphabet.value
        if len(letter_list)<=20:#if no further iteration can happen
            alphabet_display(current_letter)#redo the list
        else:
            del letter_list[:20]
            output_string = top_20_format(letter_list)#formats the list for a top 20
            alphabet_output.value = output_string#outputs to GUI
            alphabet_output.align = "left"
        return letter_list

#creating GUI
app = App(title="Searching dev",width=1400,height=700,layout="grid",
bg=(99,207,237))
instructions = Text(app,text="Please enter a phrase to be searched:",
grid=[1,1],color="white")
search_engine = TextBox(app,width=50,grid=[1,3])
enter_button = PushButton(app,command=search_phrase,text="Enter",
grid=[2,3])
mic_on = PushButton(app,command=mic_input,text="Mic on",grid=[1,4])
spacer = Text(app,text = "      ",grid=[7,1])
alphabet_text = Text(app,text="Alphabet line: ",grid=[8,1],
color="white")
alphabet = Combo(app,options=["a","b","c","d","e","f","g","h","i","j",
"k","l","m","n","o","p","q","r","s","t","u","v","w","x","y","z"],
command=alphabet_display,grid=[9,1])
alphabet_button = PushButton(app,command=alphabet_iteration,
text="Search Letter",grid=[10,1],align="left")
alphabet_output = Text(app,text="output: ",grid=[10,3],align="left")
output_text = Text(app,text="",grid=[1,5],color="white",align="left")
app.display()
```

Return to Contents Page

Functional Chunk 2 – Full Code Listing

Return to Contents Page

```
#imports
import audiomath; audiomath.RequireAudiomathVersion( '1.12.0' )
import speech_recognition as sr
from duck_typed_microphone import DuckTypedMicrophone
from fuzzywuzzy import fuzz, process #'fuzzy' matching
from guizero import *
import json

#accessing local word bank as refrence data
#creating globals
with open("Development\Chunk1_Searching\words_dictionary.json") as f:
    database = json.load(f)
f.close()
top_20_list = []
letter_list = []

#functions
def mic_input():
    """function called when mic button is pressed.
    allows the user to use the microphone for audio input to text
    uses local function to insert result into text box"""
    microphone_input = ""
    r = sr.Recognizer()
    with DuckTypedMicrophone() as source:#uses local module as a microphone
inputter
        r.adjust_for_ambient_noise(source)
        audio = r.listen(source)#converts the input from the microphone to
audio data object
    try:
        #performs a speech recognition on an audio data instance using Google Speech
Recognition API
        microphone_input = ('%s' % r.recognize_google(audio))
    except:
        #excepts unknown input that cannot be translated or any other errors
such as no network connection
        microphone_input = ("sorry, unable to translate audio")
    if 0: # plot and/or play back captured audio
        s = audiomath.Sound(audio.get_wav_data(), fs=audio.sample_rate, nChannels=1)
        s.Play()
        s.Plot()
    insert_text_box(search_engine,microphone_input)#local function inserts mic result into
text box

def search_conditioning(user_input):
    """function gets rid of any special characters or numbers but keeps spaces.
    returns the conditioned string"""
```

```
    new_string = ""
    for i in range(len(user_input)):
        #loops through the string and checks if every character is a letter
        if user_input[i] == " ":#keeps the spaces
            new_string += user_input[i]
        elif user_input[i].isalpha():
            new_string += user_input[i].lower()
            #only adds the letters to the final string
    return new_string

def fluffy_search(database,user_input,desired_percent):
    single_word_list = []#this list stores potential phrases for the user
input
    for phrase in database:#loops through the entire database
        if phrase[0] == user_input[0]:
            options_list = []#temp list to store each potential phrase
            percent_acc = fuzz.ratio(user_input,phrase)
            #uses fuzz.ratio to find how similar the input is to words in the database
            if percent_acc > desired_percent:#if the word is relatively
accurate
                options_list.append(phrase)#adds the phrase to the temp list
                options_list.append(percent_acc)#adds the level of accuracy to the temp
list
                for i in range(len(single_word_list)):#loops through the
entire selection of possibilities
                    phrase_percent = int(single_word_list[i][1])
                    if percent_acc>=phrase_percent:
                        single_word_list.insert(i,options_list)
                        #inserts the temp list with the newest phrase so that the final
list is sorted
                        break#breaks out of the loop
                    elif (i+1) == len(single_word_list):
                        single_word_list.insert(i+1,options_list)
                        #if the phrase is the least accurate it is inserted to the back of
the list
                        break#breaks out of the loop
                if len(single_word_list) == 0:#if there are no values in the list
                    single_word_list.append(options_list)
    return single_word_list

def search_phrase():
    """function is called when enter button is pressed.
    starts by conditioning string and inserting validated string in the text box"""
    user_input = search_conditioning(search_engine.value)
    search_engine.value = user_input#inserts validated string in text box
    searching_database(user_input,output_text)#checks the validated string
against database
```

```
def insert_text_box(text_box,text):
    """function used to replace tech in text box"""
    #function used in mic input function
    text_box.value = text

def searching_database(user_input,output_text):
    """takes user input as an attribute and searches through the database for it.
    this includes a fluffy search which outputs the most accurate word it can find in the d
atabase.
    this function also updates the top 20 list"""
    user_input_list = user_input.split()
    output_string = ""
    temp_list = []
    total_list = []
    for i in range(len(user_input_list)):#iterates through each inputted word
        single_word_list = fluffy_search(database,user_input_list[i],80)
#creates a list of possible phrases
        total_list.append(single_word_list)#appends all possible phrases for
each inputted word to total list
        if len(single_word_list) != 0:#if there were no found possibilities
            my_gui_string = top_20_adjust(top_20_list,total_list[i][0][0])
            #adds the searched phrase to the top 20 list and returns the new
top 20 list to output
            output_string += total_list[i][0][0] + " " #adds new value to
output to be printed

    if output_string == "":#if nothing was added to the string - if no words found in
database
        output_text.value = "sorry. that couldn't be found, please try again"
    else:
        output_string = output_string.strip()#gets rid of any possible spaces on the ends
        correction = False#used to check if input needed to be corrected
        for word_list in total_list:
            if word_list[0][1] != 100:#if the accuracy value was not 100
                correction = True#phrase was corrected
        #different output message shown telling user if the phrase was corrected or not
        if correction == True:
            output_text.value = "Showing most similar phrases: " + str(output_string)
        else:
            output_text.value = "Found it: " + str(output_string)
        #presents the validated output to the user
    top_20_output_text.value = my_gui_string

def top_20_adjust(top_20_list,new_value):
    """function adds the new value to the front of the list.
    if the list is bigger than 20 value is removes one from the back
    and adds the new one to the front"""
    if len(top_20_list) == 20:
```

```
        top_20_list.pop(-1)#removes value from the back of the list
        top_20_list.insert(0,new_value)#adds to front
    else:
        top_20_list.insert(0,new_value)#adds to front
    output_string = top_20_format(top_20_list)
    return output_string

def top_20_format(my_list):
    """function formats the lists so that they can be more easily displayed.
    creates a 2d list with up to 4 values in each of the inner lists."""
    list_1 = []
    for i in range(len(my_list)):
        if i %4 == 0:#if the index of the list is a multiple of 4 - this includes 0
            list_1.append([])#creates new inner list
            list_1[-1].append(my_list[i])#adds value to the back-most inner list
        else:
            list_1[-1].append(my_list[i])#adds value to the back-most inner list
    output_string = top_20_output(list_1)
    return output_string

def top_20_output(my_list):
    """function uses the formatted lists to display the values in
    4 colums with up to 20 values"""
    output_string = ""
    short_list = []
    iteration = 5
    if len(my_list)<iteration:#output should have 5 rows or less
        iteration = len(my_list)
    for i in range(iteration):
        short_list.append(my_list[i])#takes the first 5 "rows" of data
    col_width = max(len(word) for row in short_list for word in row)+2
    # calculates the necessary spacing between each columns for neat
formations
    counter = 0
    for row in short_list:
        if counter<5:#second level of validation to make sure no more than 5 rows
            output_string += ("".join(word.ljust(col_width) for word in row) + "\n")
            #creates a string with all values in ordered fashion
        counter+=1
    return output_string

def alphabet_display(letter):#beginning letter passed through
    """function takes a letter and creates a list of all the phrases beginning
    with that letter from the database"""
    del letter_list[:]#list is cleared before initiation so values are not
added to a full list
    for phrase in database:
        if phrase[0] == letter:#if beginning letters match
```

```
            letter_list.append(phrase)#adds it to the list
    output_string = top_20_format(letter_list)#formats the list for a top 20
    alphabet_output.value = output_string
    return letter_list

def alphabet_iteration():
    """function allows the user to iterate through the values from the letter_list
    and shows the next 20 values"""
    if len(letter_list) == 0:
        #if there are no values in the letter_list
        #function will do nothing
        return
    else:
        current_letter = alphabet.value
        if len(letter_list)<=20:#if no further iteration can happen
            alphabet_display(current_letter)#redo the list
        else:
            del letter_list[:20]
            output_string = top_20_format(letter_list)#formats the list for a top 20
            alphabet_output.value = output_string#outputs to GUI
            alphabet_output.align = "left"
        return letter_list
#creating GUI
app = App(title="Searching dev",width=1400,height=700,layout="grid",
bg=(99,207,237))
instructions = Text(app,text="Please enter a phrase to be searched:",
grid=[1,1],color="white")
search_engine = TextBox(app,width=50,grid=[1,3])
enter_button = PushButton(app,command=search_phrase,text="Enter",grid=[2,3])
mic_on = PushButton(app,command=mic_input,text="Mic on",grid=[1,4])
spacer = Text(app,text = "      ",grid=[7,1])
alphabet_text = Text(app,text="Alphabet line: ",grid=[8,1],color="white")
alphabet = Combo(app,options=["a","b","c","d","e","f","g","h","i","j","k","l",
"m","n","o","p","q","r","s","t","u","v","w","x","y","z"],
command=alphabet_display,grid=[9,1])
alphabet_button = PushButton(app,command=alphabet_iteration,text="Search
Letter",grid=[10,1],align="left")
alphabet_output = Text(app,text="output: ",grid=[10,3],align="left")
output_text = Text(app,text="",grid=[1,5],color="white",align="left")
spacer2 = Text(app,text="",grid=[1,6],color="white",align="left")
top_20_title = Text(app,text="Top 20 List: ",grid=[1,7],color="white",
align="left")
top_20_output_text = Text(app,text="",grid=[1,8],color="white",align="left")
app.display()
```

Return to Contents Page

Functional Chunk 3 – Full Code Listing

```
#imports
import audiomath; audiomath.RequireAudiomathVersion( '1.12.0' )
import speech_recognition as sr
from duck_typed_microphone import DuckTypedMicrophone
from fuzzywuzzy import fuzz, process #'fuzzy' matching
from guizero import *
from json import load
from gspread import service_account
from random import choice,randint,shuffle
from pytube import *
from moviepy.editor import *
import os
import smtplib
from datetime import date
import hashlib#importing module
import ast

#accessing local word bank as refrence data
#creating globals
with open("words_dictionary.json") as f:
    database = load(f)
f.close()

#accessing online bsl database
gc = service_account(filename="credentials.json")
sh = gc.open_by_key("1iSf4VziTbvs0nuzu7X-kcxqPyiuBg-iB4Usykh0mS7I")
bsl_worksheet = sh.sheet1#selects the first sheet of the documment
total_vids = bsl_worksheet.get_all_records()

#accessing online user database
sh2 = gc.open_by_key("1eYarci7vaZndWdLMJJlJj_KkWKNece3s-kIm-o2pUgA")
#this is the key for the google docs. this can be found in the URL after the /
d/
user_worksheet = sh2.sheet1#selects the first sheet of the documment
total_users = user_worksheet.get_all_records()

#obtaining email and password from local environment variables
EMAIL_ADDRESS = os.environ.get("EMAIL_USER")
EMAIL_PASSWORD = os.environ.get("EMAIL_PASS")

#globals
top_20_list = []
letter_list = []
consolidation_list = []
tailored_list = []
template_list = []
search_output_file_names = []
```

```
consolidation_list = []
alphabet_list = ["a","b","c","d","e","f","g","h","i","j","k","l",
"m","n","o","p","q","r","s","t","u","v","w","x","y","z"]
global signed_in
signed_in = False
predict_desc = None
sign_up_code = 0
user_object = None

#classes
class User:
    """class that creates user objects"""
    user_counter = 0
    def __init__(self,name,username,password,email,user_number,date_joined,sta
rted_TL,learning_path,total_right,total_wrong):
        """initialising the class"""
        today = date.today()
        self.name = name
        self.username = username
        self.password = password
        self.email = email
        if total_right == None:
            self.total_right = 0
        else:
            self.total_right = total_right
        if total_wrong == None:
            self.total_wrong = 0
        else:
            self.total_wrong = total_wrong
        if learning_path != None:
            try:
                self.learning_path = ast.literal_eval(learning_path)
            except:
                self.learning_path = learning_path
        else:
            self.learning_path = None
        if started_TL == None:
            self.started_TL = "No"
        else:
            self.started_TL = started_TL

        if date_joined == None:
            self.date_joined = today.strftime("%d/%m/%Y")
        else:
            self.date_joined = date_joined
        if user_number == None:
            self.user_number = User.user_counter + 1
            User.user_counter += 1
```

```
        else:
            self.user_number = user_number
    def __str__(self):
        """print method shows all the values of the object"""
        print_str = ""
        print_str+= "user number: " + str(self.user_number) + "\n"
        print_str+= "name: " + str(self.name)+ "\n"
        print_str+= "username: " + str(self.username)+ "\n"
        print_str+= "password: " + str(self.password)+ "\n"
        print_str+= "date joined: " + str(self.date_joined)+ "\n"
        print_str+= "email: " + str(self.email) + "\n"
        print_str+= "started tailored learning? " + str(self.started_TL) + "\n
"
        print_str+= "current learning path: "+ str(self.learning_path)+ "\n"
        print_str+= "total right: " + str(self.total_right) + "\n"
        print_str+= "total wrong: " + str(self.total_wrong)
        return print_str

#functions
def set_user_counter():
    """method that sets the user counter as the most recent user number.
    This function should be called each time the program is run so that
    the user number can be set to the correct number."""
    User.user_counter = total_users[-1]["User number"]

def mic_input():
    """function called when mic button is pressed.
    allows the user to use the microphone for audio input to text
    uses local function to insert result into text box"""
    microphone_input = ""
    r = sr.Recognizer()
    with DuckTypedMicrophone() as source:#uses local module as a microphone in
putter
        r.adjust_for_ambient_noise(source)
        audio = r.listen(source)#converts the input from the microphone to aud
io data object
    try:
        #performs a speech recognition on an audio data instance using Google
Speech Recognition API
        microphone_input = ('%s' % r.recognize_google(audio))
    except:
        #excepts unknown input that cannot be translated or any other errors s
uch as no network connection
        microphone_input = ("sorry, unable to translate audio")
    if 0: # plot and/or play back captured audio
        s = audiomath.Sound(audio.get_wav_data(), fs=audio.sample_rate, nChann
els=1)
        s.Play()
```

```
        s.Plot()
    search_engine.value = microphone_input

def search_conditioning(user_input):
    """function gets rid of any special characters or numbers but keeps spaces
.
    returns the conditioned string"""
    new_string = ""
    for i in range(len(user_input)):
        #loops through the string and checks if every character is a letter
        if user_input[i] == " ":#keeps the spaces
            new_string += user_input[i]
        elif user_input[i].isalpha():
            new_string += user_input[i].lower()
            #only adds the letters to the final string
    return new_string

def fluffy_search(database,user_input,desired_percent):
    single_word_list = []#this list stores potential phrases for the user inpu
t
    for phrase in database:#loops through the entire database
        if phrase[0] == user_input[0]:
            options_list = []#temp list to store each potential phrase
            percent_acc = fuzz.ratio(user_input,phrase)
            #uses fuzz.ratio to find how similar the input is to words in the
database
            if percent_acc > desired_percent:#if the word is relatively accura
te
                options_list.append(phrase)#adds the phrase to the temp list
                options_list.append(percent_acc)#adds the level of accuracy to
 the temp list
                for i in range(len(single_word_list)):#loops through the entir
e selection of possibilities
                    phrase_percent = int(single_word_list[i][1])
                    if percent_acc>=phrase_percent:
                        single_word_list.insert(i,options_list)
                        #inserts the temp list with the newest phrase so that
the final list is sorted
                        break#breaks out of the loop
                    elif (i+1) == len(single_word_list):
                        single_word_list.insert(i+1,options_list)
                        #if the phrase is the least accurate it is inserted to
 the back of the list
                        break#breaks out of the loop
                if len(single_word_list) == 0:#if there are no values in the l
ist
                    single_word_list.append(options_list)
    return single_word_list
```

```
def searching_database(user_input,output_text):
    """takes user input as an attribute and searches through the database for
it.
    this includes a fluffy search which outputs the most accurate word it can
find in the database.
    this function also updates the top 20 list"""
    user_input_list = user_input.split()
    output_string = ""
    temp_list = []
    total_list = []
    my_gui_string = ""
    for i in range(len(user_input_list)):#iterates through each inputted word

        single_word_list = fluffy_search(database,user_input_list[i],80)#creat
es a list of possible phrases
        total_list.append(single_word_list)#appends all possible phrases for e
ach inputted word to total list
        if len(single_word_list) != 0:#if there were no found possibilities
            my_gui_string = top_20_adjust(top_20_list,total_list[i][0][0])
            #adds the searched phrase to the top 20 list and returns the new t
op 20 list to output
            output_string += total_list[i][0][0] + " " #adds new value to outp
ut to be printed

    if output_string == "":#if nothing was added to the string - if no words f
ound in database
        output_text.value = "Sorry. That couldn't be found, please try again"
        signing_page_button.visible = False
    else:
        output_string = output_string.strip()#gets rid of any possible spaces
on the ends
        correction = False#used to check if input needed to be corrected
        for word_list in total_list:
            if word_list[0][1] != 100:#if the accuracy value was not 100
                correction = True#phrase was corrected
        #different output message shown telling user if the phrase was correct
ed or not
        if correction == True:
            output_text.value = "Showing most similar phrases: " + str(output_
string)
        else:
            output_text.value = "Found it: " + str(output_string)
        signing_page_button.visible = True
        #presents the validated output to the user
    top_20_output_text.value = my_gui_string

def search_phrase():
```

```
    """function is called when enter button is pressed.
    starts by conditioning string and inserting validated string in the text b
ox"""
    user_input = search_conditioning(search_engine.value)
    search_engine.value = user_input#inserts validated string in text box
    searching_database(user_input,output_text)#checks the validated string aga
inst database

def top_20_adjust(top_20_list,new_value):
    """function adds the new value to the front of the list.
    if the list is bigger than 20 value is removes one from the back
    and adds the new one to the front"""
    if len(top_20_list) == 20:
        top_20_list.pop(-1)#removes value from the back of the list
        top_20_list.insert(0,new_value)#adds to front
    else:
        top_20_list.insert(0,new_value)#adds to front
    output_string = top_20_format(top_20_list)
    return output_string

def top_20_format(my_list):
    """function formats the lists so that they can be more easily displayed.
    creates a 2d list with up to 4 values in each of the inner lists."""
    list_1 = []
    for i in range(len(my_list)):
        if i %4 == 0:#if the index of the list is a multiple of 4 - this inclu
des 0
            list_1.append([])#creates new inner list
            list_1[-1].append(my_list[i])#adds value to the back-
most inner list
        else:
            list_1[-1].append(my_list[i])#adds value to the back-
most inner list
    output_string = top_20_output(list_1)
    return output_string

def top_20_output(my_list):
    """function uses the formatted lists to display the values in
    4 colums with up to 20 values"""
    output_string = ""
    short_list = []
    iteration = 5
    if len(my_list)<iteration:#output should have 5 rows or less
        iteration = len(my_list)
    for i in range(iteration):
        short_list.append(my_list[i])#takes the first 5 "rows" of data
    col_width = max(len(word) for row in short_list for word in row)+2
```

```
    # calculates the necessary spacing between each columns for neat formation
s
    counter = 0
    for row in short_list:
        if counter<5:#second level of validation to make sure no more than 5 r
ows
            output_string += ("".join(word.ljust(col_width) for word in row) +
 "\n")
            #creates a string with all values in ordered fashion
        counter+=1
    return output_string

def alphabet_display(letter):#beginning letter passed through
    """function takes a letter and creates a list of all the phrases beginning

    with that letter from the database"""
    del letter_list[:]#list is cleared before initiation so values are not add
ed to a full list
    counter = 0
    for word in total_vids:
        if word["Phrase"][0].lower() == letter:#if beginning letters match
            letter_list.append(word["Phrase"])#adds it to the list
    output_string = top_20_format(letter_list)#formats the list for a top 20
    alphabet_output.value = output_string
    return letter_list

def alphabet_iteration():
    """function allows the user to iterate through the values from the letter_
list
    and shows the next 20 values"""
    if len(letter_list) == 0:
        #if there are no values in the letter_list
        #function will do nothing
        return
    else:
        current_letter = alphabet.value
        if len(letter_list)<=20:#if no further iteration can happen
            alphabet_display(current_letter)#redo the list
        else:
            del letter_list[:20]
            output_string = top_20_format(letter_list)#formats the list for a
top 20
            alphabet_output.value = output_string#outputs to GUI
        return letter_list

def create_url_list(user_string):
    found_values = []#list of phrases that are in input
    user_input_list = user_string.split()
```

```
    for i in range(len(user_input_list)):
        #creates 2d list of input [user word,index of given phrase in found_va
lues]
        user_input_list[i] = [user_input_list[i]]
        user_input_list[i].append(None)

    for value in total_vids:#loops through all phrases
        db_phrase = value["Phrase"].lower()#uses lowercase phrases
        if db_phrase in user_string:#if phrase present in input
            found_values.append(value["Phrase"])#appends formatted phrase to l
ist
            for i in range(len(user_input_list)):#loops through each word of u
ser input
                if db_phrase.split()[0] == user_input_list[i][0]:#lines up phr
ase and user input
                    for j in range(len(db_phrase.split())):#loops through leng
th of phrase
                        if user_input_list[i+j][1] == None:#if there isnt anyt
hing found for a phrase
                            user_input_list[i+j][1] = len(found_values)-1
                            #current phrase set as value via pointer
                        elif len(db_phrase.split()) > len(found_values[user_in
put_list[i+j][1]]):
                            #if current phrase is bigger than previous phrase
                            user_input_list[i+j][1] = len(found_values)-1
                            #current phrase set at value via pointer

    for i in range(len(user_input_list)):#iterating through user input
        if user_input_list[i][1] == None:#find word that has no signings
            temp_list = []
            for j in range(len(user_input_list[i][0])):#loops through length o
f word
                for k in range(len(found_values)):#loops through the found val
ues
                    if user_input_list[i][0][j] == found_values[k].lower():
                        #if current letter is the letter in found values
                        temp_list.append(k)#appends index of letter to temp li
st
                        break
            user_input_list[i][1] = temp_list
            #temp list is used in place where other words have a single phrase
    return user_input_list,found_values

def gif_setup():
    """this function finds the signings of the specific phrases in online data
base"""
    #obtains corrected user input
    corrected_input = ""
```

```
    for i in range(len(output_text.value)):
        if output_text.value[i] == ":":
            corrected_input = output_text.value[i+2:]
            break
    #sets intro text value
    search_result_user_input.value = "Searching for: " + corrected_input
    clear_search_page()
    del search_output_file_names[:]#clears the global list
    user_input_list,found_values = create_url_list(corrected_input.lower())
    #obtains the total phrases found in the input and a 2d list containing
    #each word in the input and a pointer to a value in the found_values list
    temp_list = []
    counter = 0
    while counter < len(user_input_list):
        #this loops round the 2d list and abstracts all of the phrase descript
ions
        #by looking at each of the indexes in found_values list
        if type(user_input_list[counter][1]) is list:
            for j in range(len(user_input_list[counter][1])):
                search_output_file_names.append(found_values[user_input_list[c
ounter][1][j]])
            counter+=1
        else:
            search_output_file_names.append(found_values[user_input_list[count
er][1]])
            counter+=len(found_values[user_input_list[counter][1]].split())
    iterate_search_result()

def iterate_search_result():
    """Outputs each signing and iterates to the next signing phrase in the lis
t when the button pressed.
    Downloads the next signing video phrase and converts it into .gif.
    The iteration button only appears if there is more than one phrase.
    Also updates the text at the top indicating what is currently being displa
yed."""
    search_result_gif_output.value = None
    #shows the iterate button if there are more than one phrases
    if len(search_output_file_names)>1:
        search_result_next_button.show()
    else:
        search_result_next_button.hide()
    try:
        #presents description of the current phrase being shown
        search_resut_current_output.value = "Currently showing: "+str(search_o
utput_file_names[0])
        if not os.path.exists(search_output_file_names[0]+".gif"):
            #if video needs to be downloaded
            for value in total_vids:#obtains correct URL
```

```
                if search_output_file_names[0] == value["Phrase"]:
                    output_url = value["URL"]
                    break
            ytd = YouTube(output_url).streams.first().download(filename=search
_output_file_names[0])
            print("Video downloaded to local directory")
            file_name = search_output_file_names[0] + ".mp4"
            clip = (VideoFileClip(file_name))#creates clip instance
            file_name = file_name.replace(".mp4",".gif")#changes the file type
 on string
            clip.write_gif(file_name)#converts .mp4 video to .gif

            search_result_gif_output.image = file_name#outputs first value in
list
            del search_output_file_names[0]#gets rid of the first value in lis
t
        else:

            file_name = search_output_file_names[0]+".gif"
            search_result_gif_output.image = file_name#outputs first value in
list
            del search_output_file_names[0]#gets rid of the first value in lis
t
    except:
        pass

def creating_user_object(name,username,password,email,user_number=None,
date_joined=None,started_TL=None,learning_path=None,total_right=None,total_wro
ng=None):
    """creates and returns a user object using values from signup page"""
    global user_object
    user_object = User(name,username,password,email,user_number,date_joined,st
arted_TL,
    learning_path,total_right,total_wrong)#creates user object

def add_user_to_database(user_object):
    """function abstracts all of the data from the user object and appends the
    data to the database. This data is added to the online database"""
    temp_list = []
    #values are added to an array
    temp_list.append(user_object.user_number)
    temp_list.append(user_object.name)
    temp_list.append(user_object.username)
    temp_list.append(user_object.password)
    temp_list.append(user_object.email)
    temp_list.append(user_object.date_joined)
    temp_list.append(user_object.started_TL)
    temp_list.append(str(user_object.learning_path))
```

```
    #user's consolidation data saved as a list within a string which is then e
xtracted on retrieval
    #array added to the online user database in a row
    user_worksheet.append_row(temp_list)

def updating_row():
    global user_object
    temp_list = []
    #values are added to an array
    temp_list.append(user_object.user_number)
    temp_list.append(user_object.name)
    temp_list.append(user_object.username)
    temp_list.append(user_object.password)
    temp_list.append(user_object.email)
    temp_list.append(user_object.date_joined)
    temp_list.append(user_object.started_TL)
    temp_list.append(str(user_object.learning_path))
    temp_list.append(user_object.total_right)
    temp_list.append(user_object.total_wrong)
    #user's consolidation data saved as a list within a string which is then e
xtracted on retrieval
    #array added to the online user database in a row
    row_number = user_object.user_number + 1
    user_worksheet.delete_row(row_number)
    user_worksheet.insert_row(temp_list,row_number)

def existing_user_login(username,password):
    """Creates a user object from user database.
    Correct details are assured as username is unique(primary key)"""
    global total_users
    total_users = user_worksheet.get_all_records()
    #refreshes the local database from the online database
    current_user = False
    for profile in total_users:
        if profile["Username"] == username:#looks for user profile
            init_password = str.encode(password)
            #encodes the password in the same way so that they match if inputs
 match
            hashed_password = hashlib.sha224(init_password).hexdigest().strip(
"0")
            if profile["Password"] == hashed_password:#if password is correct
                #obtains all the user information
                user_number = profile["User number"]
                name = profile["Name"]
                username = profile["Username"]
                password = profile["Password"]
                email = profile["Email"]
                date_joined = profile["Date joined"]
```

```
                started_TL = profile["Started tailored learning"]
                learning_path = profile["Learning path"]
                total_right = profile["Total right"]
                total_wrong = profile["Total wrong"]
                creating_user_object(name,username,password,email,
                user_number,date_joined,started_TL,learning_path,
                total_right,total_wrong)
                #creates a user object
                current_user = True
    return current_user

def create_random_practice_list():
    """creating a list of 10 random phrases with URL"""
    my_list = []
    for i in range(10):
        random_phrase = choice(total_vids)
        if random_phrase not in my_list:
            my_list.append(random_phrase)
    for value in total_vids:
        if value["Phrase"] == "Telephone":
            my_list.append(value)
    return my_list

def starting_consolidation():
    begin_text.value = "Beginning Random Consolidation"
    begin_practise_button.hide()
    consolidation_finish_text.hide()
    global consolidation_list# uses global list so that it can be accessed dur
ing iteration
    consolidation_list = create_random_practice_list()#creates a random list o
f 20 phrases
    iterate_consolidation_output()#first iteration initiallising all of the fu
nctionality

def iterate_consolidation_output():
    consolidation_gif_output.image = None
    consolidation_phrase_text.hide()
    consolidation_reveal_desc.hide()
    if len(consolidation_list)>1:
        consolidation_iterate_button.show()
        begin_practise_button.show()
    else:
        consolidation_iterate_button.hide()
        consolidation_finish_text.show()
    try:
        if not os.path.exists(consolidation_list[0]["Phrase"]+".gif"):
            #if video needs to be downloaded
            output_url = consolidation_list[0]["URL"]
```

```
            ytd = YouTube(output_url).streams.first().download(filename=consol
idation_list[0]["Phrase"])
            print("Video downloaded to local directory")
            file_name = consolidation_list[0]["Phrase"] + ".mp4"
            clip = (VideoFileClip(file_name))#creates clip instance
            file_name = file_name.replace(".mp4",".gif")#changes the file type
 on string
            clip.write_gif(file_name)#converts .mp4 video to .gif
            consolidation_gif_output.image = file_name#outputs first value in
list
            consolidation_gif_output.show()
            consolidation_reveal_desc.show()
            consolidation_phrase_text.value = "The Phrase is: " + consolidatio
n_list[0]["Phrase"]
            del consolidation_list[0]#gets rid of the first value in list
        else:
            file_name = consolidation_list[0]["Phrase"]+".gif"
            consolidation_gif_output.image = file_name#outputs first value in
list
            consolidation_gif_output.show()
            consolidation_reveal_desc.show()
            consolidation_phrase_text.value = "The Phrase is: " + consolidatio
n_list[0]["Phrase"]
            del consolidation_list[0]#gets rid of the first value in list

    except:
        pass

def show_consolidation_phrase():
    consolidation_phrase_text.show()

def send_confirmation_email():
    """function sends an email to the user containing a confirmation code"""
    sign_up_code = randint(100000,999999)#generates a random 6-
digit code as the validation code
    #obtains the user's username to be used as refrence in the email
    username = signup_username_input.value
    email = signup_email_input.value
    password = signup_password_input.value
    name = signup_name_input.value
    passed = False
    with smtplib.SMTP("smtp.outlook.com",587) as smtp:
        #initialising the connection
        smtp.ehlo()
        smtp.starttls()
        smtp.ehlo()
        smtp.login(EMAIL_ADDRESS,EMAIL_PASSWORD)
        #creates the message in the email
```

```
        subject = "Sign Language Translator Sign Up"
        body = "This is a confirmation email for an account made for Sign Lang
uage Translator.\n"
        body+="Username: " + str(username) + "\n\n"
        body += "Please enter the following code to verify your account: " + s
tr(sign_up_code)
        msg = f"Subject: {subject}\n\n{body}"#formats the message
        try:
            if len(name)>0 and len(password)>0 and len(username)> 0:
                smtp.sendmail(EMAIL_ADDRESS,email,msg)#sends message from admi
n email to user
                passed = True
        except:
            pass
    return sign_up_code,passed

def confirming_sign_up_code():
    """procedure checks whether the user has entered the correct confirmation
code"""
    try:
        user_input = int(confirmation_code_box.value)
        sign_up_code = int(confirmation_holder.value)
        if user_input == sign_up_code:
            confirmation_output_text.value = "Code confirmed."
            confirmation_continue_button.show()
            confirmation_enter_button.hide()
        else:
            #if the code wasn't correct
            confirmation_output_text.value = "Incorrect code. Please recheck y
our email"
    except:
        #if the input contains something that isnt a number
        confirmation_output_text.value = "Please enter 6-digit code"

def sign_out():
    global user_object
    global signed_in
    updating_row()
    signed_in = False
    del user_object
    user_object = None
    home_loggedin_page.hide()
    home_page.show()

def tailored_predict_desc():
    """This function is used when the user wants to see the actions/.gif
    and wants to guess the phrase/description.
    This functtion initialises this specific tailored learning"""
```

```
    tailored_phrase_then_desc.hide()
    tailored_desc_then_phrase.hide()
    tailored_consolidation_options.hide()
    global predict_desc
    predict_desc = True
    #this dictates to other functions which type of tailred learning is being
done
    global tailored_list
    del tailored_list[:]#this is the list for practised phrases (queue)
    global template_list
    del template_list[:]#this is a template used to set the queue of phrases
    template_list = create_template_list()
    for value in template_list:
        #the queue of phrases with tailored repeats is made from the template
        #phrases that the user is less confident in have more repeats
        for i in range(5-int(value["Confidence"])):
            tailored_list.append(value)
    shuffle(tailored_list)#queue is shuffled
    tailored_consolidation()#starting the consolidation

def tailored_predict_phrase():
    """This function is used when the user wants to see the phrase/
    description and wants to guess the actions/.gif"""
    tailored_phrase_then_desc.hide()
    tailored_desc_then_phrase.hide()
    tailored_consolidation_options.hide()
    global predict_desc
    predict_desc = False
    #this dictates to other functions which type of tailred learning is being
done
    global tailored_list
    del tailored_list[:]#this is the list for practised phrases (queue)
    global template_list
    del template_list[:]#this is a template used to set the queue of phrases
    template_list = create_template_list()
    for value in template_list:
        #the queue of phrases with tailored repeats is made from the template
        #phrases that the user is less confident in have more repeats
        for i in range(5-int(value["Confidence"])):
            tailored_list.append(value)
    shuffle(tailored_list)#queue is shuffled
    tailored_consolidation()#starting the consolidation

def create_template_list():
    global user_object
    #checks whether the user has done tailored learning
    #if yes the user already has a list of phrases to use
    if user_object.started_TL == "Yes":
```

```
        temp_list = []
        for practised_phrase in user_object.learning_path:
            #learning path is the attribute of the obeject
            # that contains the user history
            if practised_phrase["Confidence"] != 5:
                #the number 5 dictates the number of points needed
                #to achieve "mastery"
                temp_list.append(practised_phrase)
                #only non mastered phrases are added
                #to be learnt in continuation
    else:
        #user needs a new set of phrases to learn
        temp_list = create_random_practice_list()
        for value in temp_list:
            value.update({"Confidence":0})
        user_object.started_TL = "Yes"
        user_object.learning_path = []
        for i in range(len(temp_list)):
            user_object.learning_path.append(temp_list[i])
    return temp_list

def tailored_consolidation():
    #presets all of the GUI objects
    tailored_gif_output.image = None
    tailored_gif_output.hide()
    tailored_phrase_text.hide()
    tailored_reveal.hide()
    tailored_iteration_button.hide()
    tailored_feedback_text.hide()
    tailored_correct_button.hide()
    tailored_incorrect_button.hide()
    global predict_desc
    try:
        if not os.path.exists(tailored_list[0]["Phrase"]+".gif"):
            #if the video is not already installed in the local directory
            output_url = tailored_list[0]["URL"]
            ytd = YouTube(output_url).streams.first().download(filename=tailor
ed_list[0]["Phrase"])
            print("Video downloaded to local directory")
            file_name = tailored_list[0]["Phrase"] + ".mp4"
            clip = (VideoFileClip(file_name))#creates clip instance
            file_name = file_name.replace(".mp4",".gif")#changes the file type
 on string
            clip.write_gif(file_name)#converts .mp4 video to .gif

            tailored_gif_output.image = file_name#outputs first value in list
            tailored_reveal.show()#shows the button to reveal the phrase/gif d
epending
```

```
            tailored_phrase_text.value = "The Phrase is: " + tailored_list[0][
"Phrase"]
            if predict_desc:
                tailored_gif_output.show()#shows the gif
            else:
                tailored_phrase_text.show()#shows the description
        else:
            file_name = tailored_list[0]["Phrase"]+".gif"
            tailored_gif_output.image = file_name#outputs first value in list
            tailored_reveal.show()#shows the button to reveal the phrase/gif d
epending
            tailored_phrase_text.value = "The Phrase is: " + tailored_list[0][
"Phrase"]
            if predict_desc:
                tailored_gif_output.show() #shows the .gif
            else:
                tailored_phrase_text.show()#shows the description
    except:
        pass

def show_tailored_phrase():
    global predict_desc
    if predict_desc:
        tailored_phrase_text.show()
    else:
        tailored_gif_output.show()
    tailored_feedback_text.show()
    tailored_correct_button.show()
    tailored_incorrect_button.show()

def correct_tailored_consolidation():
    global template_list
    global tailored_list
    global user_object
    user_object.total_right += 1
    for i in range(len(template_list)):
        if template_list[i]["Phrase"] == tailored_list[0]["Phrase"]:
            template_list[i]["Confidence"] += 1
            #increases confidence in template list for specific phrase that us
er got right
            if int(template_list[i]["Confidence"]) == 5:#if user has "mastered
" the phrase
                removed = False
                while not removed:
                    #loops through and removes all other of that phrase from t
ailored_list (queue)
                    edited = False
                    for j in range(len(tailored_list)):
```

```
                        if tailored_list[j]["Phrase"] == template_list[i]["Phr
ase"]:
                            edited=True
                            del tailored_list[j]
                            break
                    if not edited:
                        removed = True
                temp_var = template_list[i]
                user_object.learning_path.append(temp_var)
                #appends mastered value to the user object to be stored
                del template_list[i]
                #deletes the value from the template list

                for k in range(len(total_vids)):#loops through the the video d
atabase
                    random_phrase = total_vids[k]
                    for complete_phrase in user_object.learning_path:
                        #loops through the users consolidation history
                        if complete_phrase["Phrase"] == random_phrase["Phrase"
]:
                            #checks if the phrases are the same
                            if k == len(total_vids)-1:#if its the last phrase
                                random_phrase = choice(total_vids)
                                #uses a random phrase for the next phrase
                            else:
                                break#uses the next available phrase
                        break
                random_phrase.update({"Confidence":0})
                #inserts confidence key:value into the dictionary
                template_list.append(random_phrase)
                #adds the new phrase to be learnt to the template list
                user_object.learning_path.append(random_phrase)
                #adds the new phrase to be learnt to the user's history
                for i in range(5):
                    tailored_list.append(random_phrase)
                    #adds the new phrase to the practise list - to be practise
d 5 times
                shuffle(tailored_list)
                #shuffles the list so the same phrases aren't adjacent
            else:
                del tailored_list[0]
                #deletes the most phrase that was answered right from the queu
e
            break
    tailored_feedback_text.show()
    tailored_correct_button.hide()
    tailored_incorrect_button.hide()
    tailored_feedback_text.hide()
```

```
    tailored_iteration_button.show()
    tailored_reveal.hide()

def incorrect_tailored_consolidation():
    global template_list
    global tailored_list
    global user_object
    user_object.total_wrong += 1
    for value in template_list:
        if value["Phrase"] == tailored_list[0]["Phrase"] and value["Confidence
"]!=0:
            #finds phrase in template list and only adjusts if confidence >0
            value["Confidence"]-
= 1#adjusts the confidence value for the given phrase
            tailored_list.append(value)
            #adds same phrase to the back twice
            #due to decrease in confidence
            break
    del tailored_list[0]#allows iteration to next value
    tailored_list.append(value)
    shuffle(tailored_list)
    tailored_feedback_text.show()
    tailored_correct_button.hide()
    tailored_incorrect_button.hide()
    tailored_feedback_text.hide()
    tailored_iteration_button.show()
    tailored_reveal.hide()

def change_password():
    global total_users
    global user_object
    total_users = user_worksheet.get_all_records()
    password = settings_old_password_input.value
    init_password = str.encode(password)
    hashed_password = hashlib.sha224(init_password).hexdigest().strip("0")
    both_equal = False
    for i in range(len(total_users)):
        if total_users[i]["Password"] == hashed_password:
            if settings_new_password_input.value == settings_confirm_password_
input.value:
                both_equal = True
                new_password = str.encode(settings_new_password_input.value)
                hashed_new_password = hashlib.sha224(new_password).hexdigest()
.strip("0")
                user_object.password = hashed_new_password
                updating_row()
                settings_old_password_input.value = ""
                settings_new_password_input.value = ""
```

```
                settings_confirm_password_input.value = ""
                settings_password_error.value = "Password changed successfully
"
        elif i == len(total_users)-1:
            settings_password_error.value = "Please enter the correct original
 password"
    if not both_equal:
        settings_password_error.value = "Please enter the same new password"

#GUI page switching procedures
def quit_program():
    app.destroy()

def home_page_login():
    username = home_page_username_input.value
    password = home_page_password_input.value
    global signed_in
    if not signed_in:
        signed_in = existing_user_login(username,password)
    if signed_in:
        home_page_to_loggedin()
    else:
        #if it cannot sign in it will display an error message to the user
        home_page_login_error.show()

def home_page_to_loggedin():
    home_page.hide()
    home_loggedin_page.show()

def loggedin_to_search():
    home_loggedin_page.hide()
    search_page.show()

def loggedin_to_settings():
    home_loggedin_page.hide()
    settings_page.show()
    adjusted_name = user_object.name.title()
    settings_name.value = "Name: " + str(adjusted_name)
    settings_username.value = "Username: " + str(user_object.username)
    settings_date_joined.value = "Date joined: " + str(user_object.date_joined
)
    settings_email.value = "Email: " + str(user_object.email)
    settings_total_right.value = "Total right answers: " + str(user_object.tot
al_right)
    settings_total_wrong.value = "Total wrong answers: " + str(user_object.tot
al_wrong)

def settings_to_loggedin():
```

```
    settings_page.hide()
    home_loggedin_page.show()
    settings_old_password_input.value = ""
    settings_new_password_input.value = ""
    settings_confirm_password_input.value = ""
    settings_password_error.value = ""

def clear_search_page():
    del letter_list[:]
    del letter_list[:]
    search_engine.value = ""
    alphabet_output.value = ""
    output_text.value = ""

def clear_home_page():
    home_page_username_input.value = ""
    home_page_password_input.value = ""
    home_page_login_error.hide()

def clear_signup_page():
    signup_name_input.value = ""
    signup_username_input.value = ""
    signup_password_input.value = ""
    signup_email_input.value = ""

def clear_confirmation_page():
    confirmation_code_box.value = ""

def clear_tailored_learning():
    tailored_phrase_text.hide()
    tailored_gif_output.hide()
    tailored_iteration_button.hide()
    tailored_reveal.hide()
    tailored_feedback_text.hide()
    tailored_correct_button.hide()
    tailored_incorrect_button.hide()
    tailored_phrase_then_desc.show()
    tailored_desc_then_phrase.show()
    tailored_consolidation_options.show()

def clear_consolidation():
    del consolidation_list[:]
    consolidation_phrase_text.hide()
    consolidation_finish_text.hide()
    consolidation_reveal_desc.hide()
    consolidation_iterate_button.hide()
    consolidation_gif_output.hide()
```

```
def home_to_search():
    home_page.hide()
    search_page.show()
    clear_home_page()

def search_to_home():
    global signed_in
    if signed_in:
        search_page.hide()
        home_loggedin_page.show()
        clear_search_page()
    else:
        search_page.hide()
        home_page.show()
        clear_search_page()

def home_to_signup():
    home_page.hide()
    sign_up_page.show()
    clear_home_page()

def signup_to_home():
    sign_up_page.hide()
    home_page.show()
    clear_signup_page()

def home_to_consolidation():
    home_page.hide()
    consolidation_page.show()

def consolidation_to_home():
    consolidation_page.hide()
    home_page.show()
    clear_consolidation()

def loggedin_to_tailored_consolidation():
    home_loggedin_page.hide()
    tailored_consolidation_page.show()

def tailored_consolidation_to_loggedin():
    tailored_consolidation_page.hide()
    home_loggedin_page.show()
    clear_tailored_learning()
    global template_list
    for i in range(len(user_object.learning_path)):
        for j in range(len(template_list)):
            if user_object.learning_path[i]["Phrase"] == template_list[j]["Phr
ase"]:
```

```
                user_object.learning_path[i]["Confidence"] = template_list[j][
"Confidence"]
                break

def search_result_to_search():
    search_result_page.hide()
    search_page.show()
    search_result_gif_output.value = None

def search_to_search_result():
    search_page.hide()
    search_result_page.show()
    signing_page_button.visible = False
    gif_setup()

def sign_up_to_confirmation():
    sign_up_code,passed = send_confirmation_email()
    if not passed:
        signup_output_text.value = "Please enter valid values into each box"
    username = signup_username_input.value
    email = signup_email_input.value
    for profile in total_users:
        if profile["Username"] == username:
            passed = False
            signup_output_text.value = "This username has been taken. Please t
ry another"
        elif profile["Email"] == email:
            passed = False
            signup_output_text.value = "There is already an account for this e
mail. Please try a different email or login"
    if passed:#passed variable is true if the email was sent
        sign_up_page.hide()
        confirmation_holder.value = sign_up_code
        confirmation_page.show()

def confiramtion_to_sign_up():
    sign_up_page.show()
    confirmation_page.hide()
    confirmation_output_text.value = ""

def confirmation_to_home():
    name = signup_name_input.value
    username = signup_username_input.value
    password = signup_password_input.value
    init_password = str.encode(password)
    hashed_password = hashlib.sha224(init_password).hexdigest().strip("0")
    email = signup_email_input.value
    creating_user_object(name,username,hashed_password,email)
```

```
    add_user_to_database(user_object)
    signed_in = True
    confirmation_page.hide()
    home_page.show()
    clear_confirmation_page()
    clear_signup_page()
    home_page_to_loggedin()


set_user_counter()
#creating GUI
app = App(title="Consolidation Dev",width=1400,height=700,layout="grid",bg=(99
,207,237))

#home page objects
home_page = Box(app,layout="grid",align="left",grid=[0,0],visible=True)
home_page_welcome_text = Text(home_page,text="Welcome to the sign language tra
nslator",grid=[0,0],color="white",align="left",size=20)
home_page_login_text = Text(home_page,text="Login Here: ",grid=[2,2],color="wh
ite")
home_page_user_name_input_text = Text(home_page,text="Username: ",grid=[1,3],c
olor="white")
home_page_username_input = TextBox(home_page,width=25,grid=[2,3])
home_page_password_input_text = Text(home_page,text="Password: ",grid=[1,4],co
lor="white")
home_page_password_input = TextBox(home_page,width=25,grid=[2,4],hide_text=Tru
e)
home_page_login_button = PushButton(home_page,command=home_page_login,text="Lo
gin",grid=[3,5])
home_page_login_error = Text(home_page,text="Couldn't sign you in. Please try
again",color="white",grid=[2,5],visible=False)
home_page_spacer2 = Text(home_page,text="",grid=[2,6])
home_page_signup_text = Text(home_page,text="Don't have a login? Make one here
:",color="white",grid=[2,7])
home_page_signup_button = PushButton(home_page,command=home_to_signup,text="Si
gn up",grid=[3,7])
home_page_continue_text = Text(home_page,text="Continue to Search engine: ",co
lor="white",grid=[7,2])
home_page_searching_button = PushButton(home_page,command=home_to_search,text=
"Search",grid=[7,3])
home_page_consolidation_text = Text(home_page,text="Continue to Consolidation:
 ",color="white",grid=[8,2])
home_page_consolidation_button = PushButton(home_page,command=home_to_consolid
ation,text="Consolidation",grid=[8,3])
quit_button = PushButton(home_page,command=quit_program,text="QUIT",grid=[20,2
0])
#logged in homepage
home_loggedin_page = Box(app,layout="grid",align="left",grid=[0,0],visible=Fal
se)
```

```
home_loggedin_title = Text(home_loggedin_page,text="Welcome to the sign langua
ge translator",grid=[0,0],color="white",align="left",size=20)
home_loggedin_spacer1 = Text(home_loggedin_page,text="",grid=[0,1])
home_loggedin_search_text = Text(home_loggedin_page,text="Click here to contin
ue to Search page: ",color="white",grid=[0,2],align="left")
home_loggedin_search_button = PushButton(home_loggedin_page,command=loggedin_t
o_search,text="Continue to Search",grid=[0,3],align="left")
home_loggedin_consolidation_text = Text(home_loggedin_page,text="Click here to
 continue to Consolidation page",color="white",grid=[4,2],align="left")
home_loggedin_consolidation_button = PushButton(home_loggedin_page,command=log
gedin_to_tailored_consolidation,text="Continue to Consolidation",grid=[4,3],al
ign="left")
home_loggedin_spacer2 = Text(home_loggedin_page,text="",grid=[0,5])
home_loggedin_settings_button = PushButton(home_loggedin_page,command=loggedin
_to_settings,text="Settings",grid=[10,10],align="left")
home_loggedin_logout_button = PushButton(home_loggedin_page,command=sign_out,t
ext="Log out",grid=[0,10],align="left")
home_loggedin_logout_text = Text(home_loggedin_page,text="Progress will auto-
save when you logout",grid=[0,9],color="white",align="left")
#signup page objects
sign_up_page = Box(app,layout="grid",align="left",grid=[0,0],visible=False)
sign_up_title = Text(sign_up_page,text="Sign Up",size=20,color="white",grid=[0
,0])
signup_to_home_button = PushButton(sign_up_page,command=signup_to_home,text="R
eturn Home",grid=[0,10],align="left")
signup_text = Text(sign_up_page,text="Sign up Here: ",grid=[1,2],color="white"
)
signup_name_input_text = Text(sign_up_page,text="Full Name: ",grid=[0,3],color
="white")
signup_name_input = TextBox(sign_up_page,width=25,grid=[1,3])
signup_user_name_input_text = Text(sign_up_page,text="Username: ",grid=[0,4],c
olor="white")
signup_username_input = TextBox(sign_up_page,width=25,grid=[1,4])
signup_password_input_text = Text(sign_up_page,text="Password: ",grid=[0,5],co
lor="white")
signup_password_input = TextBox(sign_up_page,width=25,grid=[1,5],hide_text=Tru
e)
signup_email_input_text = Text(sign_up_page,text="Email: ",grid=[0,6],color="w
hite")
signup_email_input = TextBox(sign_up_page,width=25,grid=[1,6])
signup_button = PushButton(sign_up_page,command=sign_up_to_confirmation,text="
Sign up",grid=[2,7])
signup_output_text = Text(sign_up_page,text="",grid=[3,2],color="white")

#confirmation of signup objects
confirmation_page = Box(app,layout="grid",align="left",grid=[0,0],visible=Fals
e)
```

```
confirmation_title = Text(confirmation_page,text="A code has been sent to the
provided email",size=15,color="white",grid=[0,0],align="left")
confirmation_subtitle = Text(confirmation_page,text="Enter confirmation code",
color="white",grid=[0,1],align="right")
confirmation_code_box = TextBox(confirmation_page,grid=[1,1],width=25)
confirmation_enter_button = PushButton(confirmation_page,command=confirming_si
gn_up_code,text="Enter",grid=[2,1])
confirmation_output_text = Text(confirmation_page,text="",color="white",grid=[
0,3],align="right")
confirmation_continue_button = PushButton(confirmation_page,command=confirmati
on_to_home,text="Continue",grid=[1,3],visible=False)
confirmation_back_button = PushButton(confirmation_page,command=confiramtion_t
o_sign_up,text="Back",grid=[0,20])
confirmation_holder = Text(confirmation_page,text="",color="white",grid=[10,10
],align="right",visible=False)

#search page objects
search_page = Box(app,layout="grid",align="left",grid=[0,0],visible=False)
search_page_title = Text(search_page,text="Search Page",grid=[0,0],color="whit
e",size=20)
instructions = Text(search_page,text="Please enter a phrase to be searched:",g
rid=[1,1],color="white")
search_engine = TextBox(search_page,width=50,grid=[1,3])
enter_button = PushButton(search_page,command=search_phrase,text="Enter",grid=
[2,3])
mic_on = PushButton(search_page,command=mic_input,text="Mic on",grid=[1,4])
spacer = Text(search_page,text = "      ",grid=[7,1])
alphabet_text = Text(search_page,text="Alphabet line: ",grid=[8,1],color="whit
e")
alphabet = Combo(search_page,options=alphabet_list,command=alphabet_display,gr
id=[9,1])
alphabet_button = PushButton(search_page,command=alphabet_iteration,text="Sear
ch Letter",grid=[10,1],align="left")
alphabet_output = Text(search_page,text="output: ",grid=[10,3],align="left",co
lor="white")
output_text = Text(search_page,text="",grid=[1,5],color="white",align="left")
signing_page_button = PushButton(search_page,command=search_to_search_result,t
ext="Show Signing",grid=[2,5],visible=False)
spacer2 = Text(search_page,text="",grid=[1,6],color="white",align="left")
top_20_title = Text(search_page,text="Top 20 List: ",grid=[1,7],color="white",
align="left")
top_20_output_text = Text(search_page,text="",grid=[1,8],color="white",align="
left")
search_to_home_button = PushButton(search_page,command=search_to_home,text="Re
turn Home",grid=[0,10])

#search result objects
```

```
search_result_page = Box(app,layout="grid",align="left",grid=[0,0],visible=Fal
se)
search_result_title = Text(search_result_page,text="Search Result Page",grid=[
0,0],color="white",size=20)
search_result_user_input = Text(search_result_page,text="",color="white",grid=
[0,2],align="left",size=15)
search_result_gif_output = Picture(search_result_page,image=None,grid=[2,4])
search_result_next_button = PushButton(search_result_page,command=iterate_sear
ch_result,text="Next",grid=[4,4],visible=False)
search_result_to_search_button = PushButton(search_result_page,command=search_
result_to_search,text="Exit",grid=[0,20],align="left")
search_resut_current_output = Text(search_result_page,text="",color="white",gr
id=[0,3],align="left")

#consolidation of learning page
consolidation_page = Box(app,layout="grid",align="left",grid=[0,0],visible=Fal
se)
consolidation_page_title = Text(consolidation_page,text="Consolidation Page",g
rid=[0,0],color="white",size=20)
begin_practise_button = PushButton(consolidation_page,command=starting_consoli
dation,text="Begin Session",grid=[1,3])
begin_text = Text(consolidation_page,text="",color="white",grid=[0,3])
consolidation_to_home_button = PushButton(consolidation_page,command=consolida
tion_to_home,text="Exit",grid=[0,10],align="left")

consolidation_phrase_text = Text(consolidation_page,text="",grid=[2,5],color="
white",visible=False,size=15)
consolidation_gif_output = Picture(consolidation_page,image=None,grid=[2,7])
consolidation_iterate_button = PushButton(consolidation_page,command=iterate_c
onsolidation_output,text="Next signing",grid=[4,7],visible=False)
consolidation_reveal_desc = PushButton(consolidation_page,command=show_consoli
dation_phrase,text="Reveal phrase",grid=[2,8],visible=False)
consolidation_finish_text = Text(consolidation_page,text="To restart practisin
g press the Begin Session button",color="white",grid=[2,9],visible=False)

#tailored consolidation page
tailored_consolidation_page = Box(app,layout="grid",align="left",grid=[0,0],vi
sible=False)
tailored_consolidation_title = Text(tailored_consolidation_page,text="Tailored
 Consolidation",grid=[0,0],color="white",size=20)
tailored_consolidation_options = Text(tailored_consolidation_page,text="Please
 choose the method of practise: ",color="white",grid=[0,1])
tailored_phrase_then_desc = PushButton(tailored_consolidation_page,command=tai
lored_predict_desc,text="Predict the description",grid=[1,1])
tailored_desc_then_phrase = PushButton(tailored_consolidation_page,command=tai
lored_predict_phrase,text="Predict the phrase",grid=[2,1])
tailored_phrase_text = Text(tailored_consolidation_page,text="",color="white",
grid=[4,2])
```

```
tailored_gif_output = Picture(tailored_consolidation_page,image=None,grid=[4,4
])
tailored_iteration_button = PushButton(tailored_consolidation_page,command=tai
lored_consolidation,text="Next signing",grid=[5,4],visible=False)
tailored_reveal = PushButton(tailored_consolidation_page,command=show_tailored
_phrase,text="Reveal",grid=[4,5],visible=False)
tailored_feedback_text = Text(tailored_consolidation_page,text="Did you guess
correctly? ",color="white",grid=[4,6],align="right",visible=False)
tailored_correct_button = PushButton(tailored_consolidation_page,command=corre
ct_tailored_consolidation,text="Right",grid=[5,6],visible=False)
tailored_incorrect_button = PushButton(tailored_consolidation_page,command=inc
orrect_tailored_consolidation,text="Wrong",grid=[6,6],visible=False)
tailored_consolidation_home = PushButton(tailored_consolidation_page,command=t
ailored_consolidation_to_loggedin,text="Home",grid=[0,20],align="left")

#settings page
settings_page = Box(app,layout="grid",align="left",grid=[0,0],visible=False)
settings_title = Text(settings_page,text="Settings Page",grid=[0,0],color="whi
te",size=20,align="left")
settings_home_button = PushButton(settings_page,command=settings_to_loggedin,t
ext="Home",grid=[0,20])
settings_name = Text(settings_page,text="Name: ",grid=[0,2],color="white",alig
n="left")
settings_username = Text(settings_page,text="Username: ",grid=[0,4],color="whi
te",align="left")
settings_date_joined = Text(settings_page,text="Date Joined: ",grid=[0,6],colo
r="white",align="left")
settings_email = Text(settings_page,text="Email: ",grid=[0,8],color="white",al
ign="left")
settings_spacer = Text(settings_page,text="                              ",grid=[
1,1])

settings_old_password_text = Text(settings_page,text="Enter old password here:
 ",color="white",grid=[2,2],align="left")
settings_old_password_input = TextBox(settings_page,grid=[3,2],hide_text=True,
width=30)
settings_new_password_text = Text(settings_page,text="Enter new password here:
 ",color="white",grid=[2,4],align="left")
settings_new_password_input = TextBox(settings_page,grid=[3,4],hide_text=True,
width=30)
settings_confirm_password_text = Text(settings_page,text="Confirm new password
: ",color="white",grid=[2,6],align="left")
settings_confirm_password_input = TextBox(settings_page,grid=[3,6],hide_text=T
rue,width=30)
settings_change_password_button = PushButton(settings_page,command=change_pass
word,text="Change password",grid=[4,6])
settings_password_error = Text(settings_page,text="",color="white",grid=[2,8])
```

```
settings_total_right = Text(settings_page,text="Total right answers: ",color="
white",grid=[2,10],align="left")
settings_total_wrong = Text(settings_page,text="Total wrong answers: ",color="
white",grid=[2,12],align="left")

app.display()
```